Algebraic Program Analysis of Probabilistic Programs

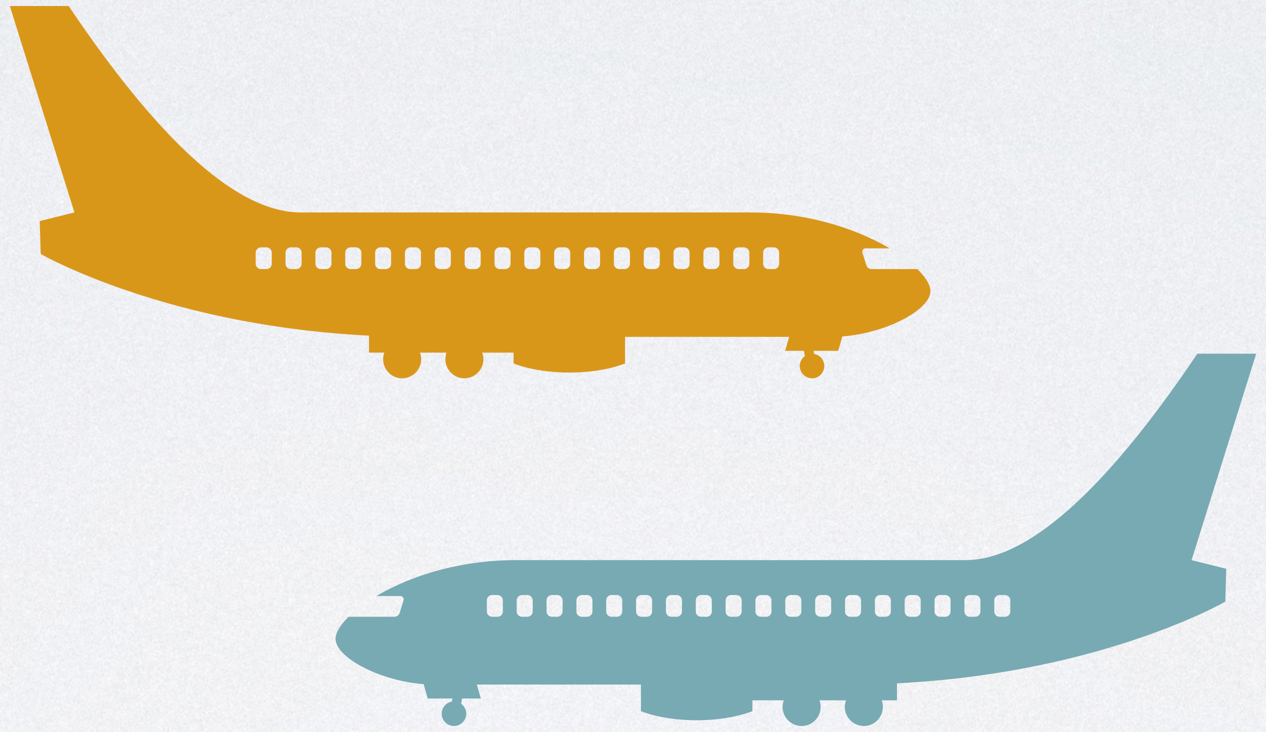
Di Wang
Peking University
wangdi95@pku.edu.cn

Apr 10, 2024

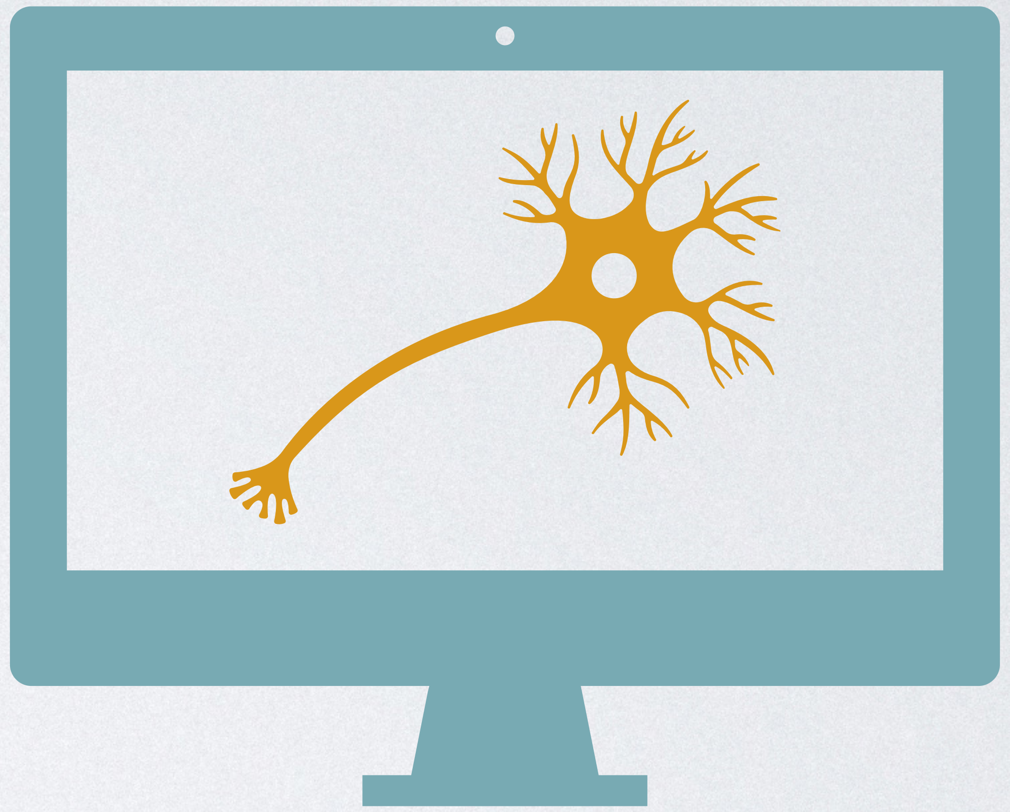
Joint Work with **Jan Hoffmann** and **Thomas Reps**

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

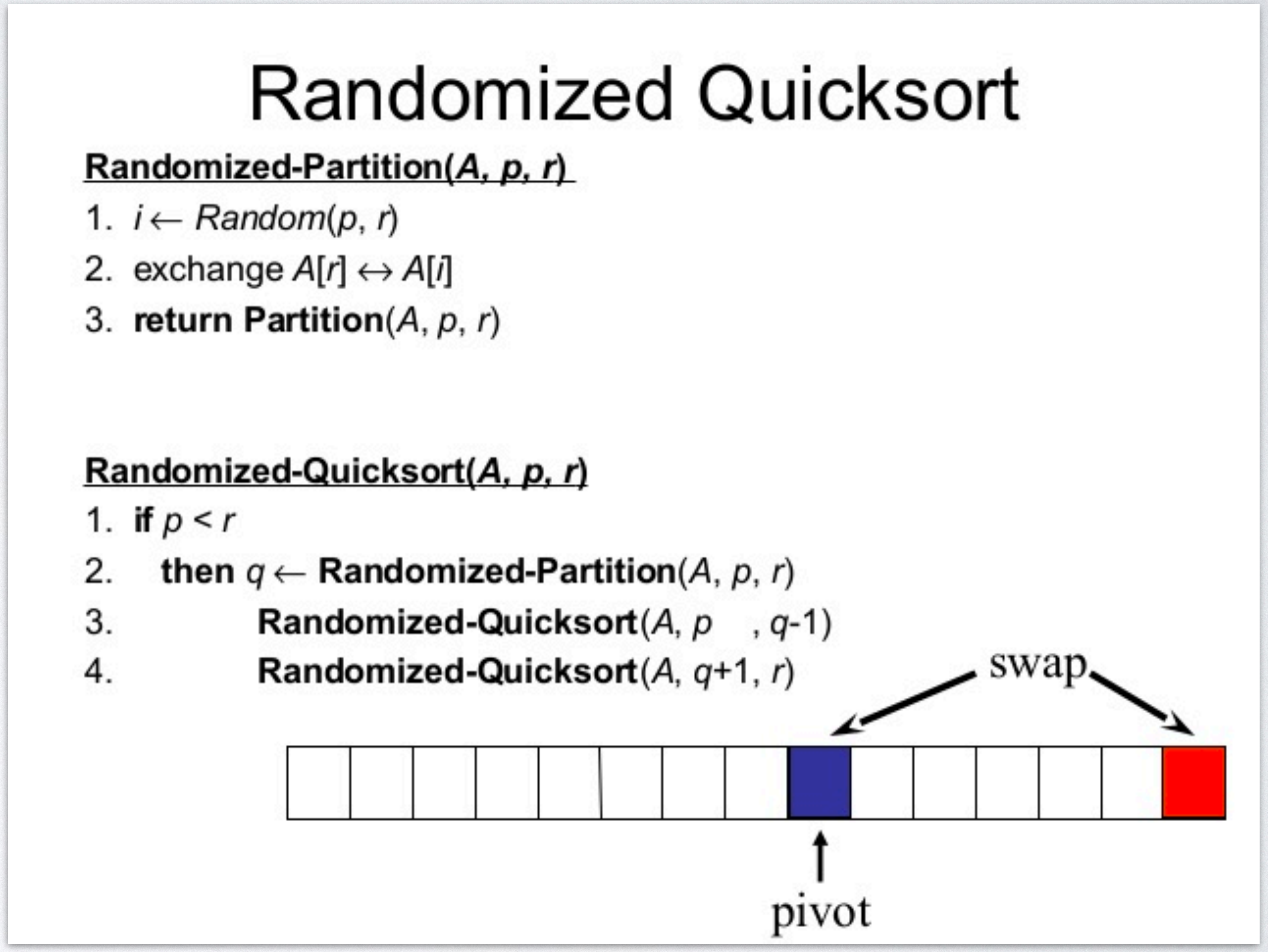


Image source: https://geekfactorial.blogspot.com/2016/08/randomized-quick-sort-algorithm.html.

Application: Airborne Collision Avoidance

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

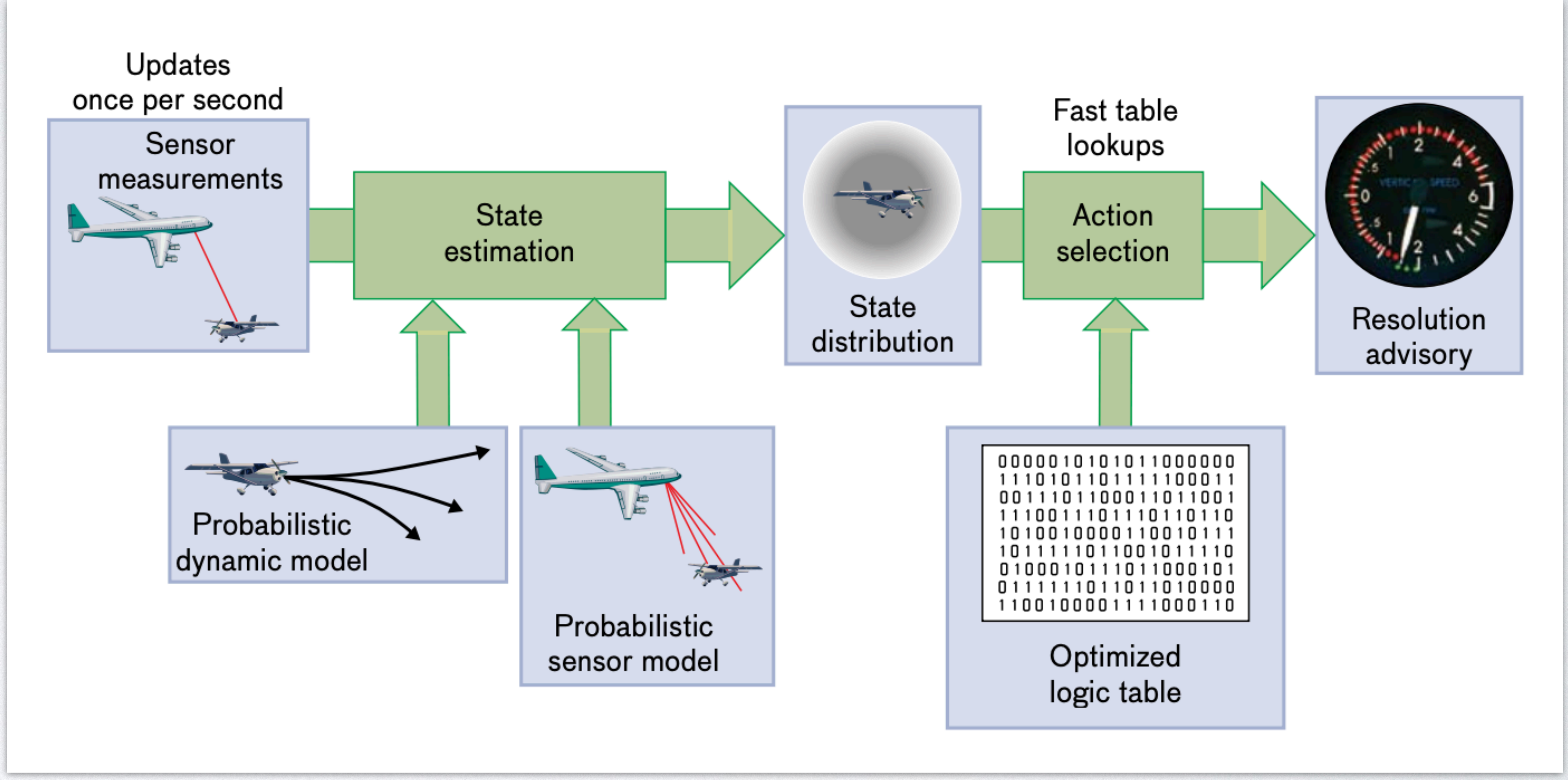
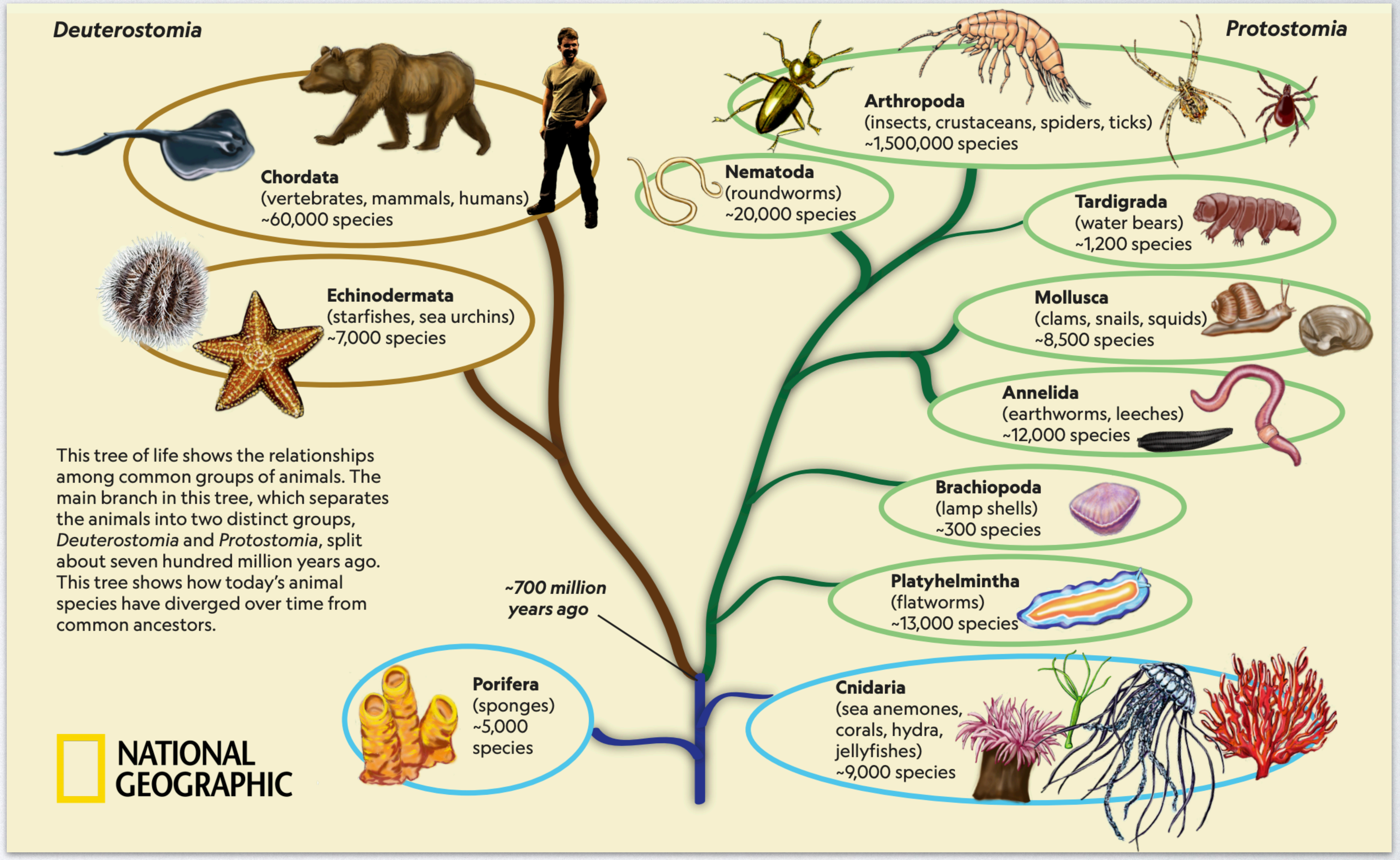


Image source: Kochenderfer, et al. "Next-Generation Airborne Collision Avoidance System." Lincoln Laboratory Journal (2012).

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

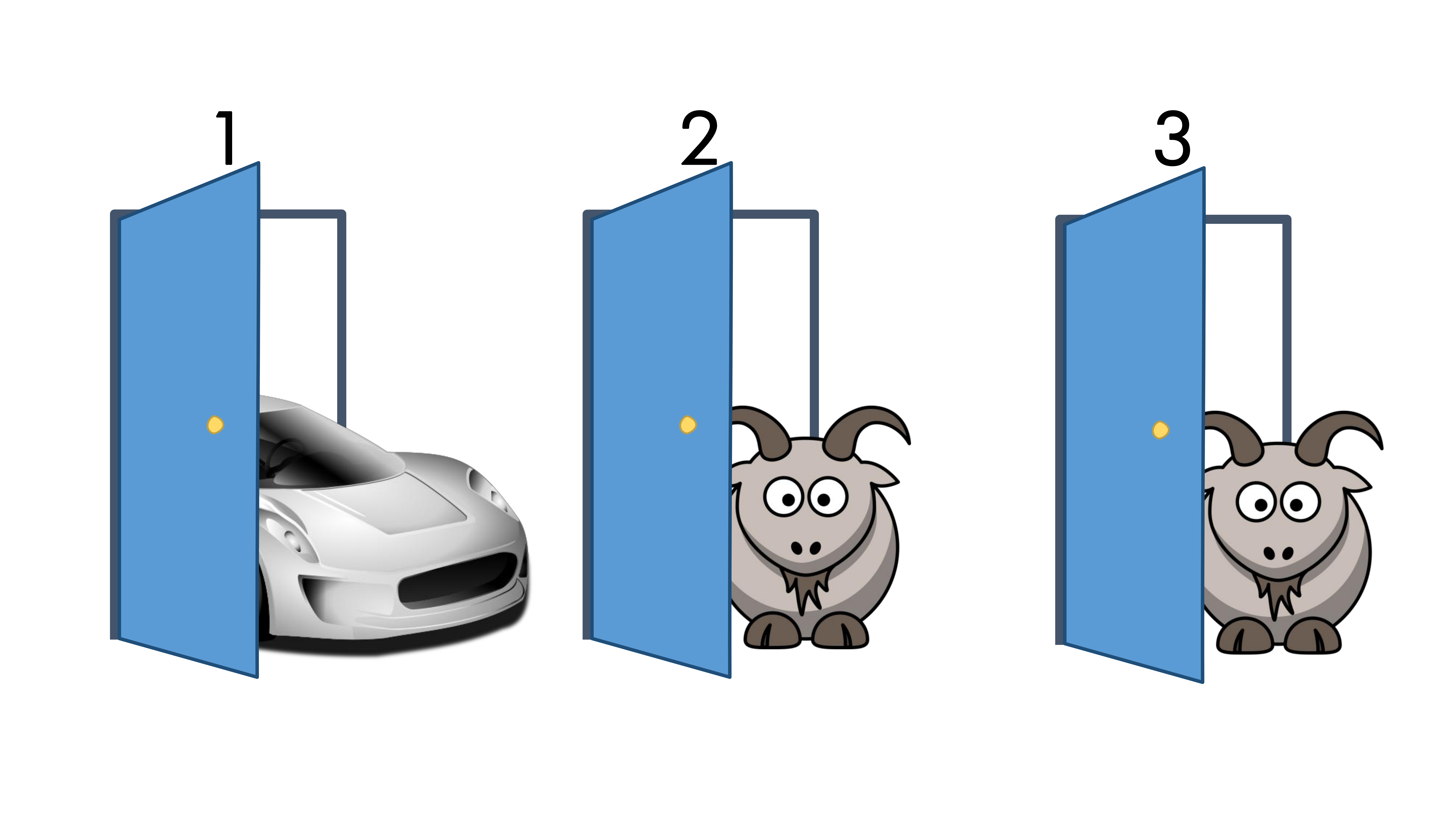


Image source: Maria Gorinova's Advances in Programming Languages (Guest Lecture) slides on Probabilistic Programming.

