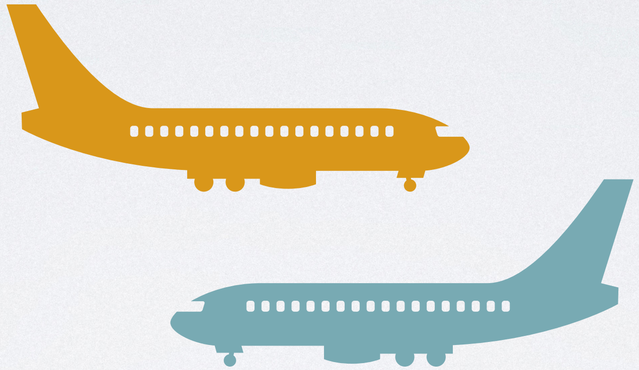
概率程序的代数程序分析

王迪
wangdi95@pku.edu.cn
2023 年 06 月 30 日

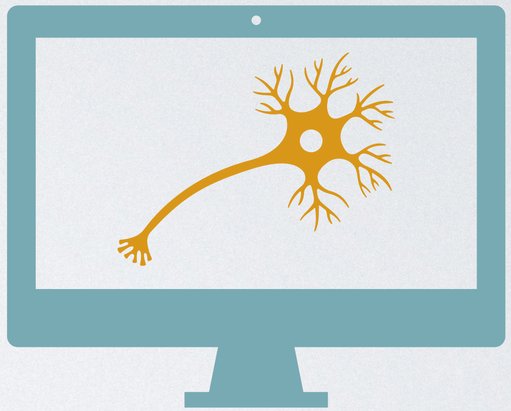
与 Jan Hoffmann 和 Thomas Reps 的合作工作

Probabilistic Systems are Becoming Pervasive

Randomized Algorithms
(improve efficiency)



Cyber-Physical Systems
(model uncertainty)



Artificial Intelligence
(describe statistical models)

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.

Application: Randomized Quicksort

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

Randomized Quicksort

Randomized-Partition(*A, p, r*)

1. $i \leftarrow$ *Random*(*p*, *r*)
2. exchange $A[r] \leftrightarrow A[i]$
3. **return Partition**(*A*, *p*, *r*)

Randomized-Quicksort(*A, p, r*)

1. **if** $p < r$
2. **then** $q \leftarrow$ **Randomized-Partition**(*A*, *p*, *r*)
3. **Randomized-Quicksort**(*A*, *p* , *q*-1)
4. **Randomized-Quicksort**(*A*, *q*+1, *r*)

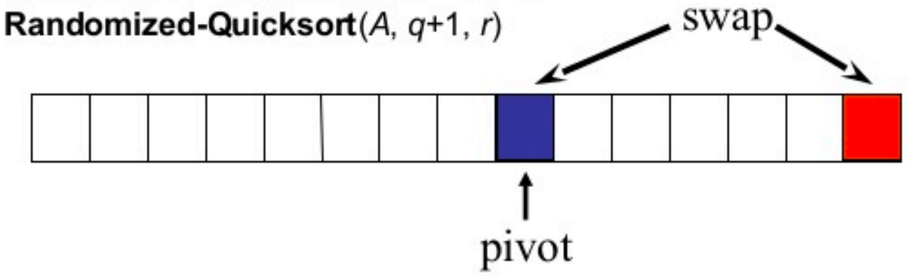


Image source: https://geekfactorial.blogspot.com/2016/08/randomized-quick-sort-algorithm.html.

Application: Airborne Collision Avoidance

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