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
```

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

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
```

Do Kleene Algebras Suffice?

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

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

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

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

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
```

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

Do Kleene Algebras Suffice?

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

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

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

Do Kleene Algebras Suffice?

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

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

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

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**

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

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Our Approach: Markov Algebras

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

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

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**

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

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Easy to extend with more confluence operations!

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!

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

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

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

$$a \sqcap a = a$$

...

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
```

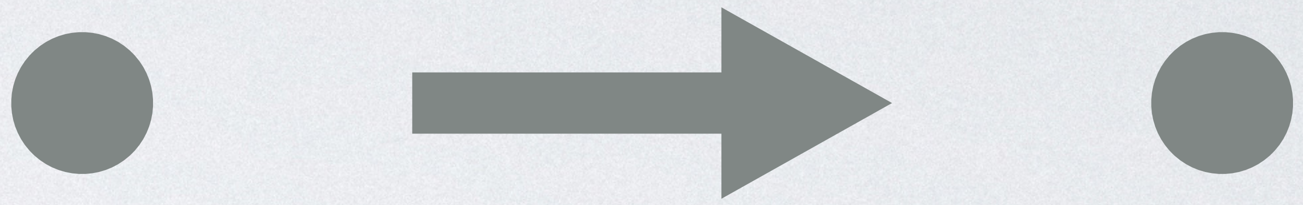
$$\mu S \,.\, ((x := x{+}1 \,{}_{1/2}\!\oplus\, x := x{-}1) \otimes S) \,{}_{[x>0]}\!\oplus\, \mathbf{skip}$$

Recursive Program Scheme

Flexibility: Different Semantics

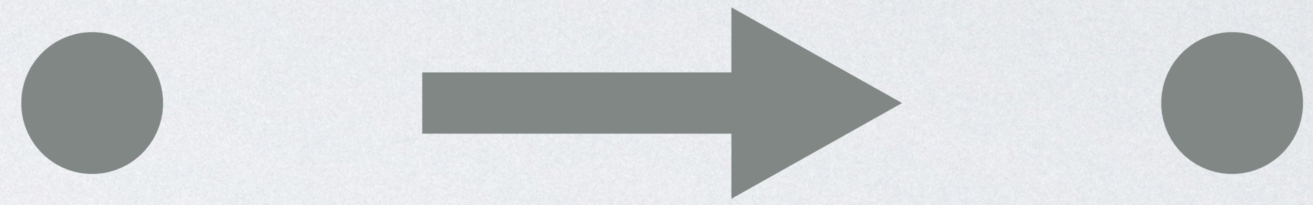
Flexibility: Different Semantics

Standard: $State \rightarrow State$



Flexibility: Different Semantics

Standard: $State \rightarrow State$



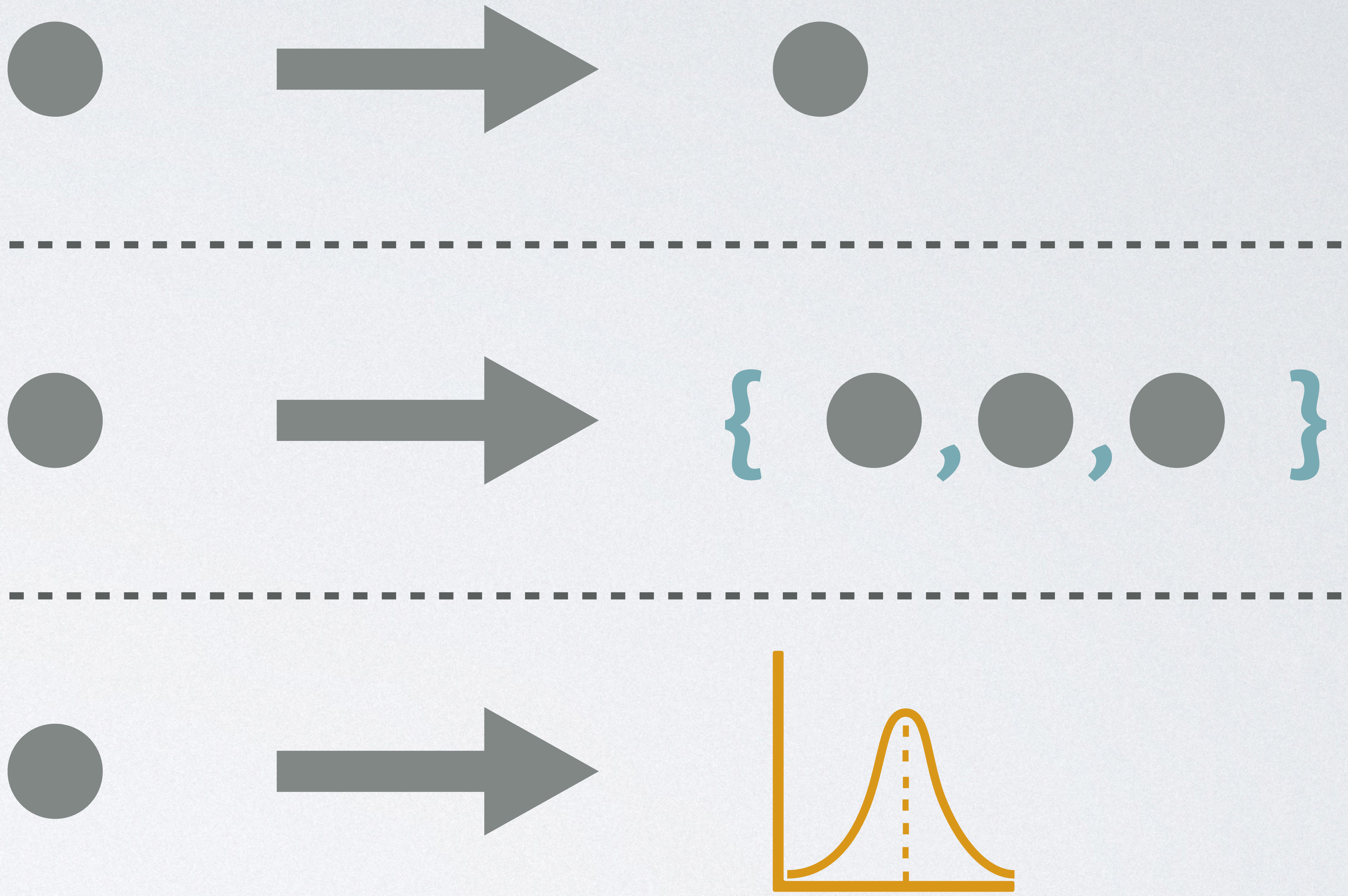
GCL: $State \rightarrow \wp(State)$

Flexibility: Different Semantics

Standard: $State \to State$

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

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



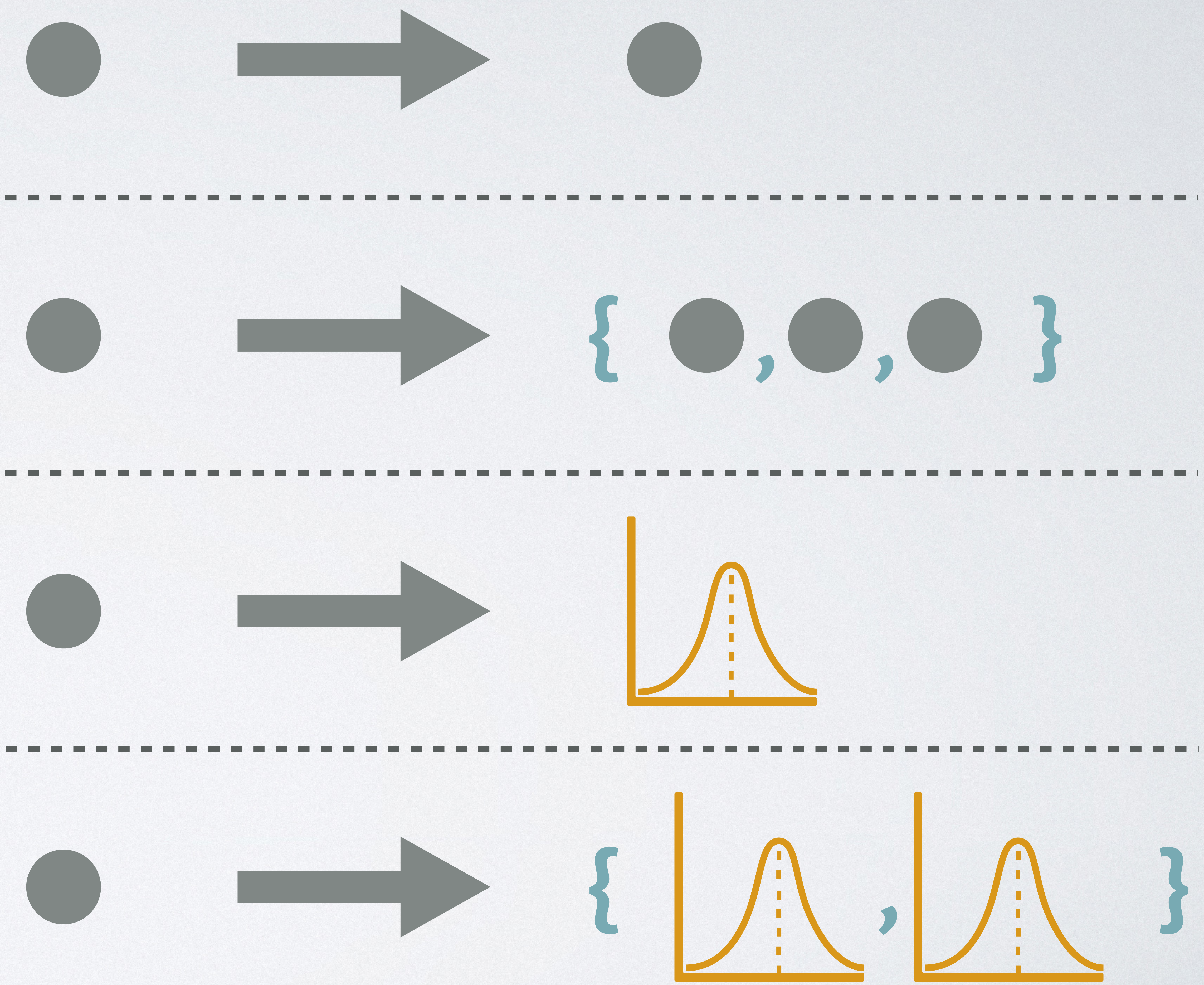
Flexibility: Different Semantics

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

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

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

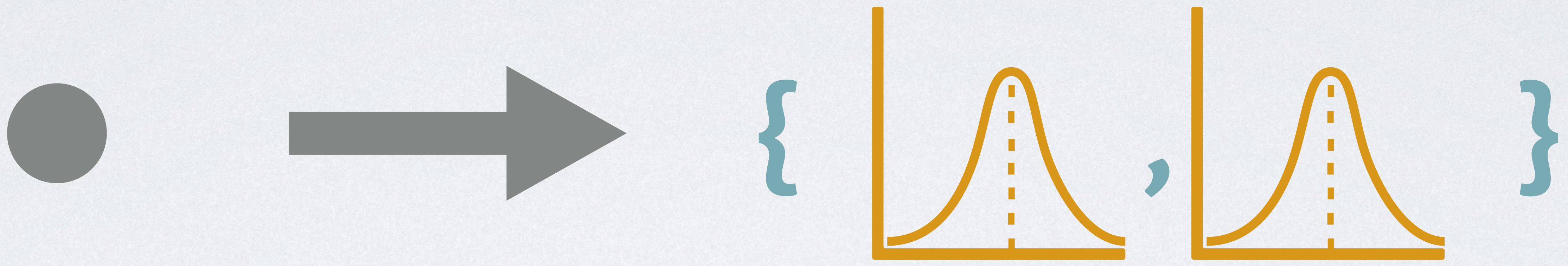
pGCL: $\mathit{State} \to \wp(\mathbb{D}(\mathit{State}))$



Flexibility: Different Models for Nondeterminism

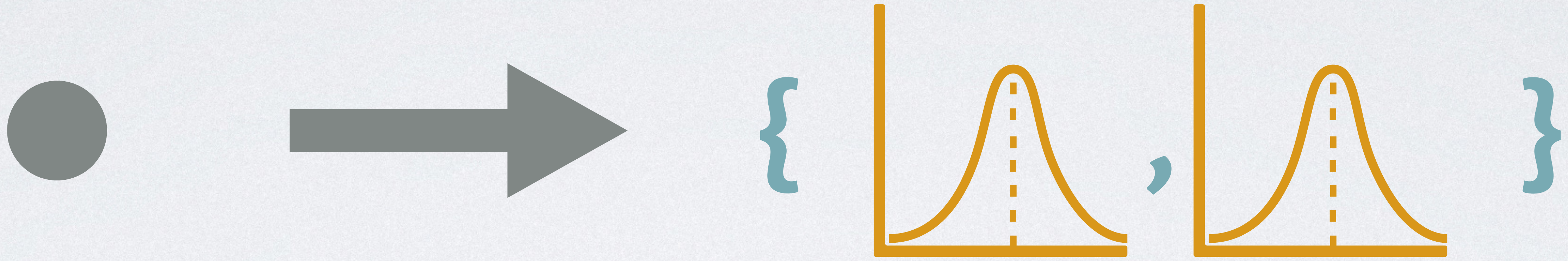
Flexibility: Different Models for Nondeterminism

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

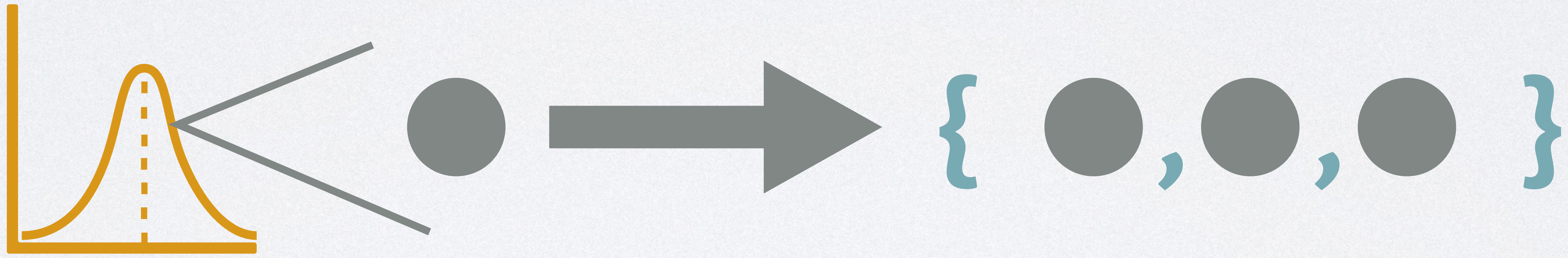


Flexibility: Different Models for Nondeterminism

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

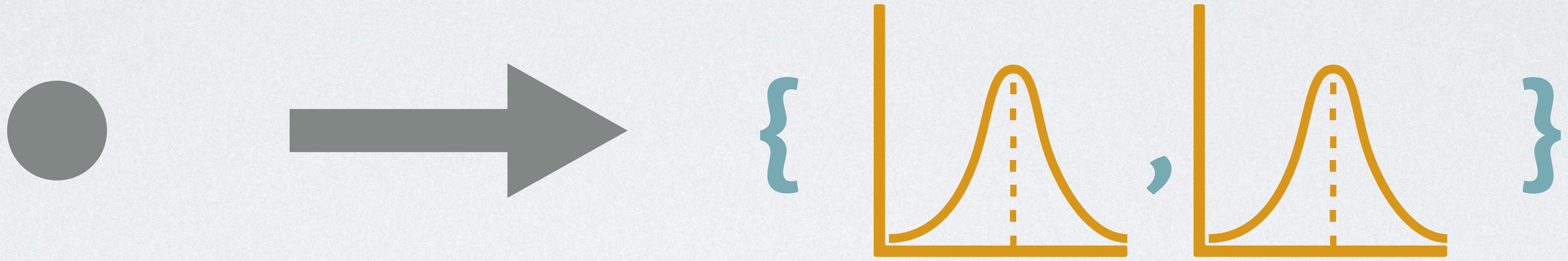


- Cousot's **Probabilistic Abstract Interpretation** (PAI): $\mathbb{D}(\mathit{State} \to \wp(\mathit{State}))$

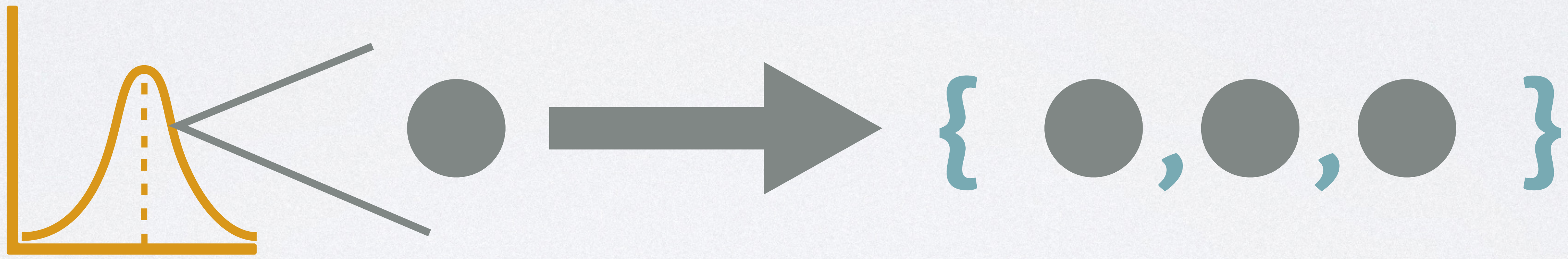


Flexibility: Different Models for Nondeterminism

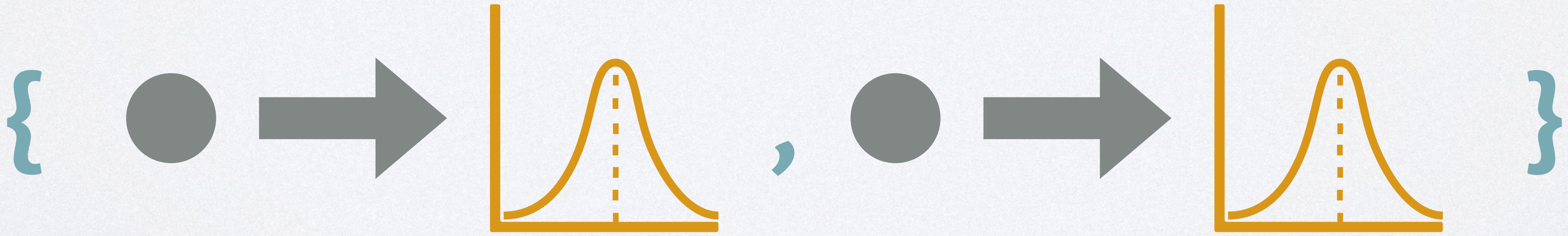
- pGCL: $\mathit{State} \to \wp(\mathbb{D}(\mathit{State}))$



- 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
```

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

A Control-Flow-Graph's Perspective

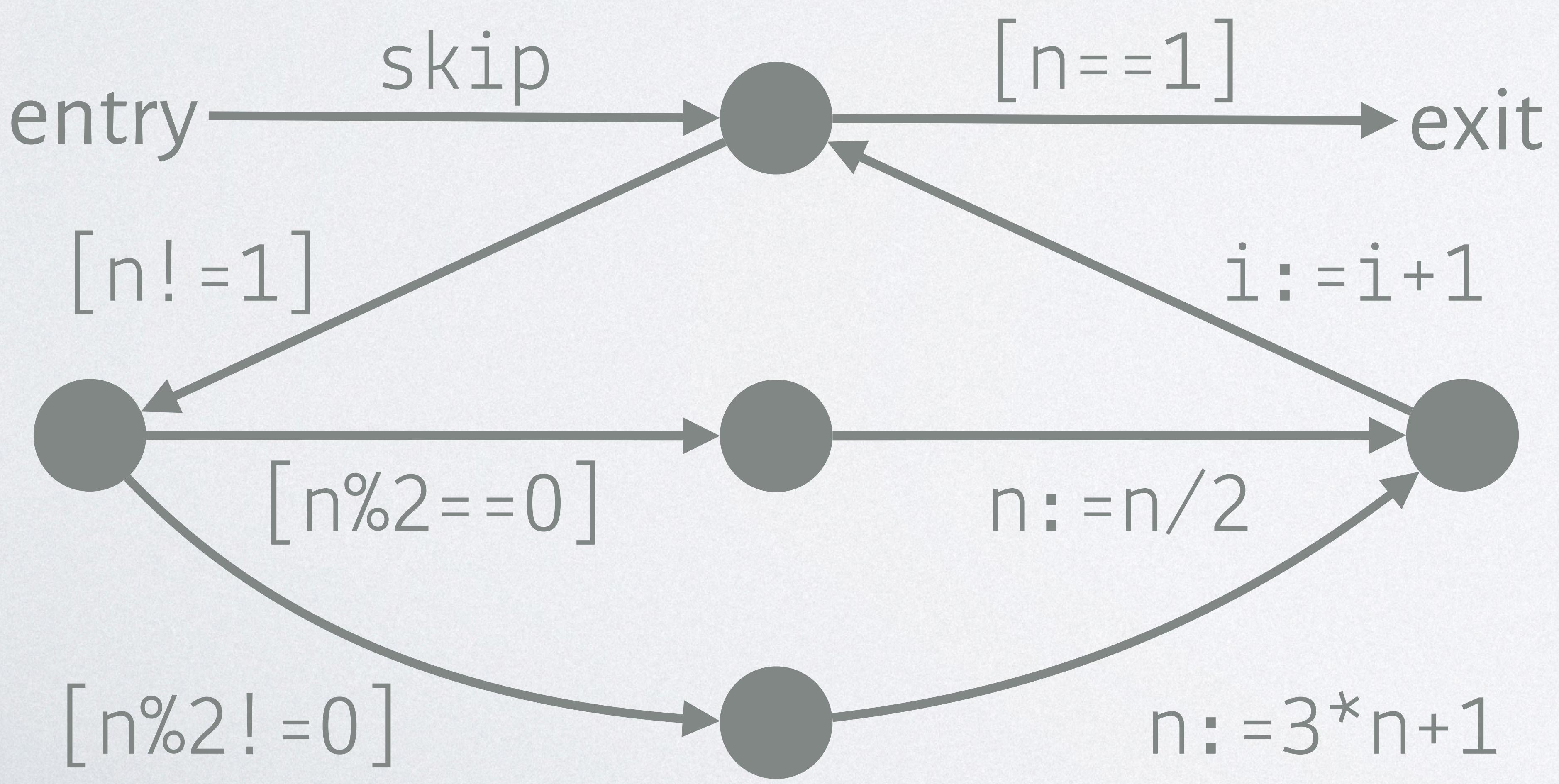
- **Kleene Algebras** are compatible with **control-flow graphs** via **regular expressions**

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**

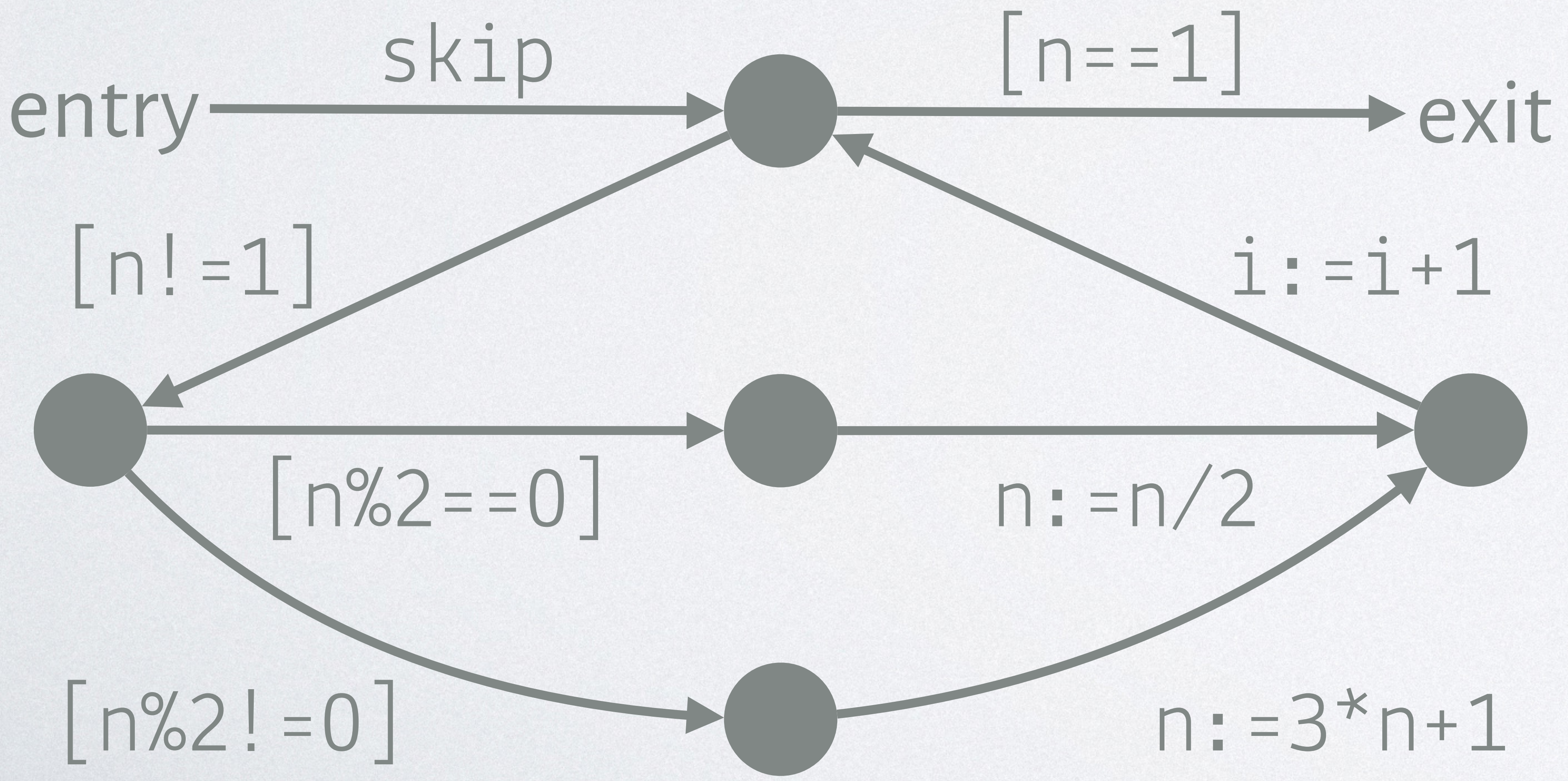
Program Construct	Algebraic Representation
A program S	An interpretation $\mathbb{S}$ of S into the algebra
The **control-flow graph** of S	A **regular expression** over $\underline{0}$, $\underline{1}$, $\oplus$, $\otimes$, and $*$



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 $*$



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

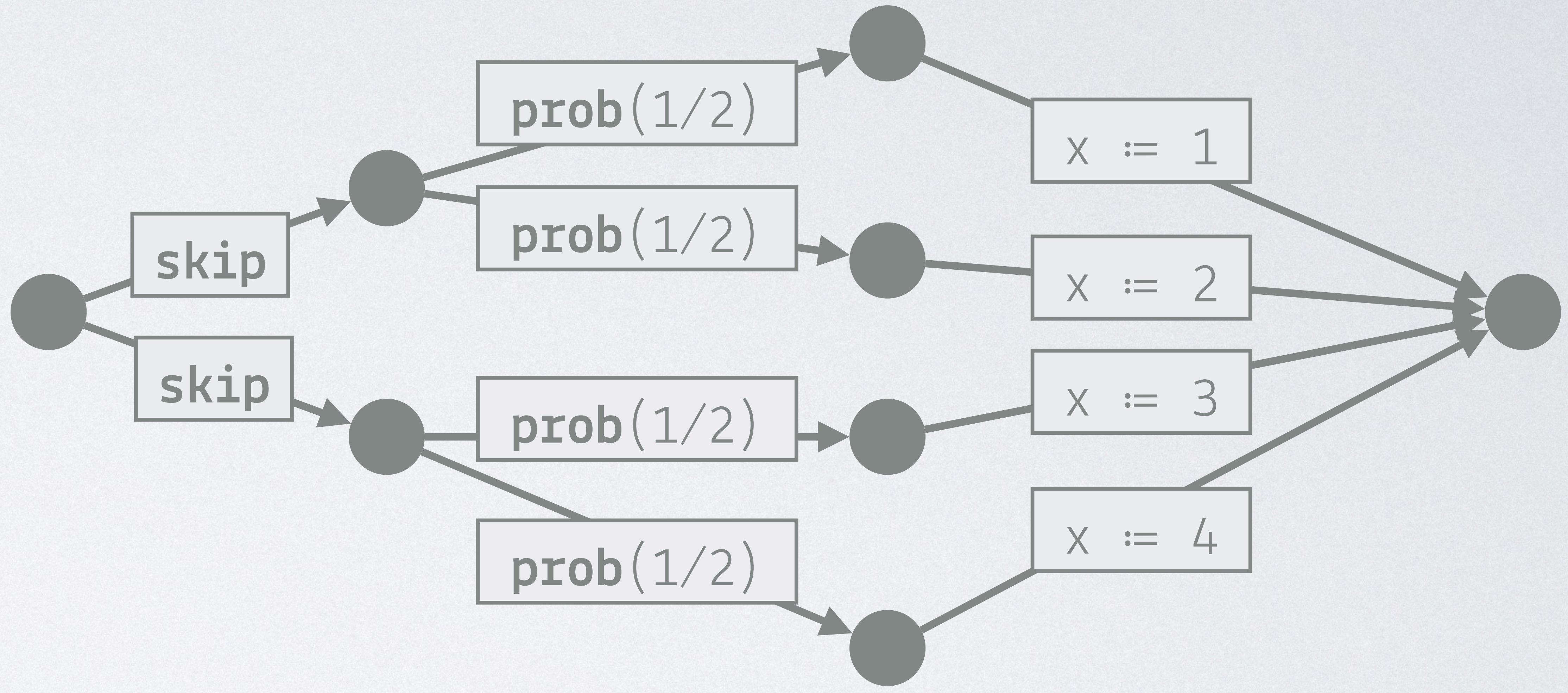
Do Control-Flow Graphs Suffice?

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
```

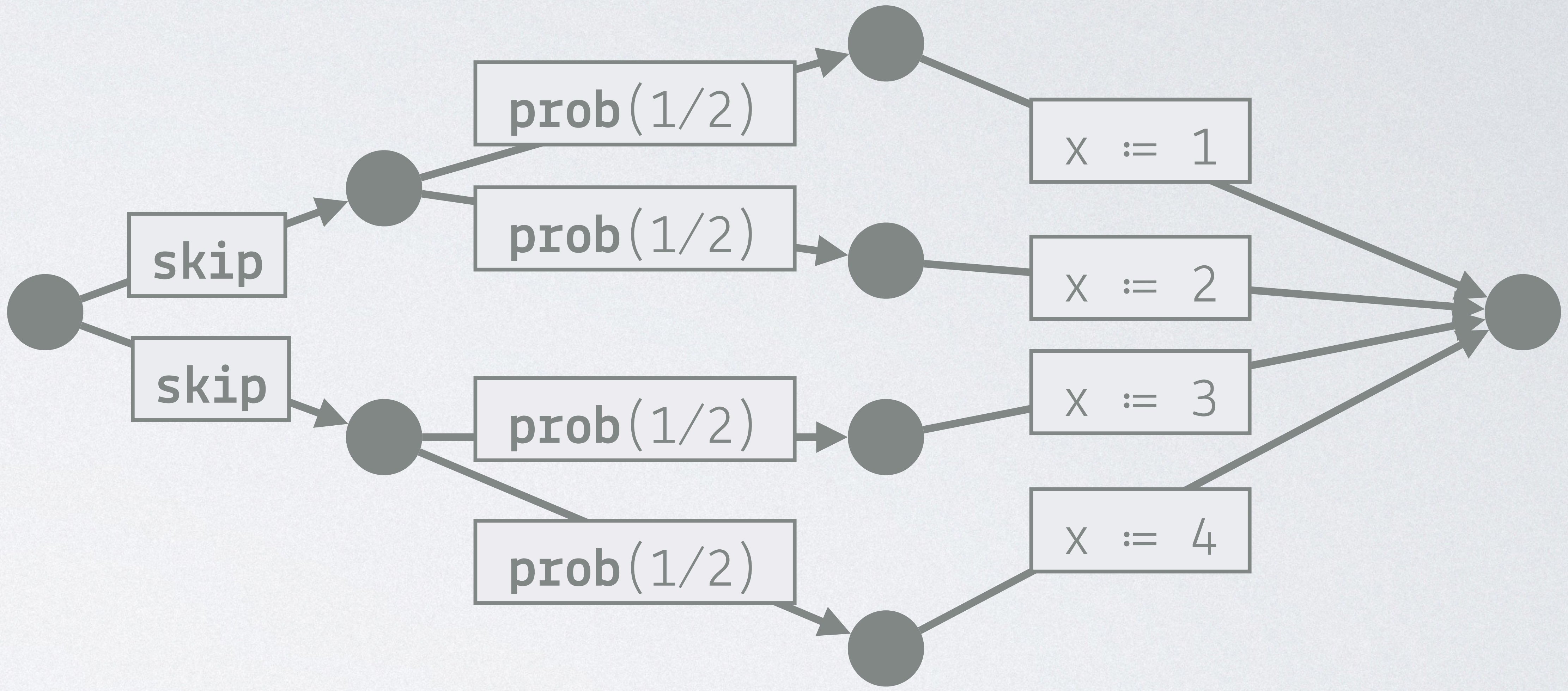
Do Control-Flow Graphs Suffice?

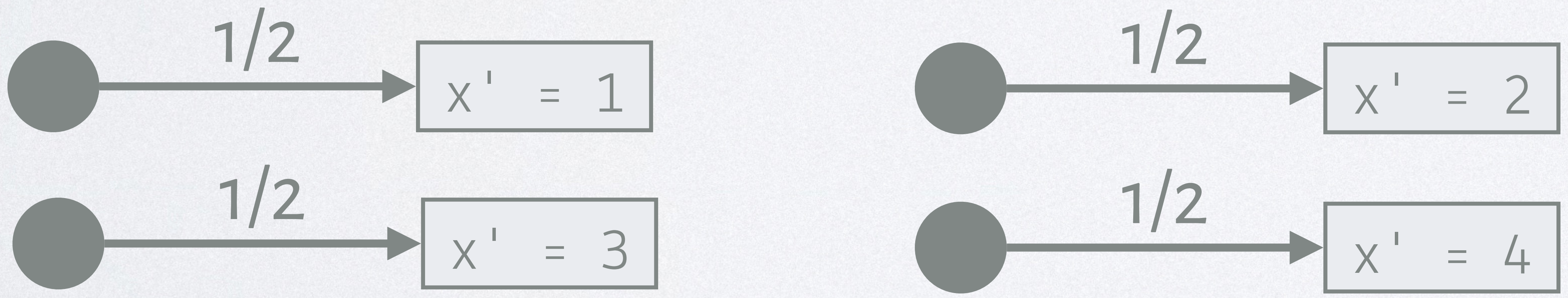
```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```



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
```

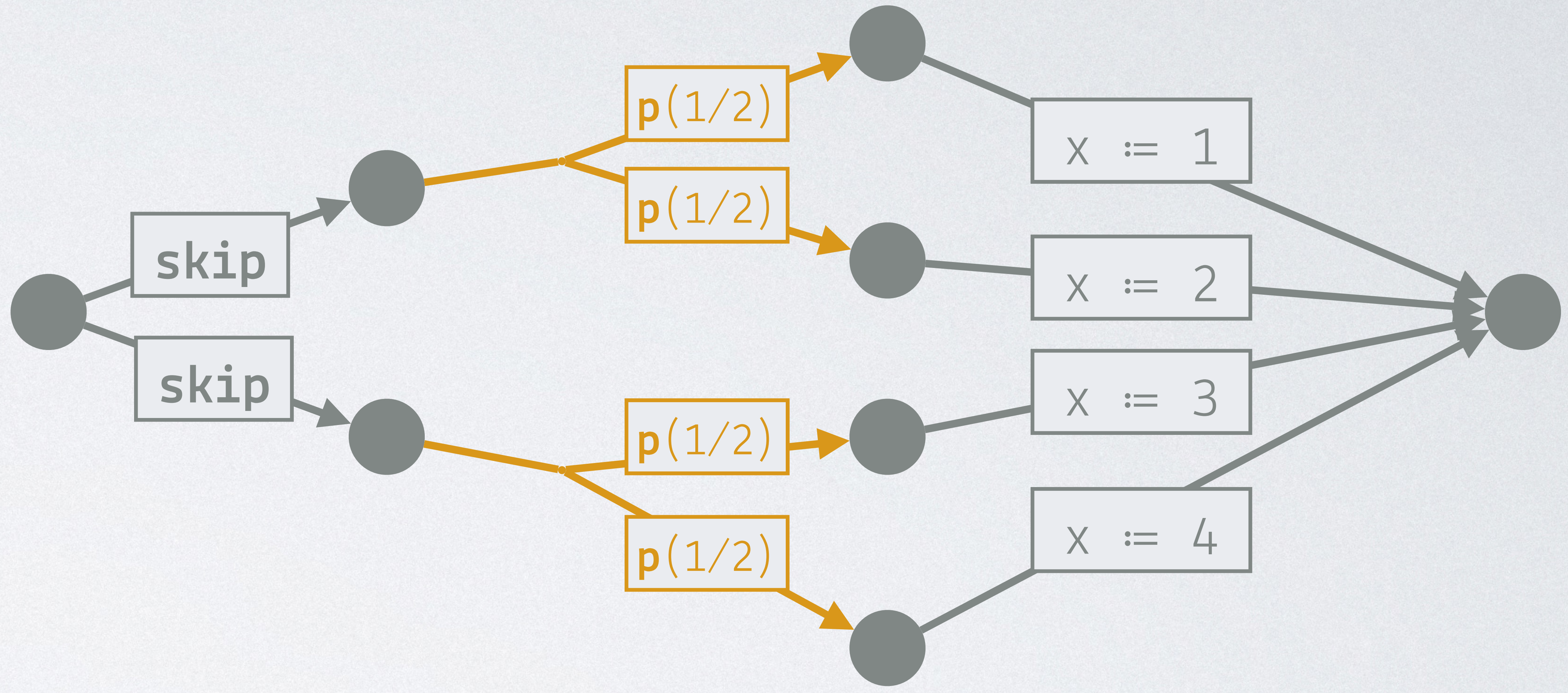




Probabilities sum up to 2!

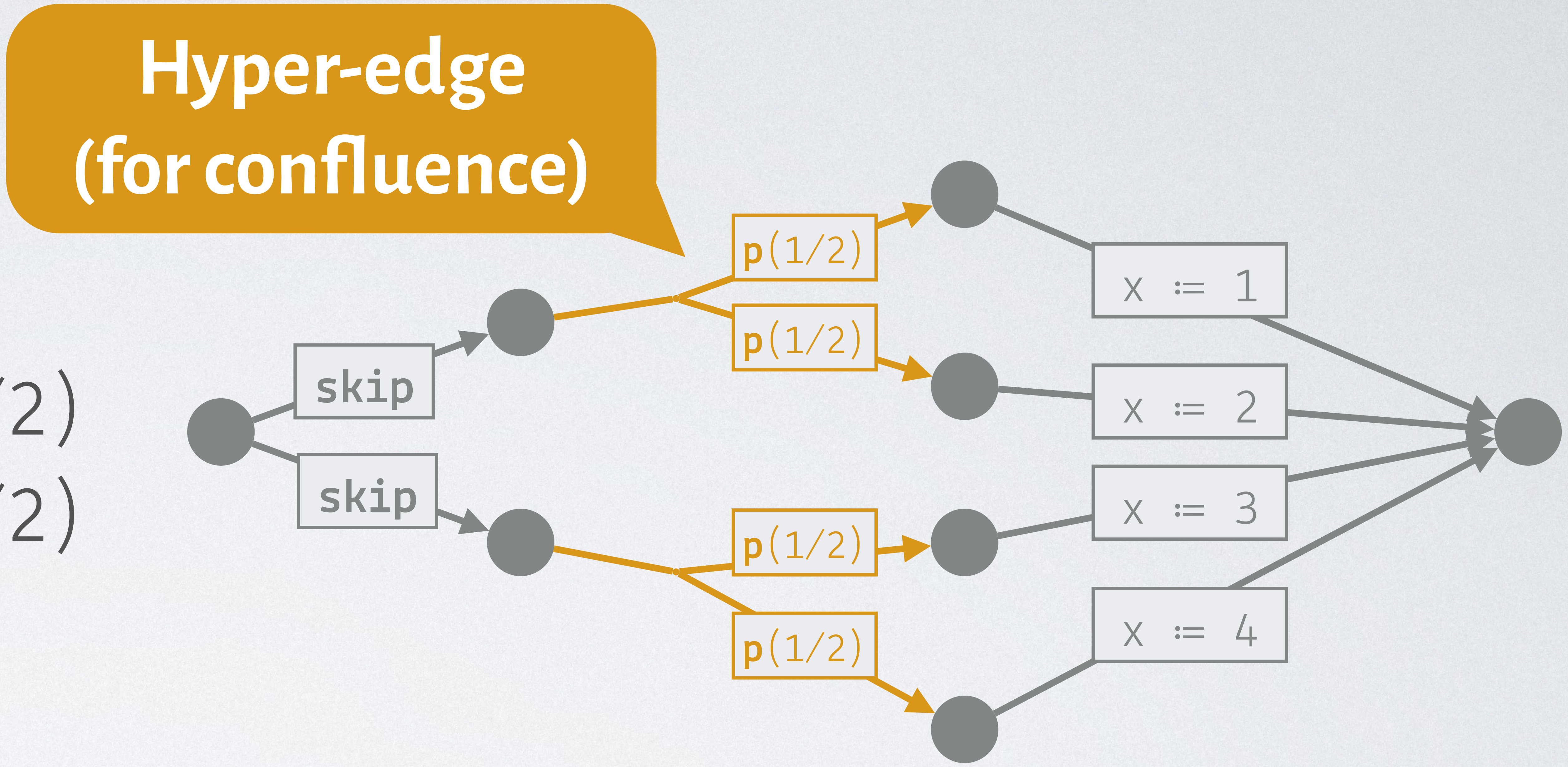
Our Approach: Control-Flow Hyper-Graphs

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



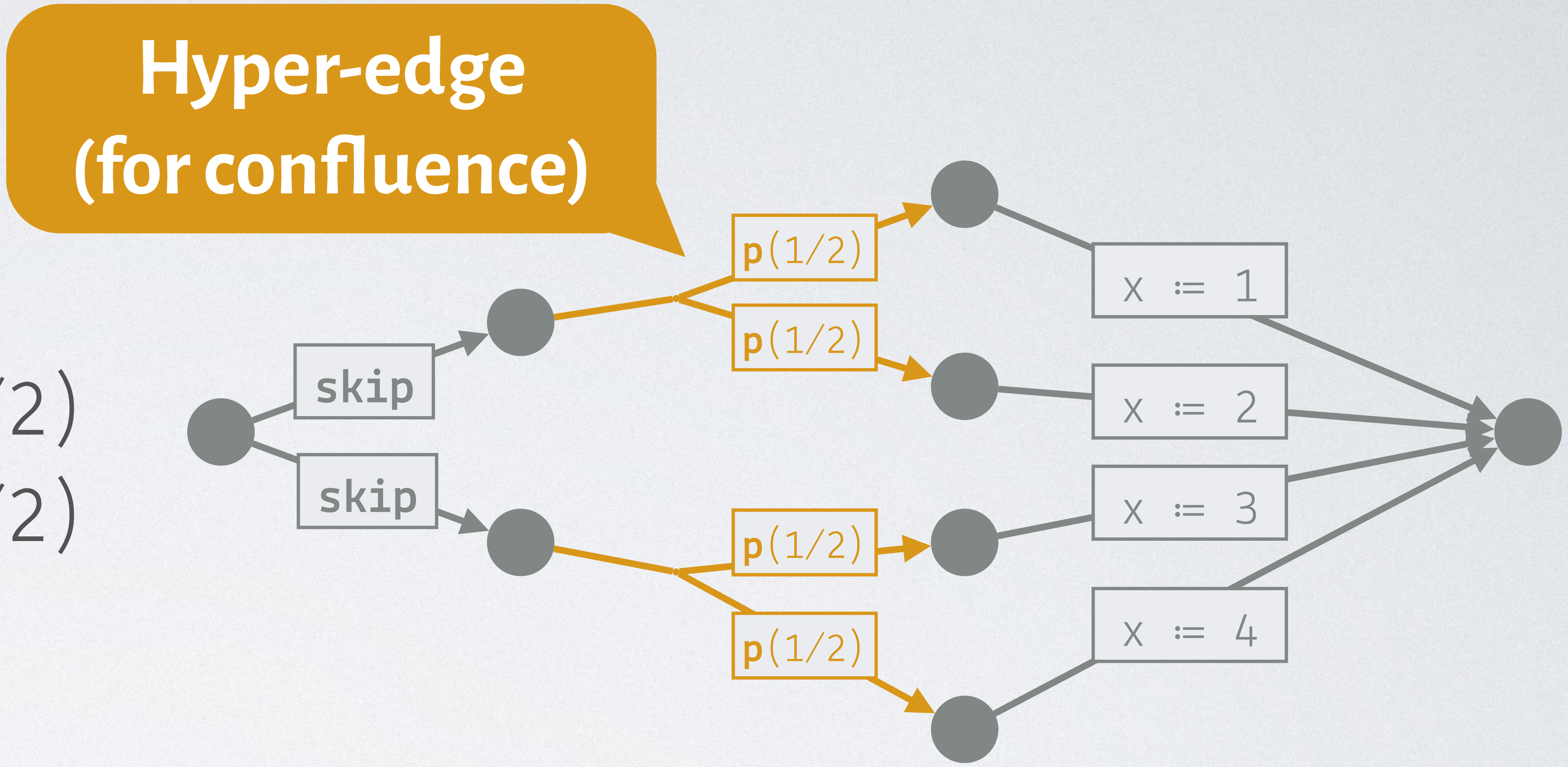
Our Approach: Control-Flow Hyper-Graphs

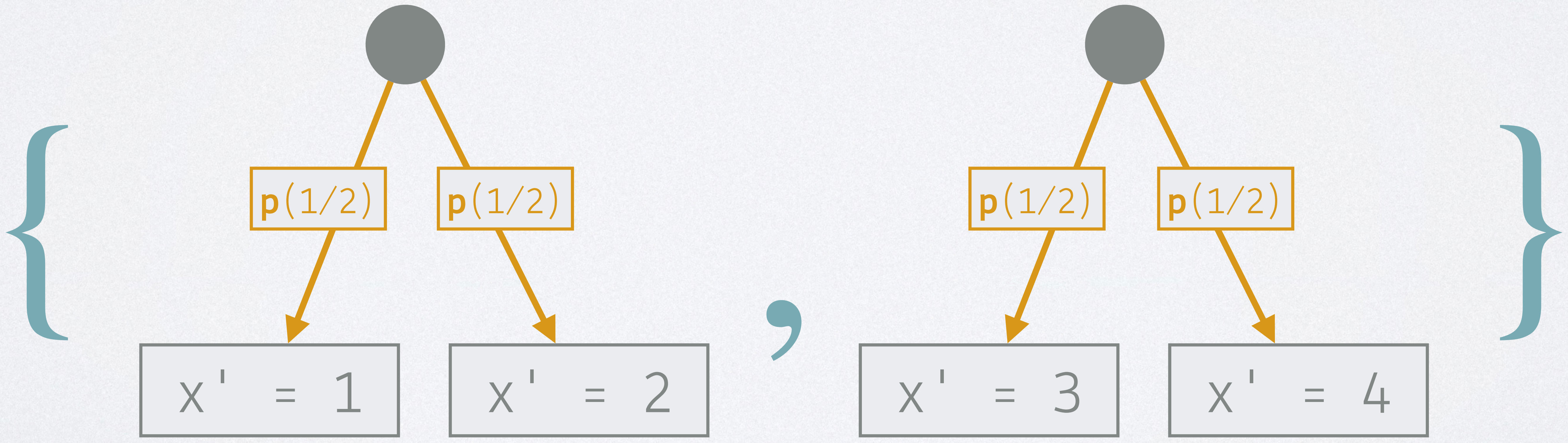
```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```



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
```

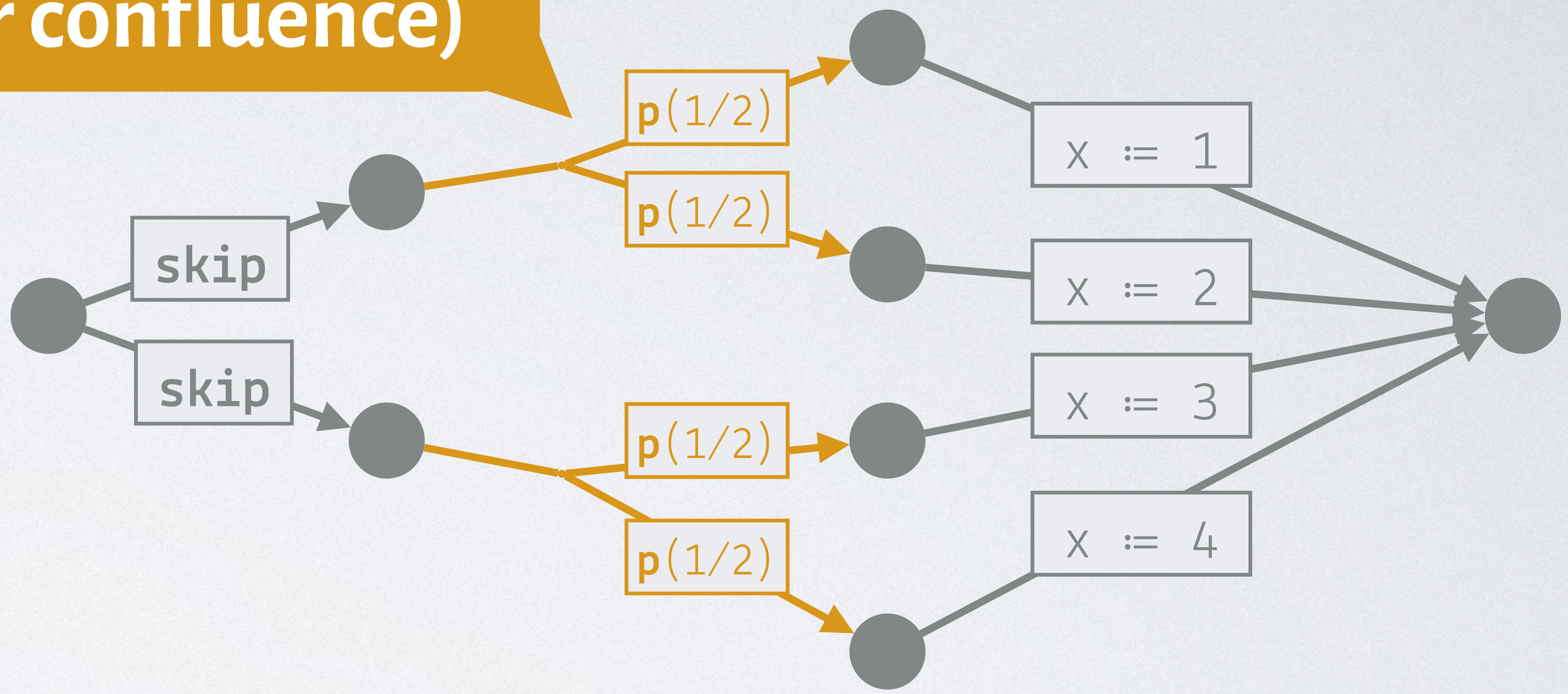




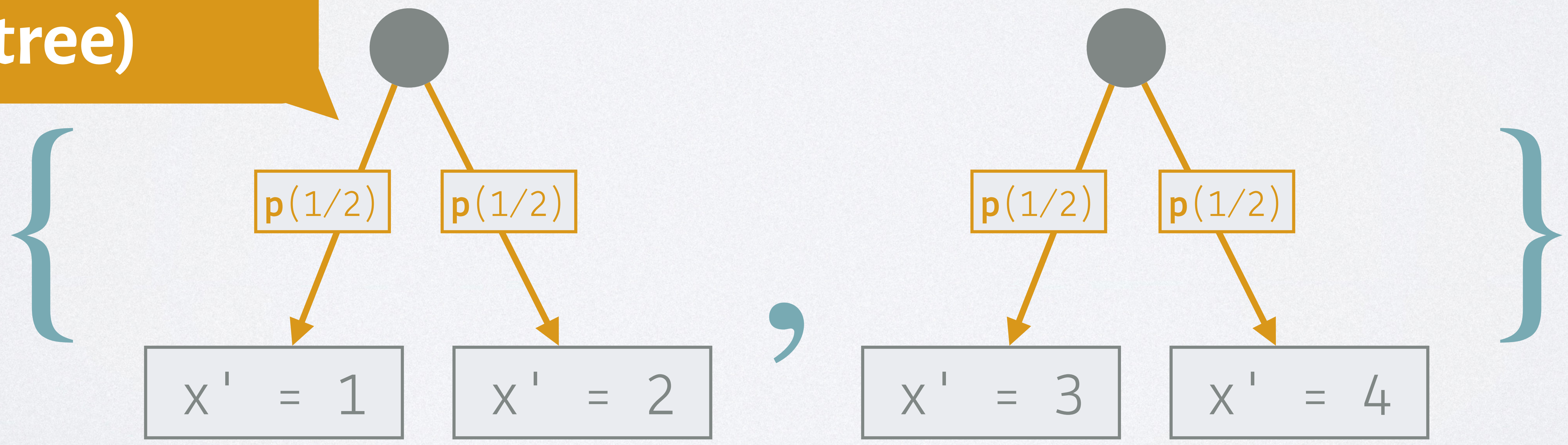
Our Approach: Control-Flow Hyper-Graphs

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

Hyper-edge (for confluence)

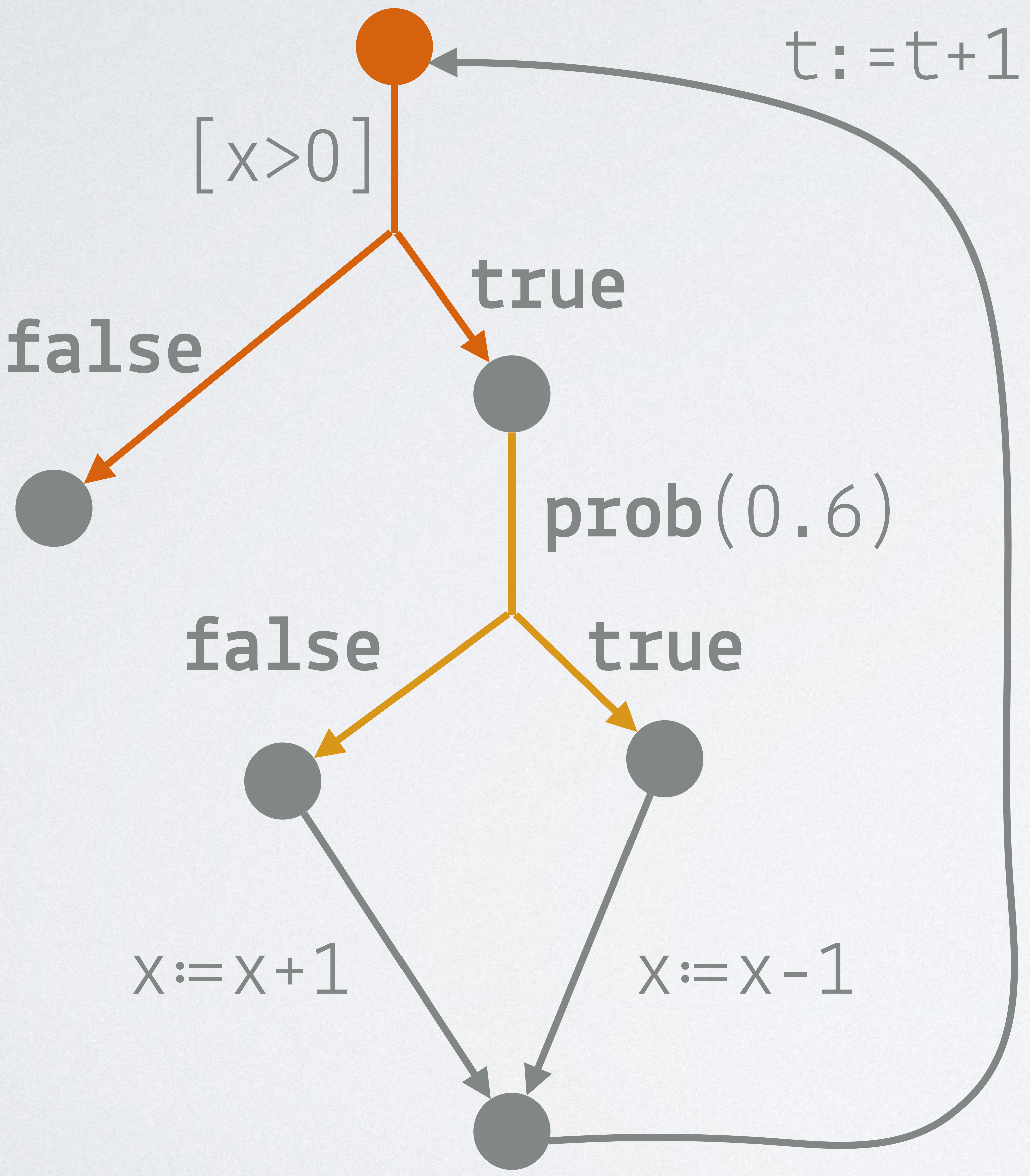


Hyper-path (like a tree)

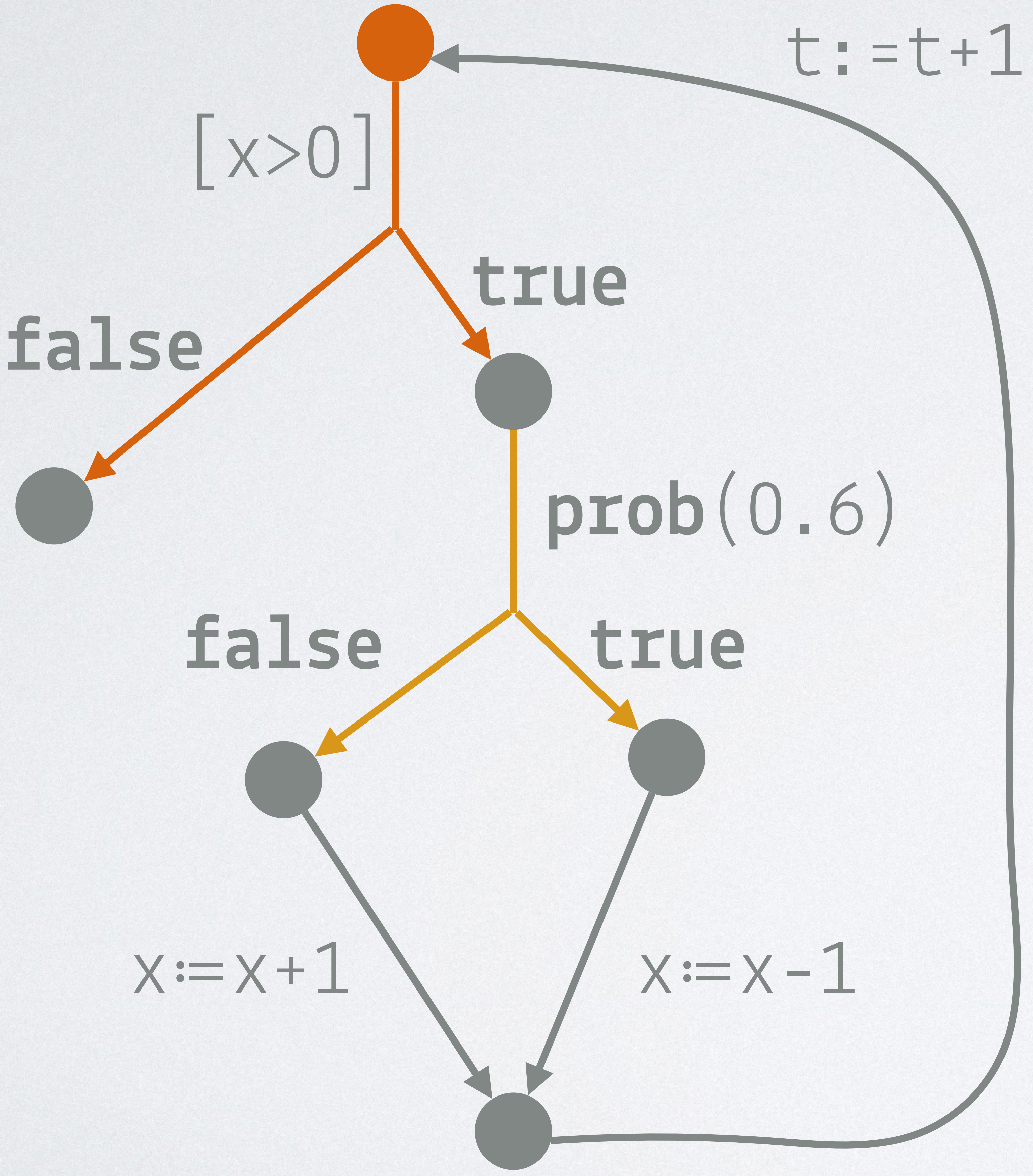


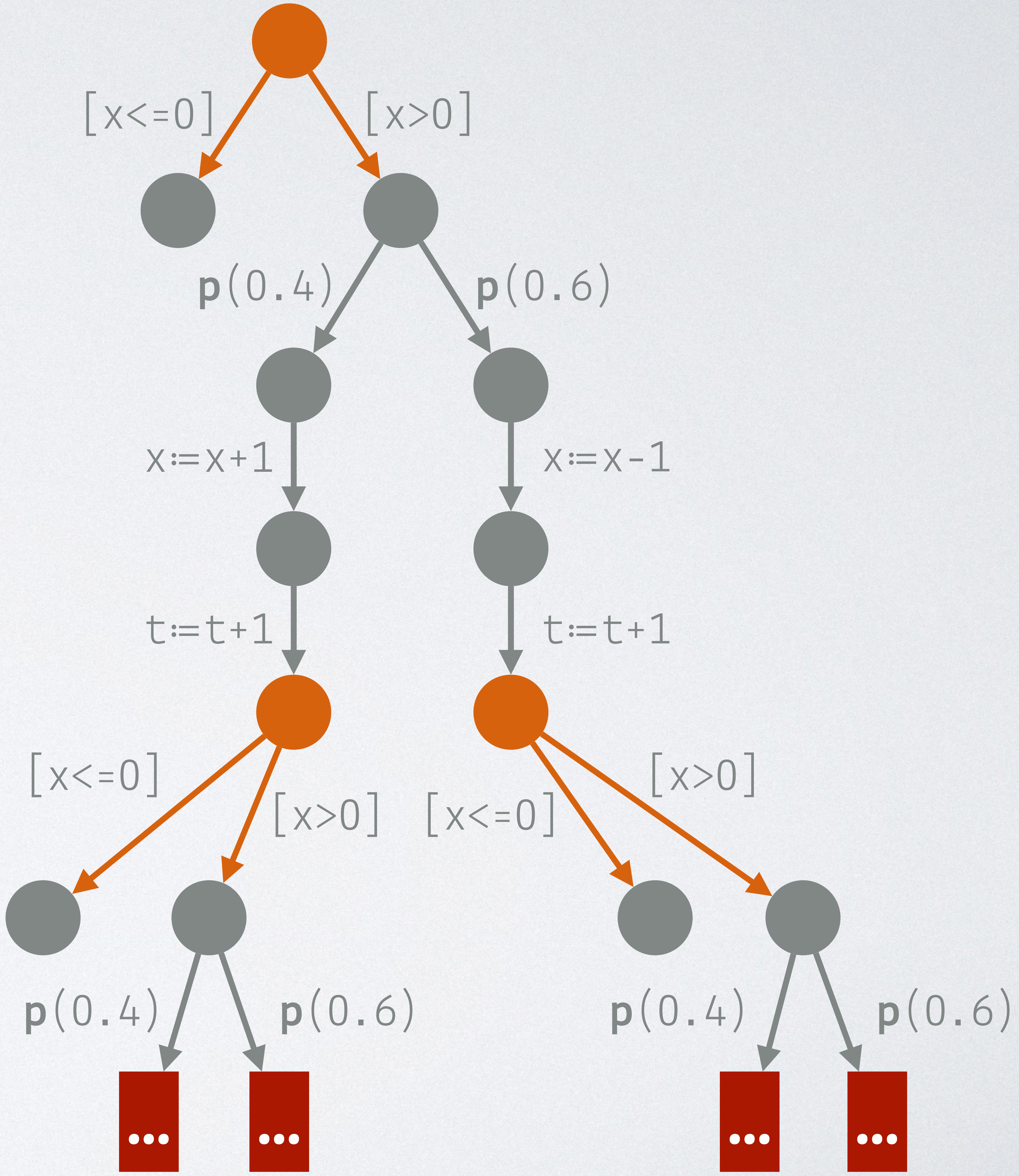
Hyper-Paths are Infinite Trees!

Hyper-Paths are Infinite Trees!

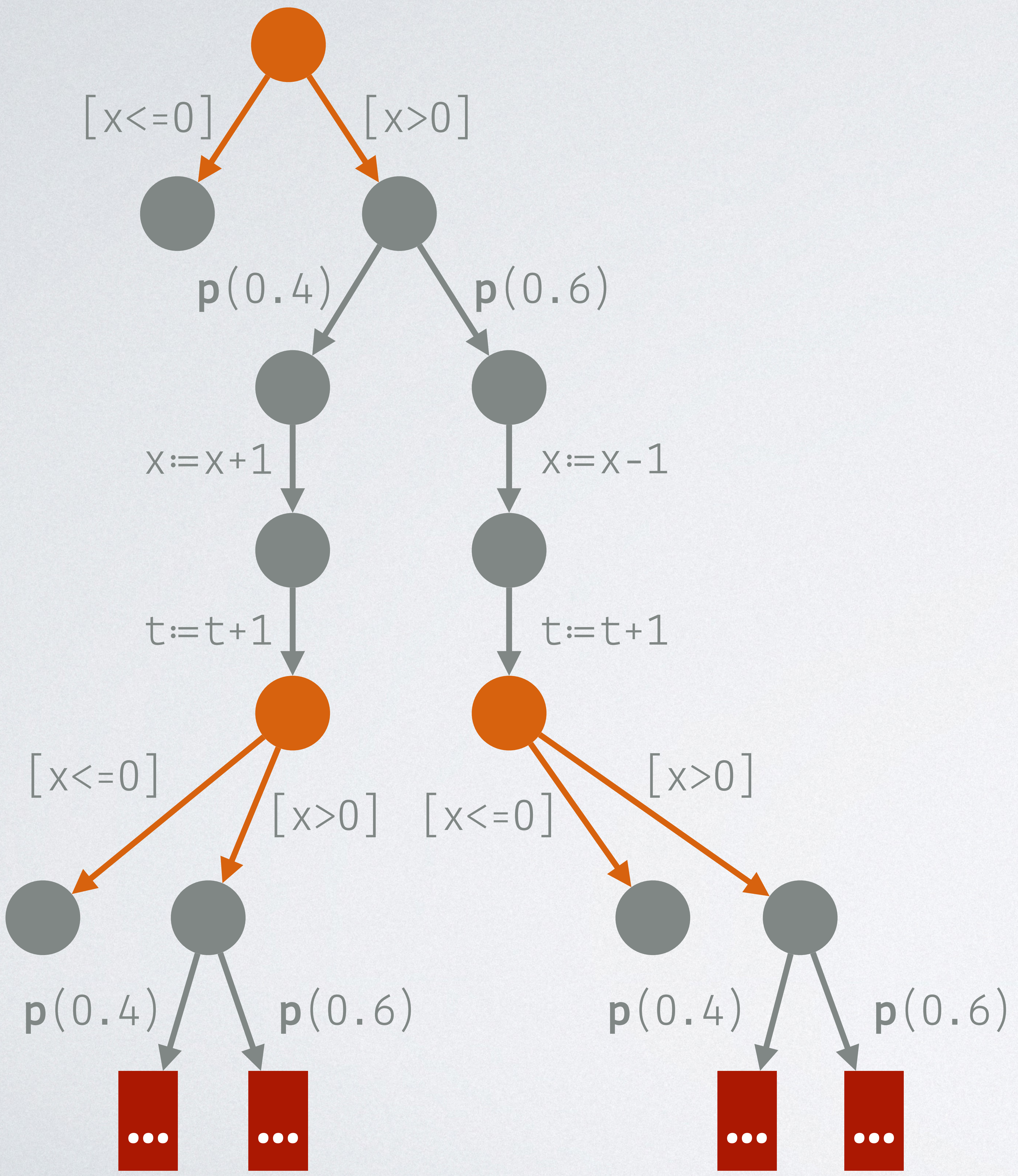


Hyper-Paths are Infinite Trees!

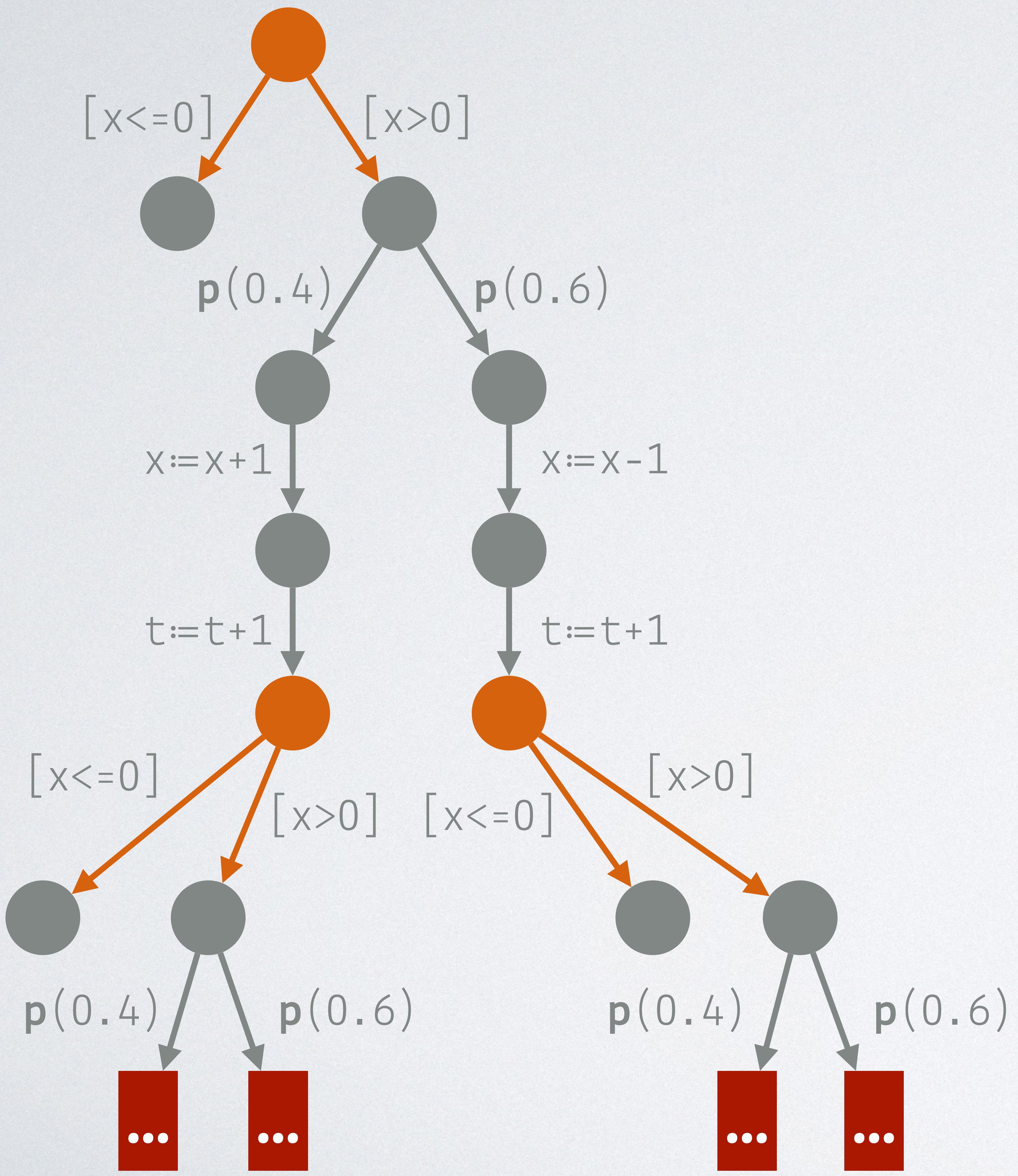


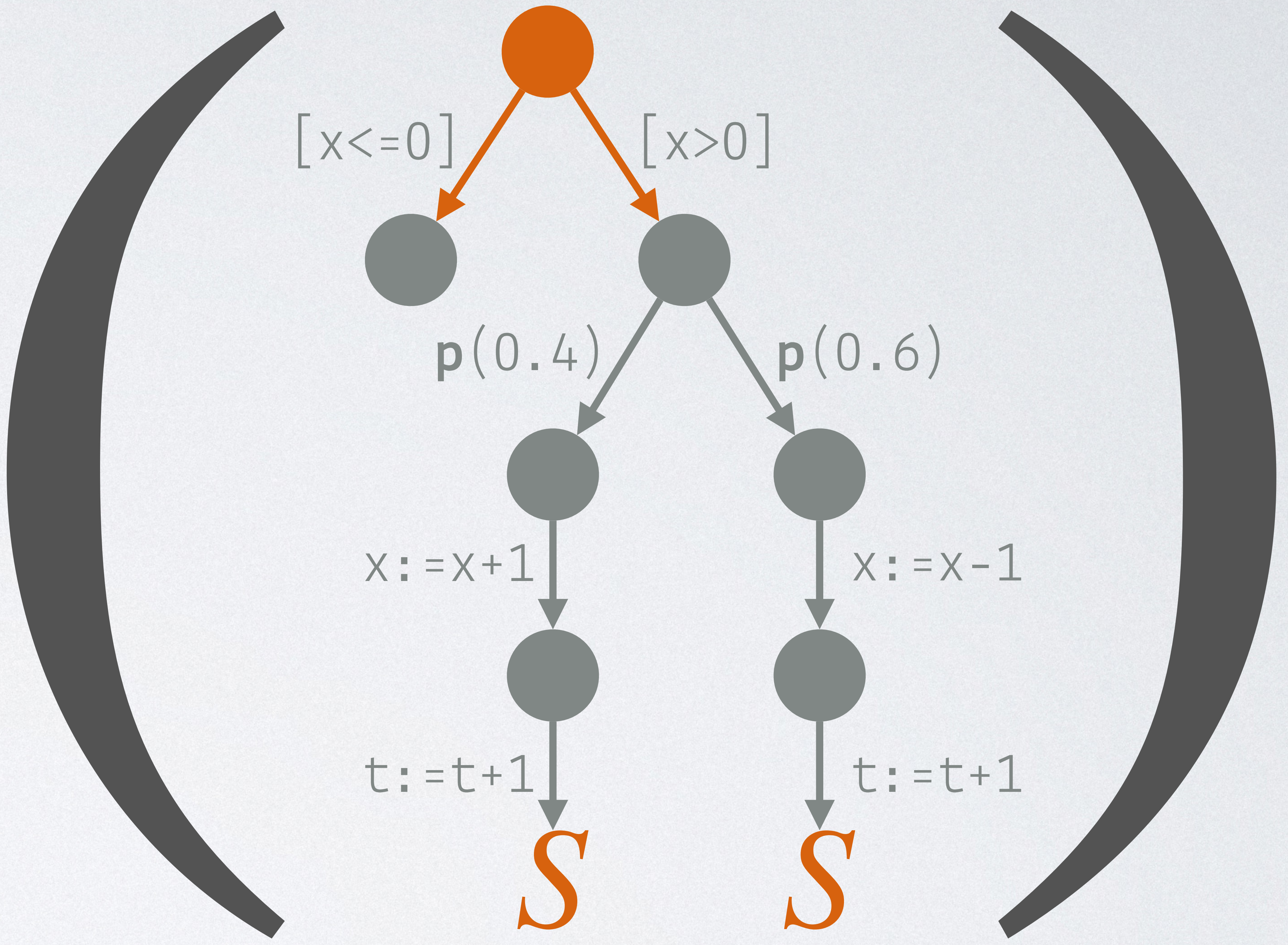


Recursive Program Schemes

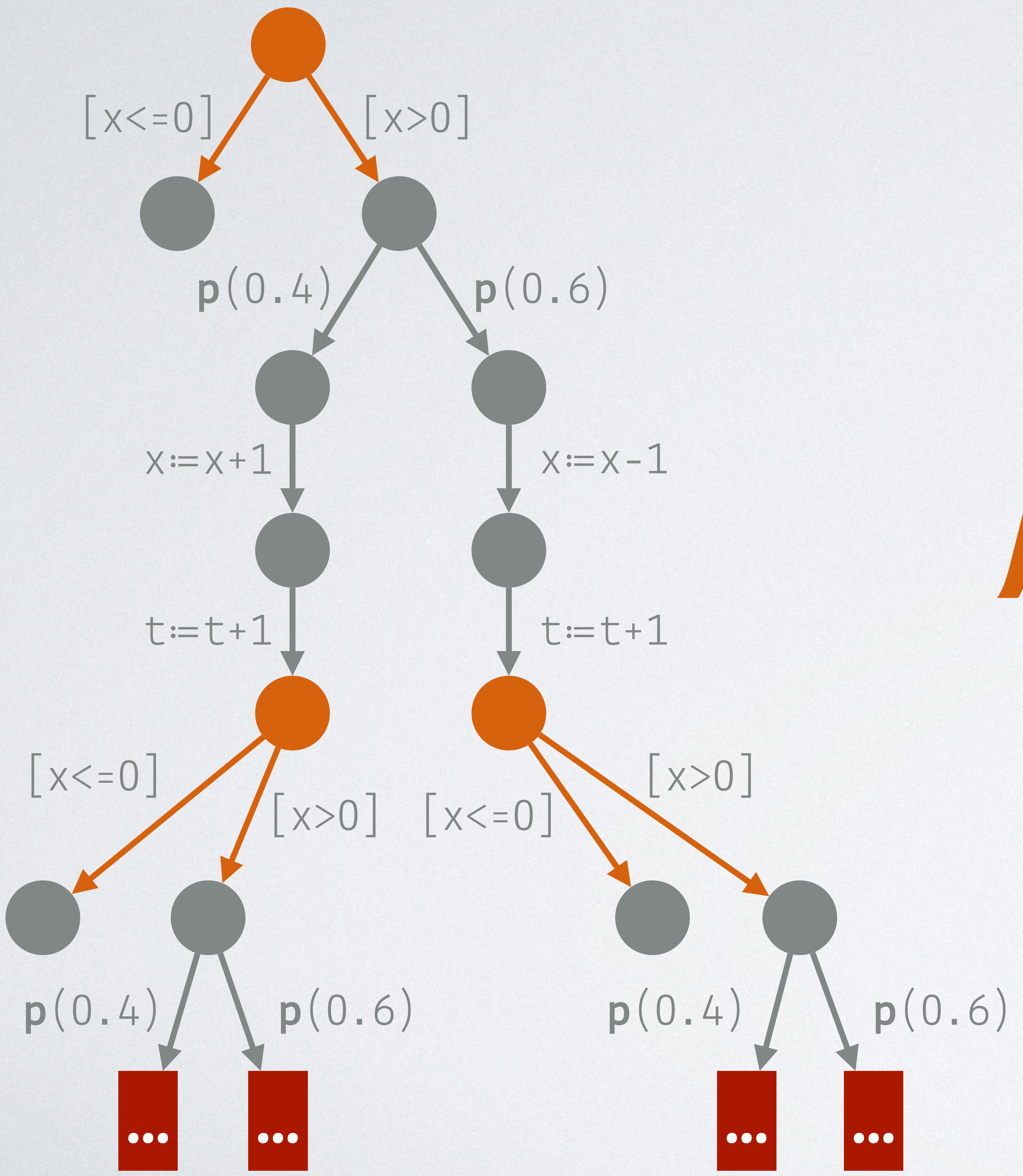


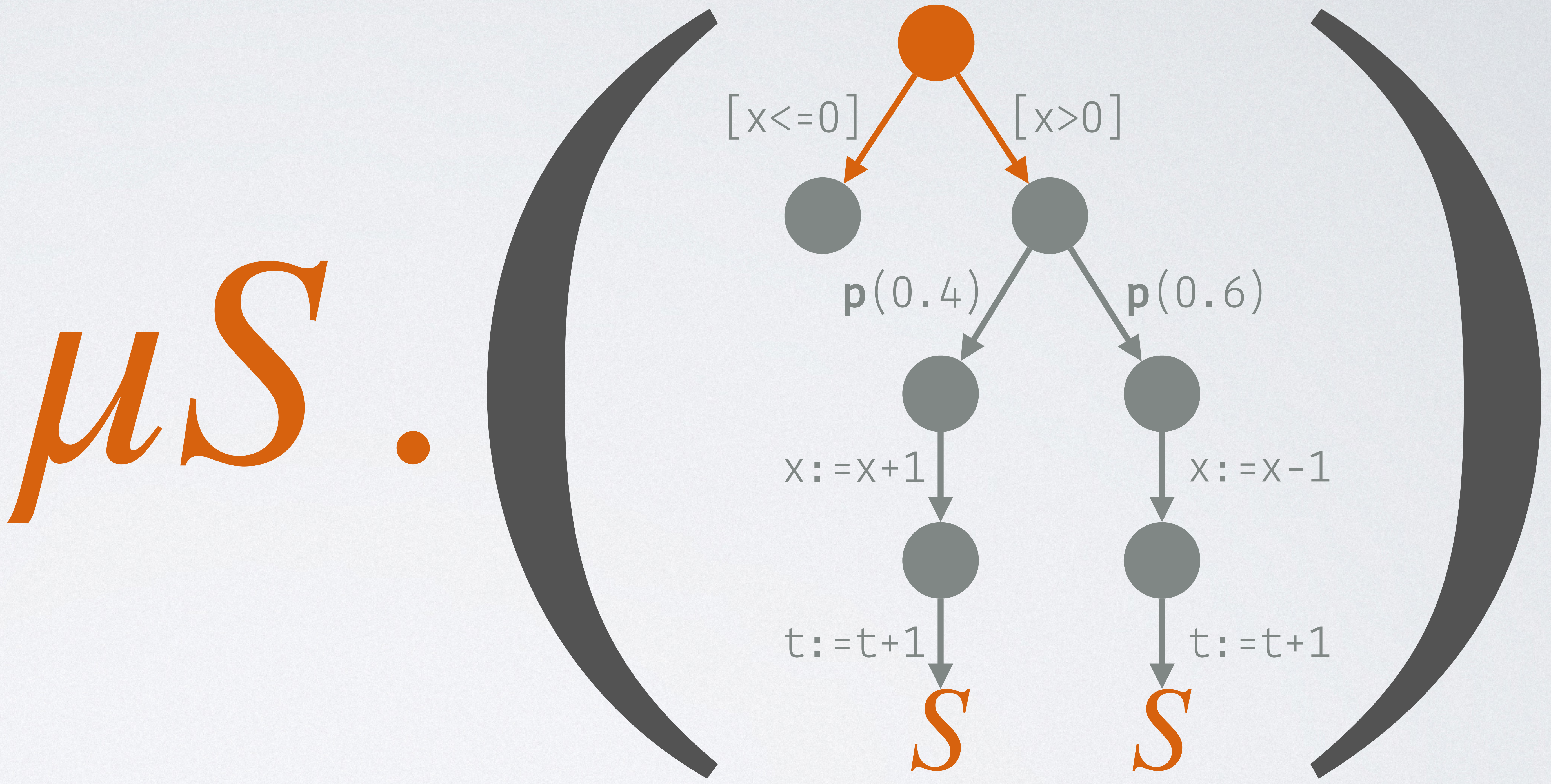
Recursive Program Schemes



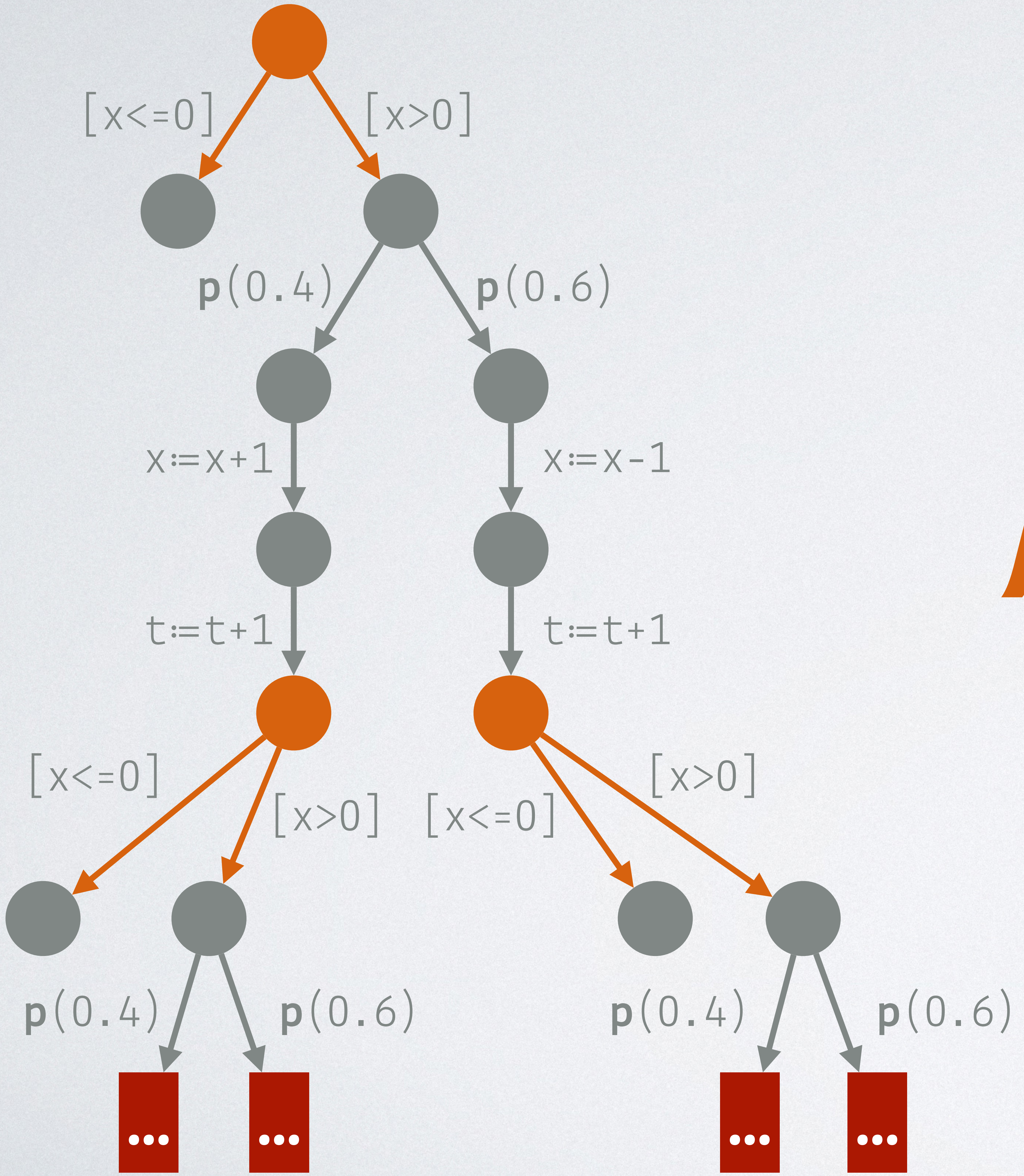


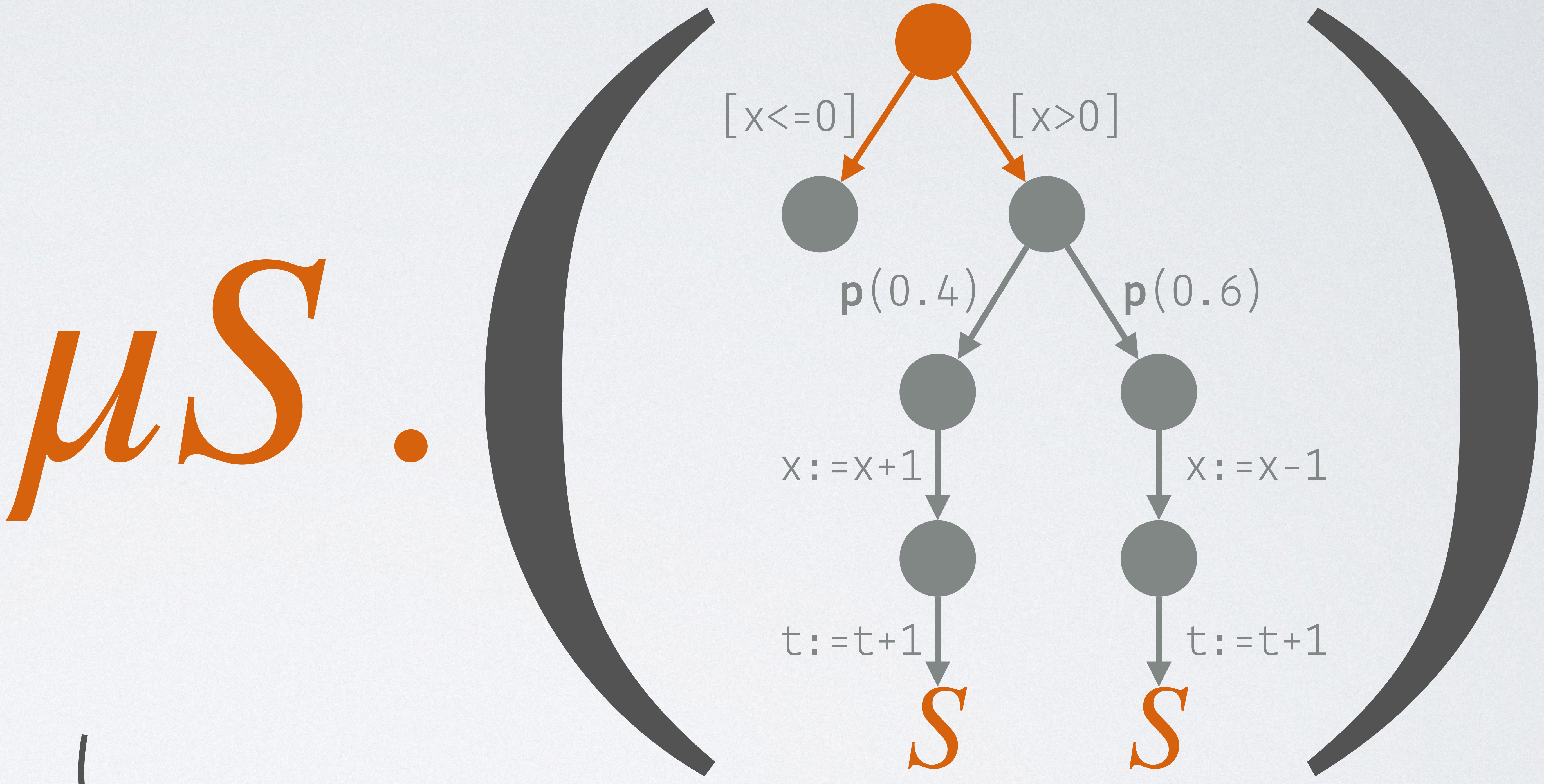
Recursive Program Schemes





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**

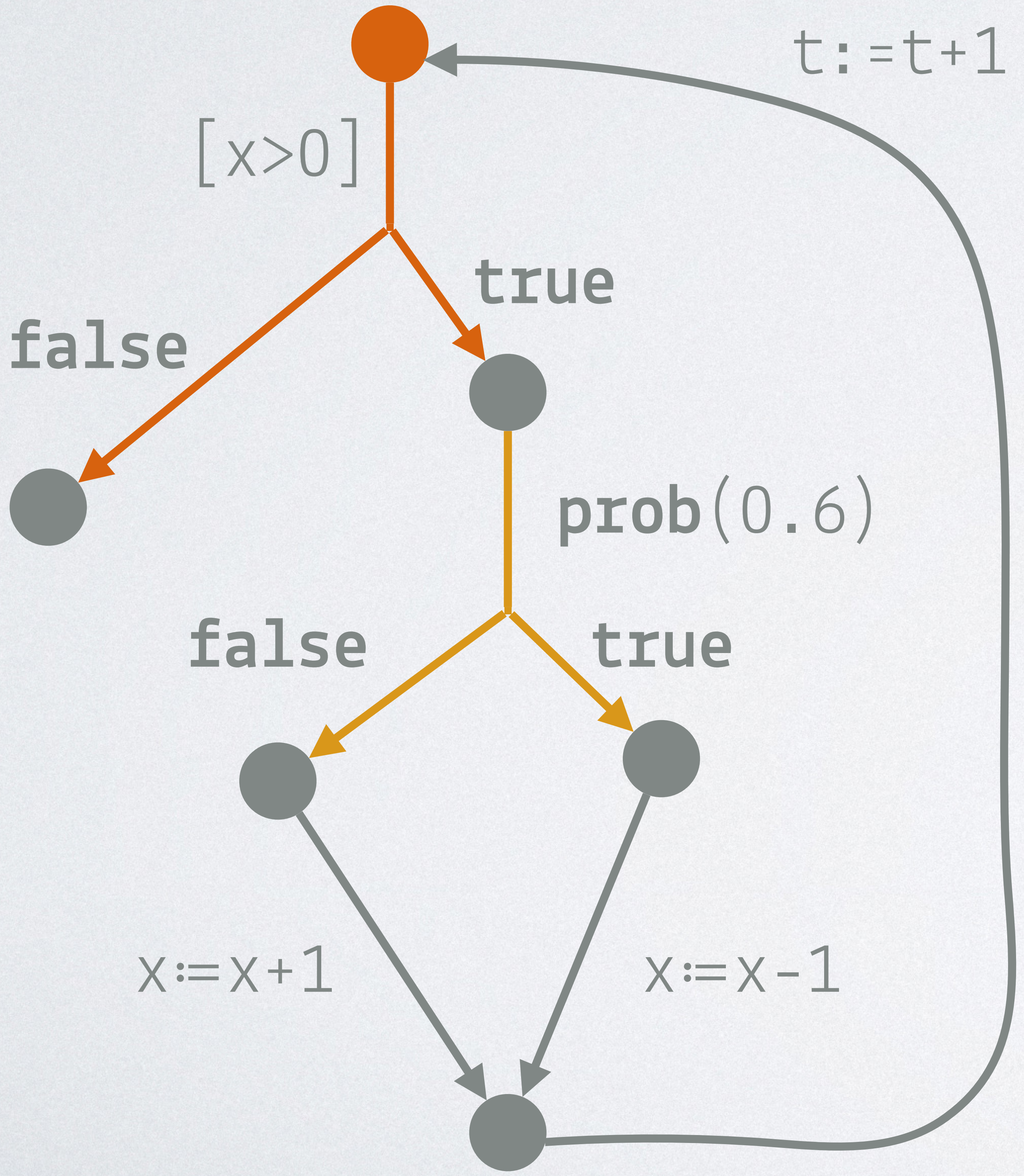
A Control-Flow-Hyper-Graph's Perspective

- **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

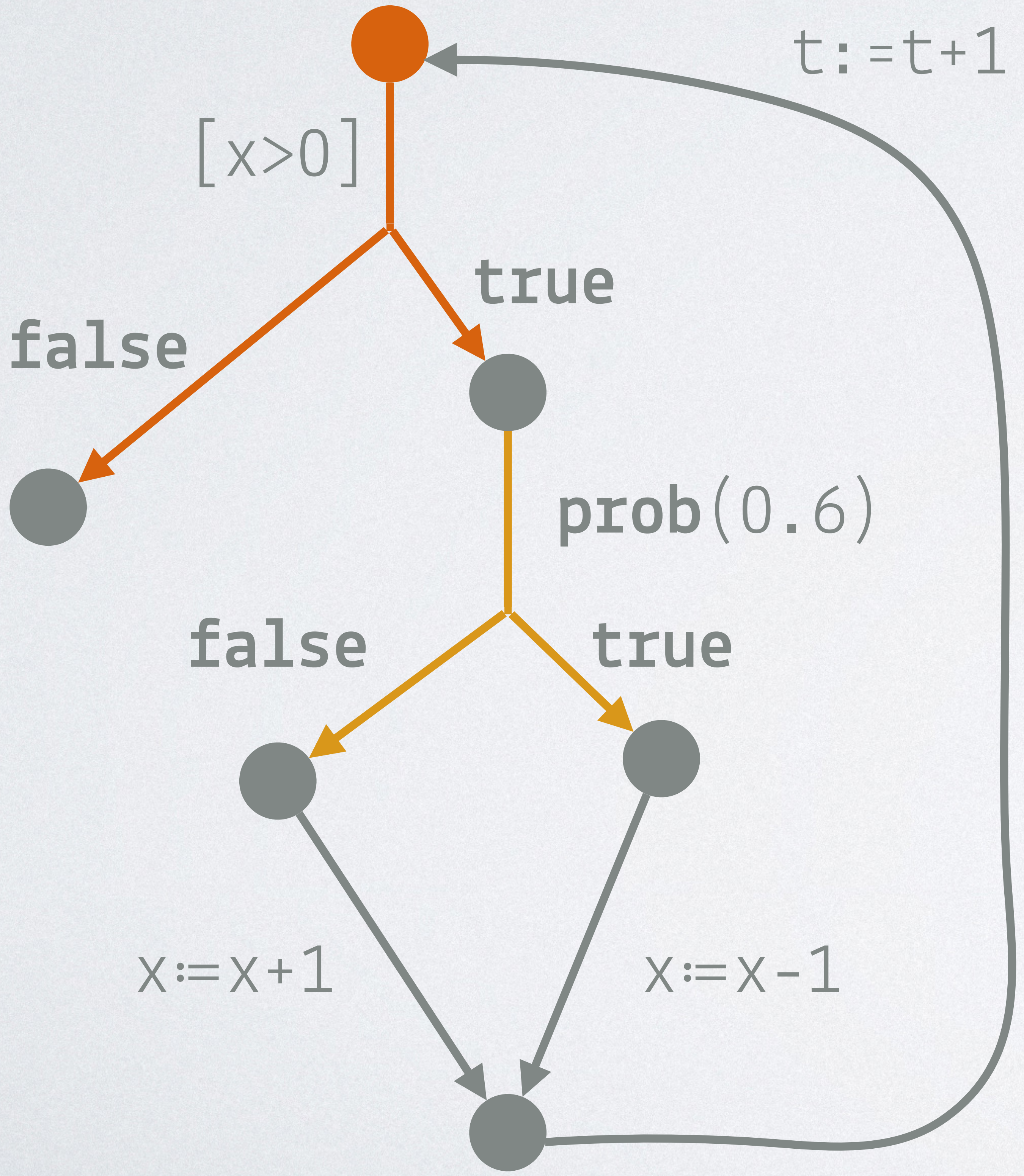
- **Markov Algebras** are compatible with **control-flow hyper-graphs** via **recursive program schemes**




```
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
```

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

Challenge III:
How to carry out quantitative analyses efficiently?

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

Iterative Program 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 Program Analysis

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

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

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

Iterative Program Analysis

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

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

$$\Downarrow$$

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

- 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 Program Analysis

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

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

$\Downarrow$

$\kappa^{(0)} = 0$

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

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

$\cdots$

$\kappa^{(\infty)} = 2$

Need ∞ iterations to converge!

- 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 Program Analysis

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

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

- 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 Program Analysis

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

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

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

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, expectations, or matrices
  - **Loop-Invariant Generation:** Derive probabilistic or expectation invariants

Beyond Loops

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

Beyond Loops

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

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

Beyond Loops

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

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

Beyond Loops

```
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!

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

Beyond Loops

```
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

Beyond Loops

```
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

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

Beyond Loops

```
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

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

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

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

Beyond Loops

```
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!

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

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

Newton's Method

Newton's Method

- Solve the equation $f(x) = 0$ where $f'(x)$ is well-defined

Newton's Method

- Solve the equation $f(x) = 0$ where $f'(x)$ is well-defined

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

$$f(x) = 1/3 * 1 + 2/3 * (x * x) - x$$
$$= 2/3 * x^2 - x + 1/3$$
$$f'(x) = 4/3 * x - 1$$

Newton's Method

- Solve the equation $f(x) = 0$ where $f'(x)$ is well-defined

- Start from an initial approximation $\nu^{(0)}$

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

$$\begin{aligned} f(x) &= 1/3 * 1 + 2/3 * (x * x) - x \\ &= 2/3 * x^2 - x + 1/3 \\ f'(x) &= 4/3 * x - 1 \end{aligned}$$

$$\nu^{(0)} \leftarrow 0$$

Newton's Method

- Solve the equation $f(x) = 0$ where $f'(x)$ is well-defined
- Start from an initial approximation $\nu^{(0)}$
- At step i, solve a linear equation $f(\nu^{(i)}) + f'(\nu^{(i)}) * (y - \nu^{(i)}) = 0$, i.e., set $\nu^{(i+1)} = \nu^{(i)} - f(\nu^{(i)})/f'(\nu^{(i)})$

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

$$\begin{aligned} f(x) &= 1/3 * 1 + 2/3 * (x * x) - x \\ &= 2/3 * x^2 - x + 1/3 \\ f'(x) &= 4/3 * x - 1 \end{aligned}$$

$$\nu^{(0)} \leftarrow 0$$

Newton's Method

- Solve the equation $f(x) = 0$ where $f'(x)$ is well-defined
- Start from an initial approximation $\nu^{(0)}$
- At step i, solve a linear equation $f(\nu^{(i)}) + f'(\nu^{(i)}) * (y - \nu^{(i)}) = 0$, i.e., set $\nu^{(i+1)} = \nu^{(i)} - f(\nu^{(i)})/f'(\nu^{(i)})$

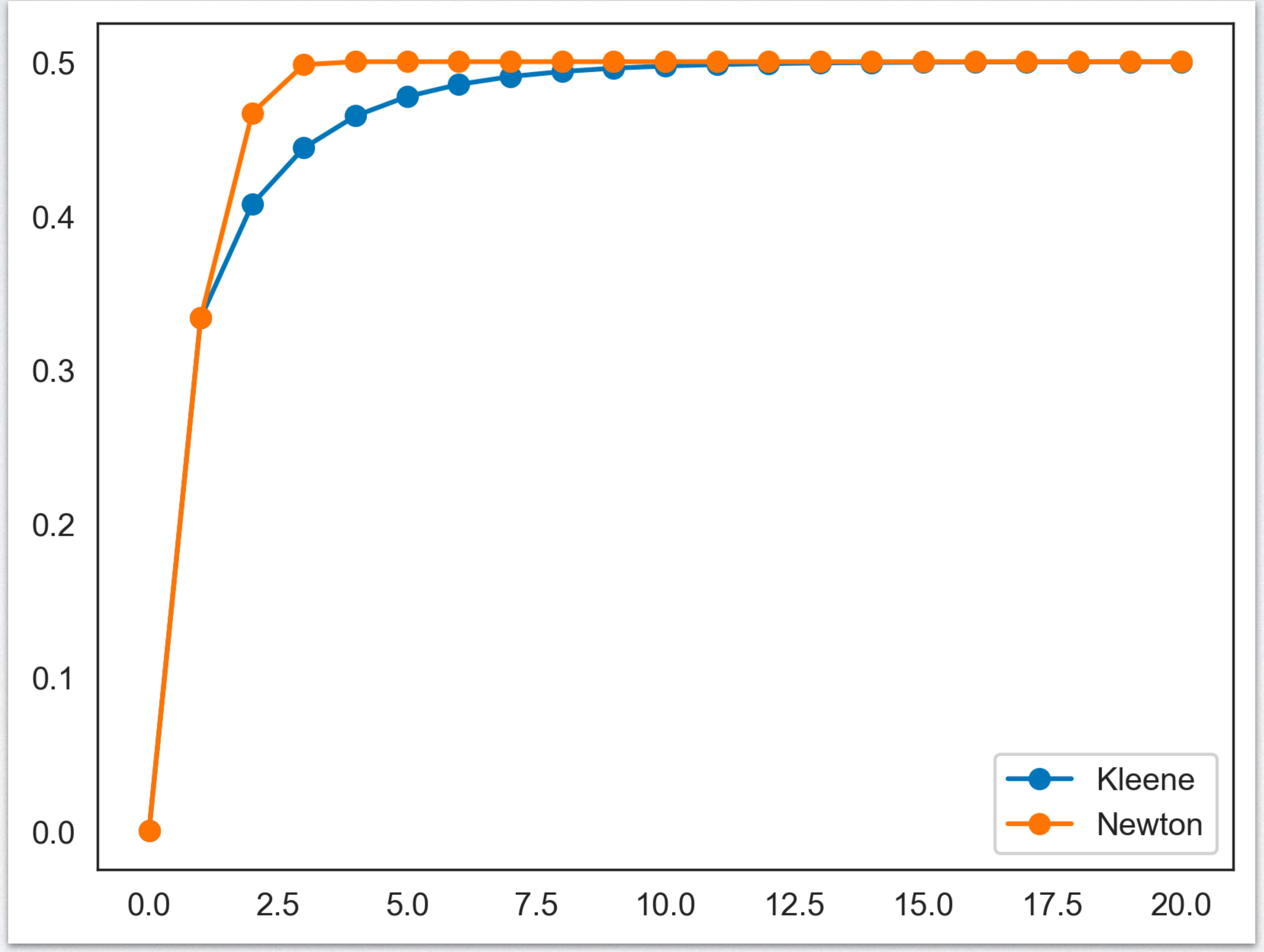
$$p = 1/3 * 1 + 2/3 * (p * p)$$

$$\begin{aligned} f(x) &= 1/3 * 1 + 2/3 * (x * x) - x \\ &= 2/3 * x^2 - x + 1/3 \\ f'(x) &= 4/3 * x - 1 \end{aligned}$$

$$\nu^{(0)} \leftarrow 0$$

$$\begin{aligned} \nu^{(1)} &= 0 - f(0)/f'(0) = 1/3 \\ \nu^{(2)} &= 1/3 - f(1/3)/f'(1/3) = 7/15 \\ \nu^{(3)} &= 7/15 - f(7/15)/f'(7/15) = 127/255 \\ &\cdots \\ \nu^{(\infty)} &= 1/2 \end{aligned}$$

Newton's Method Converges Faster



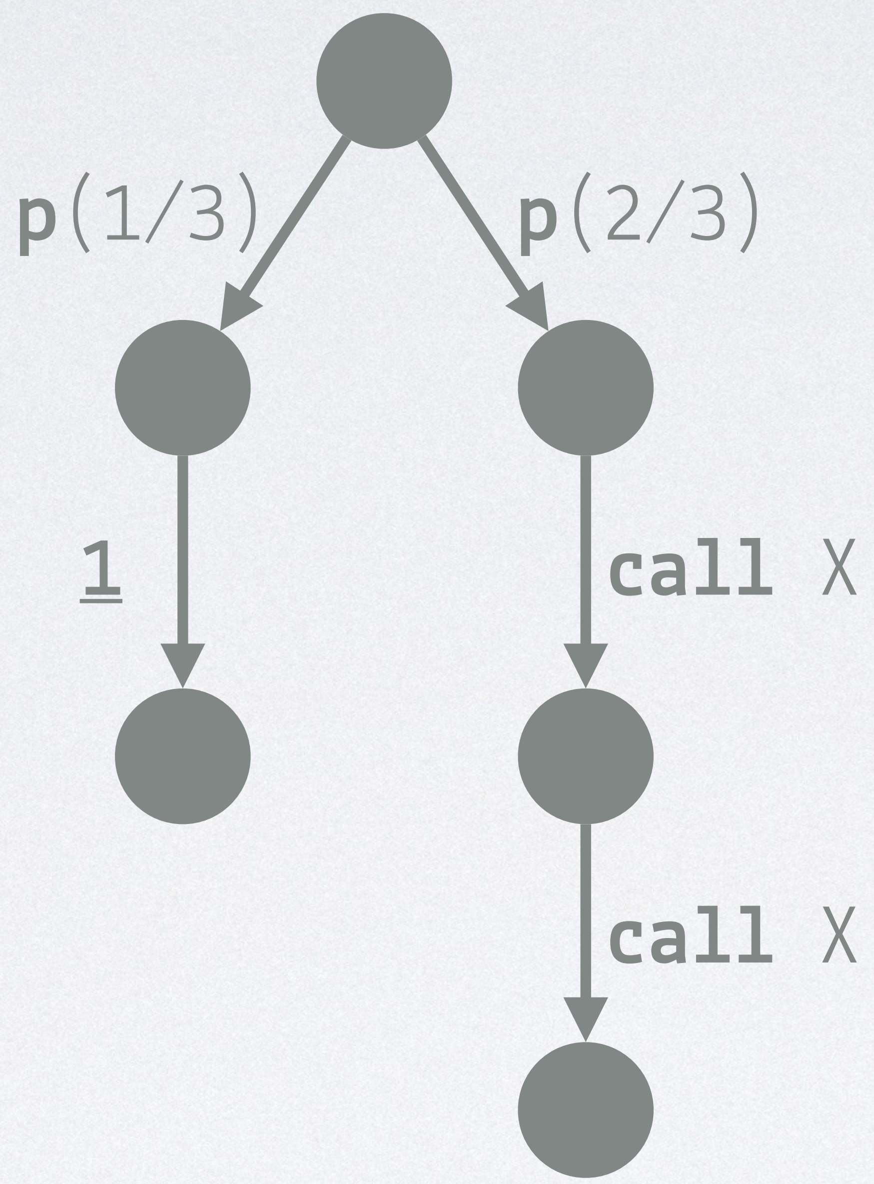
Newtonian Program Analysis (NPA)

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

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

Newtonian Program Analysis (NPA)

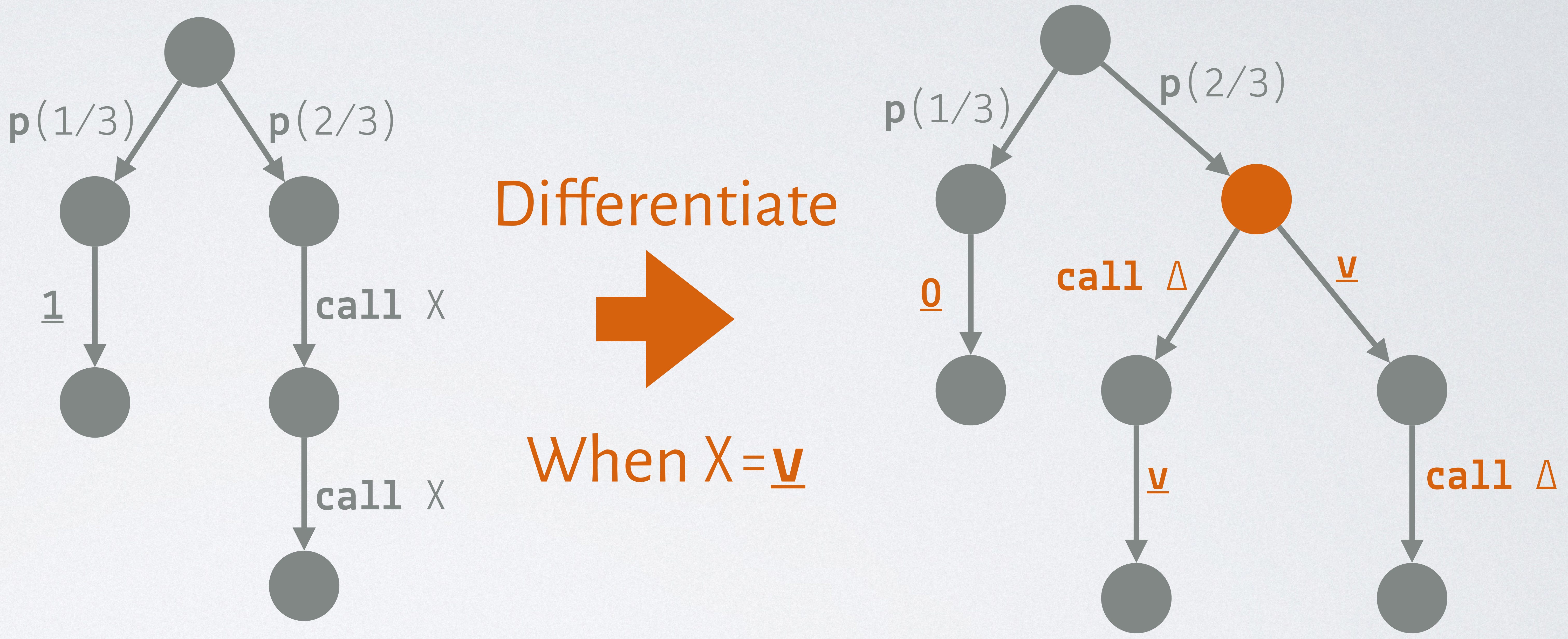
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



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

Newtonian Program Analysis (NPA)

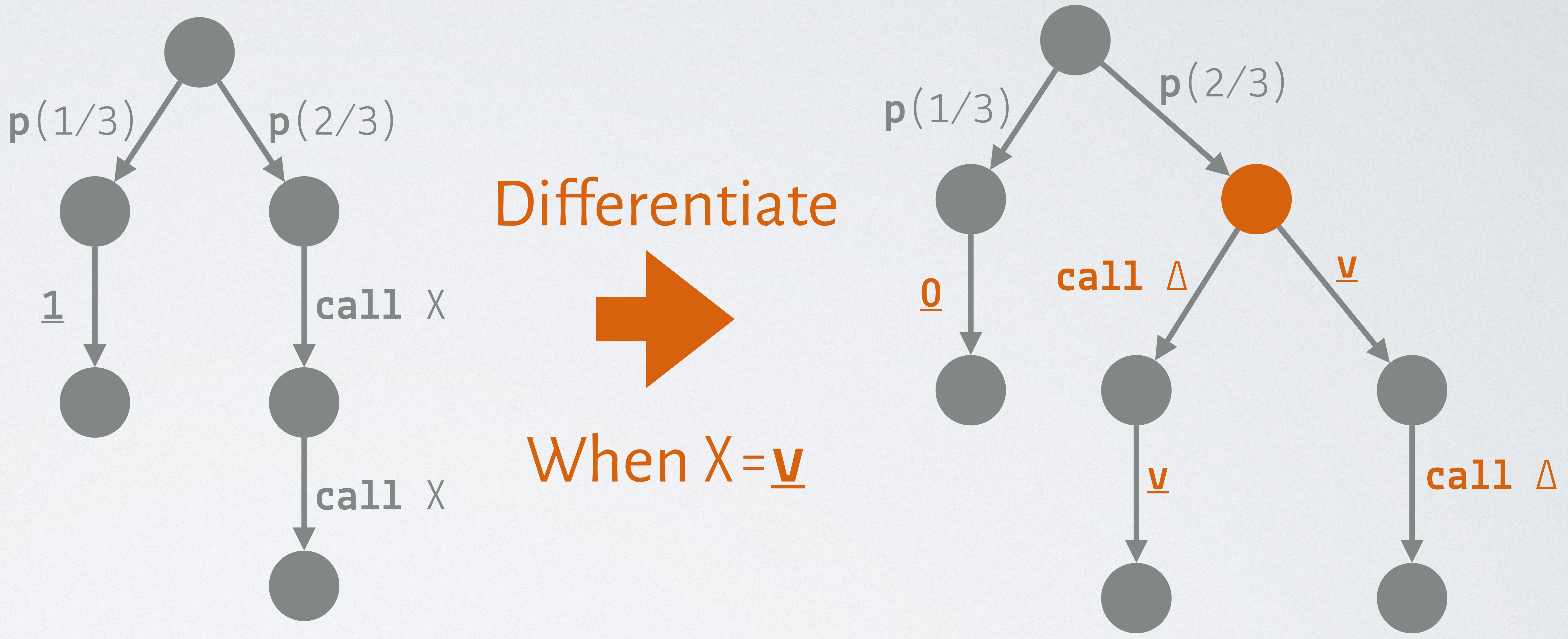
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



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

Newtonian Program Analysis (NPA)

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

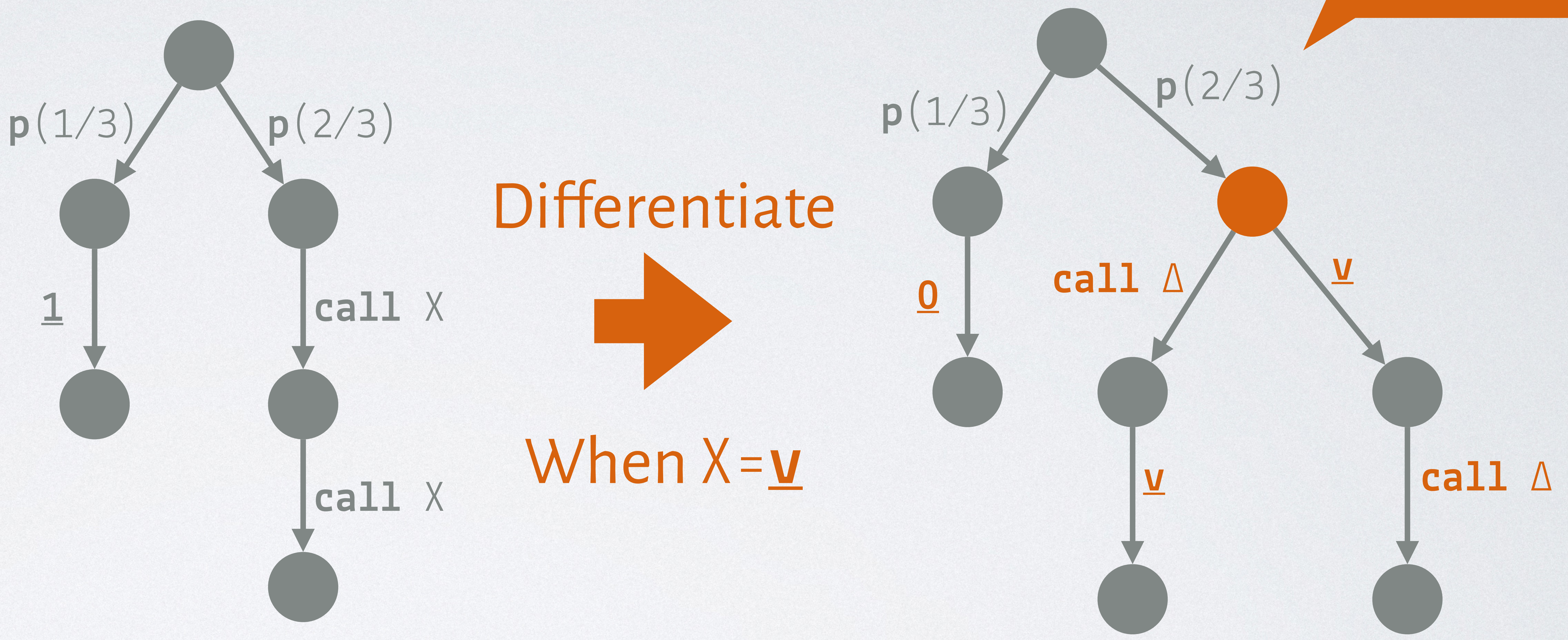


Every root-to-leaf path contains **at most one call**!

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

Newtonian Program Analysis (NPA)

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



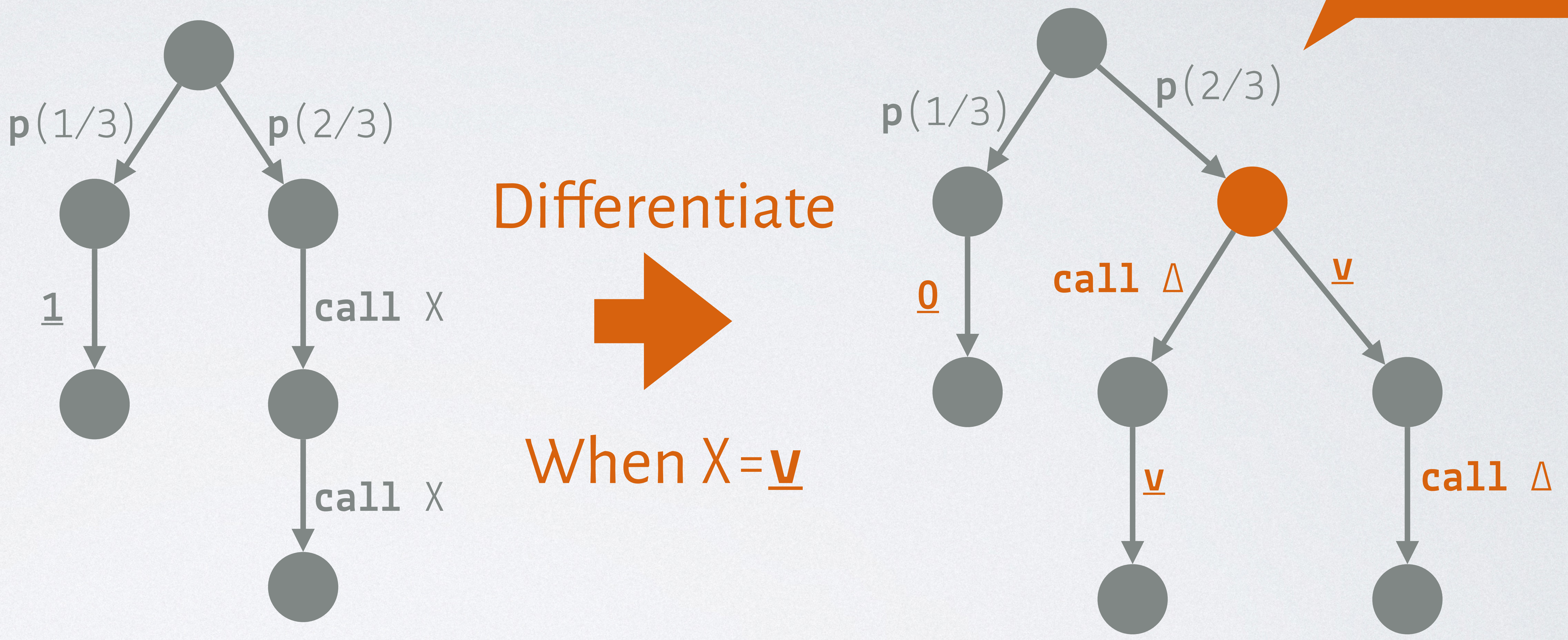
Linear!

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

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

Newtonian Program Analysis (NPA)

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



Linear!

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

$$\boxed{\frac{d(f*g)}{dx} = \frac{df}{dx}*g + f*\frac{dg}{dx}}$$

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

Newtonian Program Analysis (NPA)

Newtonian Program Analysis (NPA)

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

Newtonian Program Analysis (NPA)

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

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

$$\mathscr{D}(X)|_\nu(Y) \triangleq Y$$

$$\mathscr{D}(f \oplus g)|_\nu(Y) \triangleq \mathscr{D}f|_\nu(Y) \oplus \mathscr{D}g_\nu(Y)$$

$$\mathscr{D}(f \otimes g)|_\nu(Y) \triangleq (\mathscr{D}f|_\nu(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g|_\nu(Y))$$

Newtonian Program Analysis (NPA)

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

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

Procedure call to X

$$\mathscr{D}(X)|_\nu(Y) \triangleq Y$$

$$\mathscr{D}(f \oplus g)|_\nu(Y) \triangleq \mathscr{D}f|_\nu(Y) \oplus \mathscr{D}g_\nu(Y)$$

$$\mathscr{D}(f \otimes g)|_\nu(Y) \triangleq (\mathscr{D}f|_\nu(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g|_\nu(Y))$$

Newtonian Program Analysis (NPA)

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

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

Branching

$$\mathscr{D}(X)\,|_{\nu}(Y) \triangleq Y$$

$$\mathscr{D}(f \oplus g)\,|_{\nu}(Y) \triangleq \mathscr{D}f\,|_{\nu}(Y) \oplus \mathscr{D}g_{\nu}(Y)$$

$$\mathscr{D}(f \otimes g)\,|_{\nu}(Y) \triangleq (\mathscr{D}f\,|_{\nu}(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g\,|_{\nu}(Y))$$

Newtonian Program Analysis (NPA)

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

$$\mathscr{D}(X)|_{\nu}(Y) \triangleq Y$$

Sequencing

$$\oplus\, g)|_{\nu}(Y) \triangleq \mathscr{D}f|_{\nu}(Y) \oplus \mathscr{D}g_{\nu}(Y)$$

$$\mathscr{D}(f \otimes g)|_{\nu}(Y) \triangleq (\mathscr{D}f|_{\nu}(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g|_{\nu}(Y))$$

Newtonian Program Analysis (NPA)

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

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

$$\mathscr{D}(X)\,|_\nu(Y) \triangleq Y$$

Sequencing

$$\oplus\, g)\,|_\nu(Y) \triangleq \mathscr{D}f\,|_\nu(Y) \oplus \mathscr{D}g_\nu(Y)$$

$$\mathscr{D}(f \otimes g)\,|_\nu(Y) \triangleq (\mathscr{D}f\,|_\nu(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g\,|_\nu(Y))$$

Interpreting calls to X in g by ν

Newtonian Program Analysis (NPA)

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

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

$$\mathscr{D}(X)|_\nu(Y) \triangleq Y$$

Sequencing

$$\oplus\, g)|_\nu(Y) \triangleq \mathscr{D}f|_\nu(Y) \oplus \mathscr{D}g_\nu($$

Interpreting calls to X in g by ν

$$\mathscr{D}(f \otimes g)|_\nu(Y) \triangleq (\mathscr{D}f|_\nu(Y) \otimes g(\nu)) \oplus (f(\nu) \otimes \mathscr{D}g|_\nu(Y))$$

Interpreting calls to X in g by ν

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**

Support multiple confluences, loops, and recursion

- We develop a differentiation routine for **recursive program schemes**

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

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

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

$\oplus$ defines a partial order and gives an additive structure

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

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**

- We develop a differentiation routine for **recursive program schemes**

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

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

Equation System

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

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**

- We develop a differentiation routine for **recursive program schemes**

$$\text{Equation System} \xrightarrow{\text{Differentiate at } \nu^{(i)}} \text{Linearized Equation System}$$

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

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**

- We develop a differentiation routine for **recursive program schemes**

Equation System

Differentiate at $\nu^{(i)}$

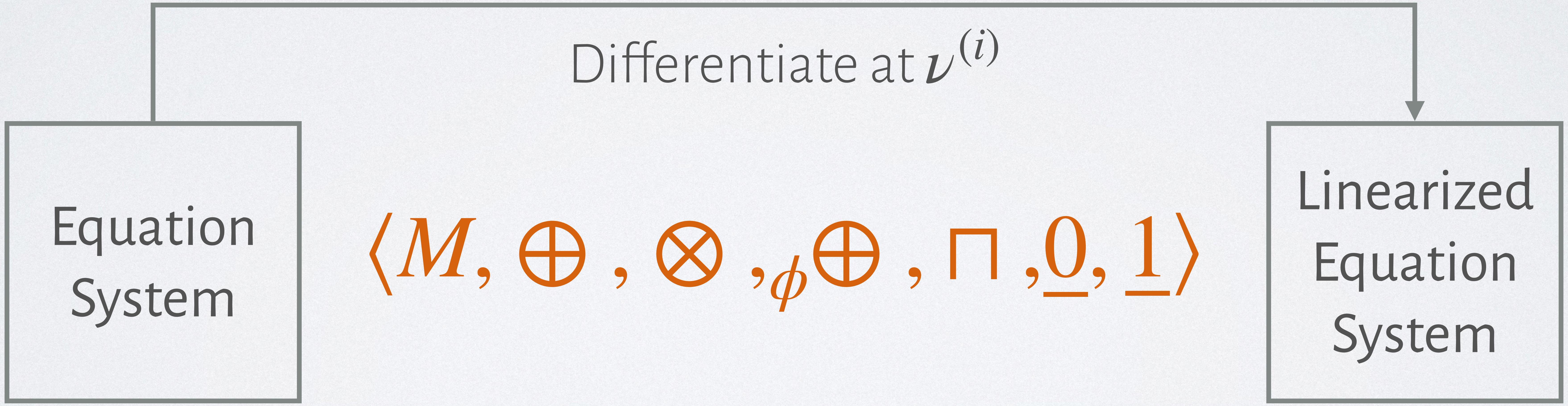
Linearized Equation System

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

Analysis-specified loop-solving strategy

Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**



Our Approach: NPA for pre-Markov Algebras

- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**

Differentiate at $\nu^{(i)}$

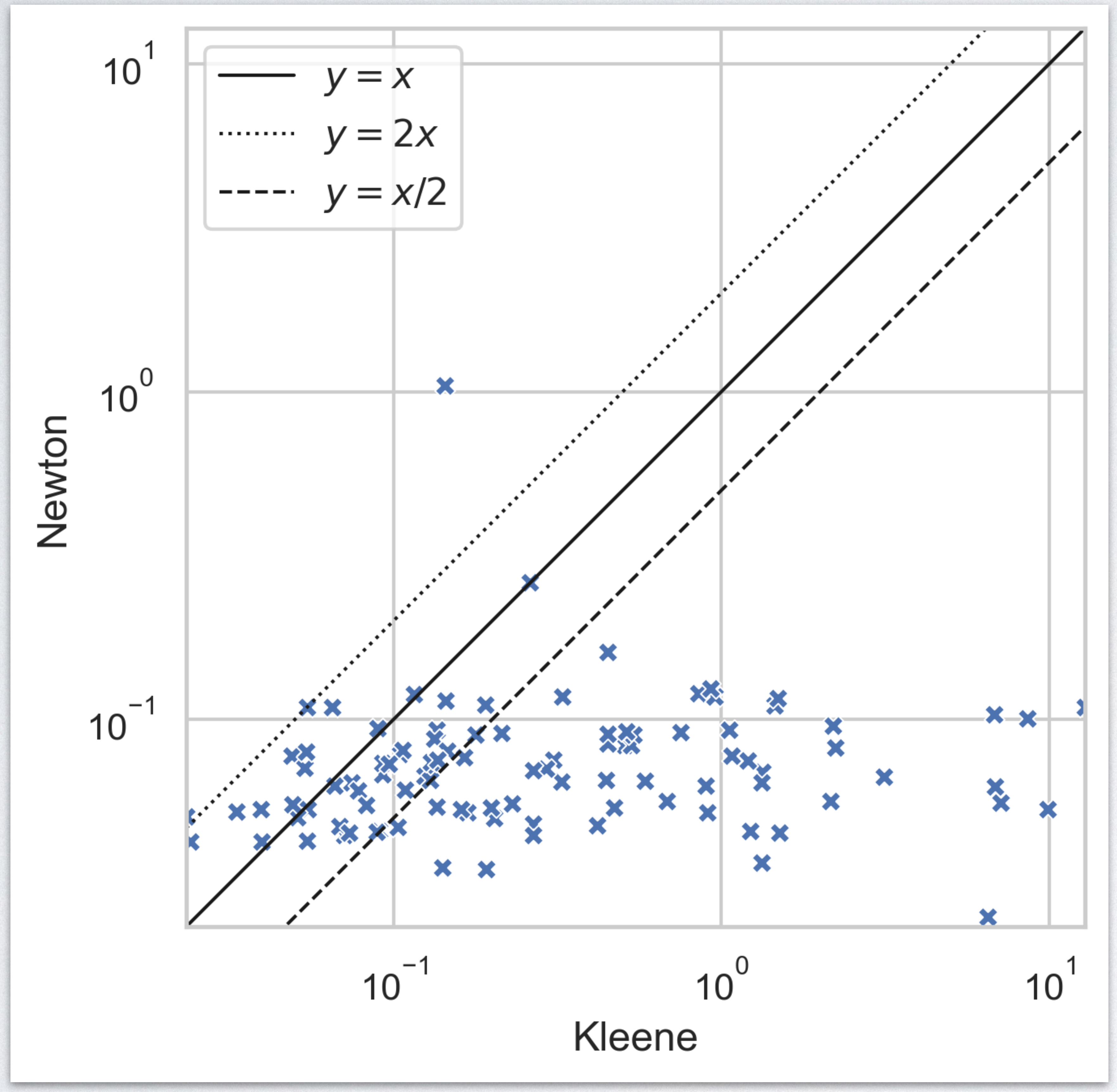
Equation System

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

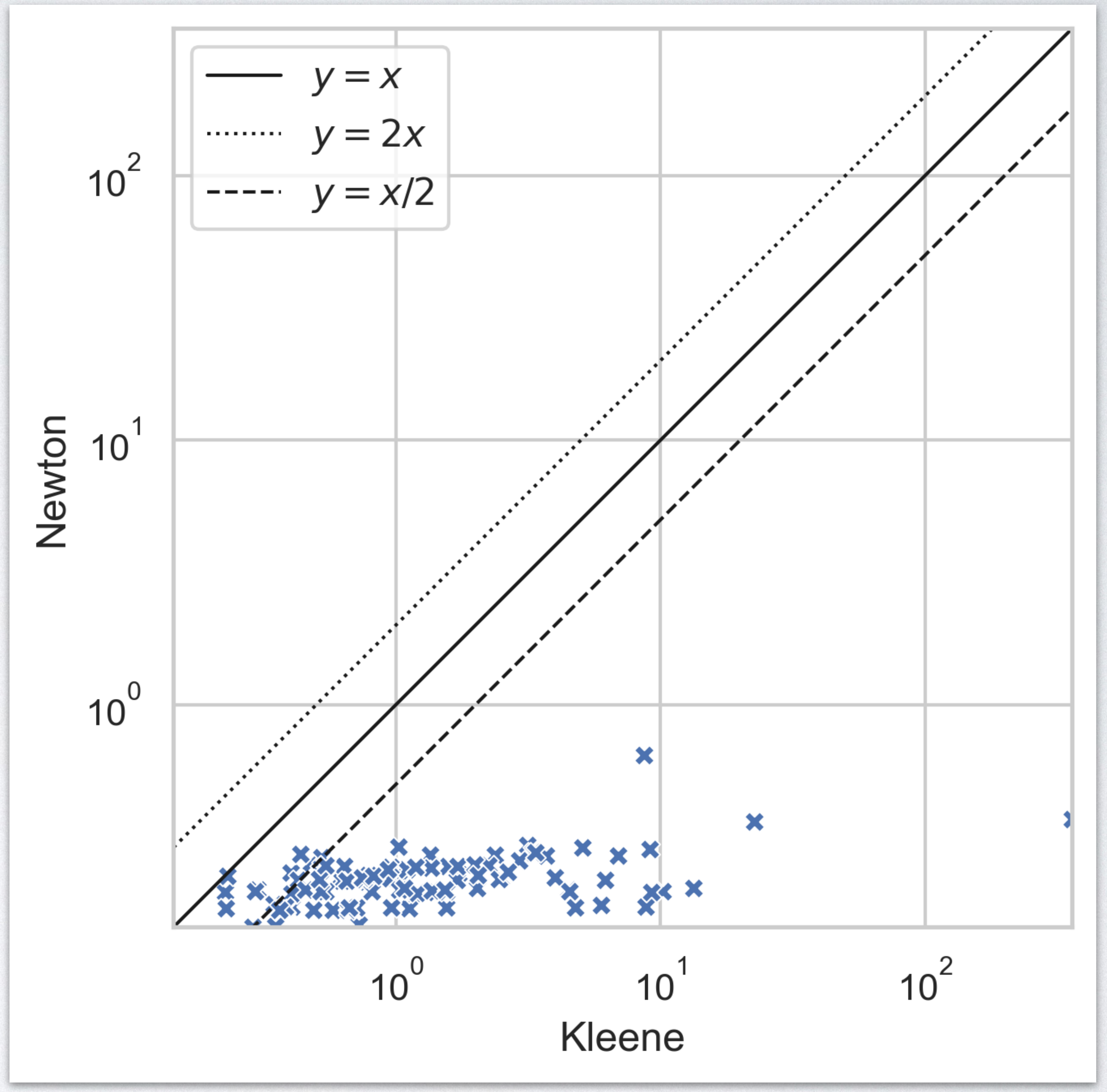
Linearized Equation System

Solve the linear system to get $\nu^{(i+1)}$

Preliminary Evaluation



Reaching Probability



Expected Reward

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
- ☑ **Representation:** Construction of Recursive Program Schemes
- ☑ **Algorithm:** Newton's Method for pre-Markov Algebras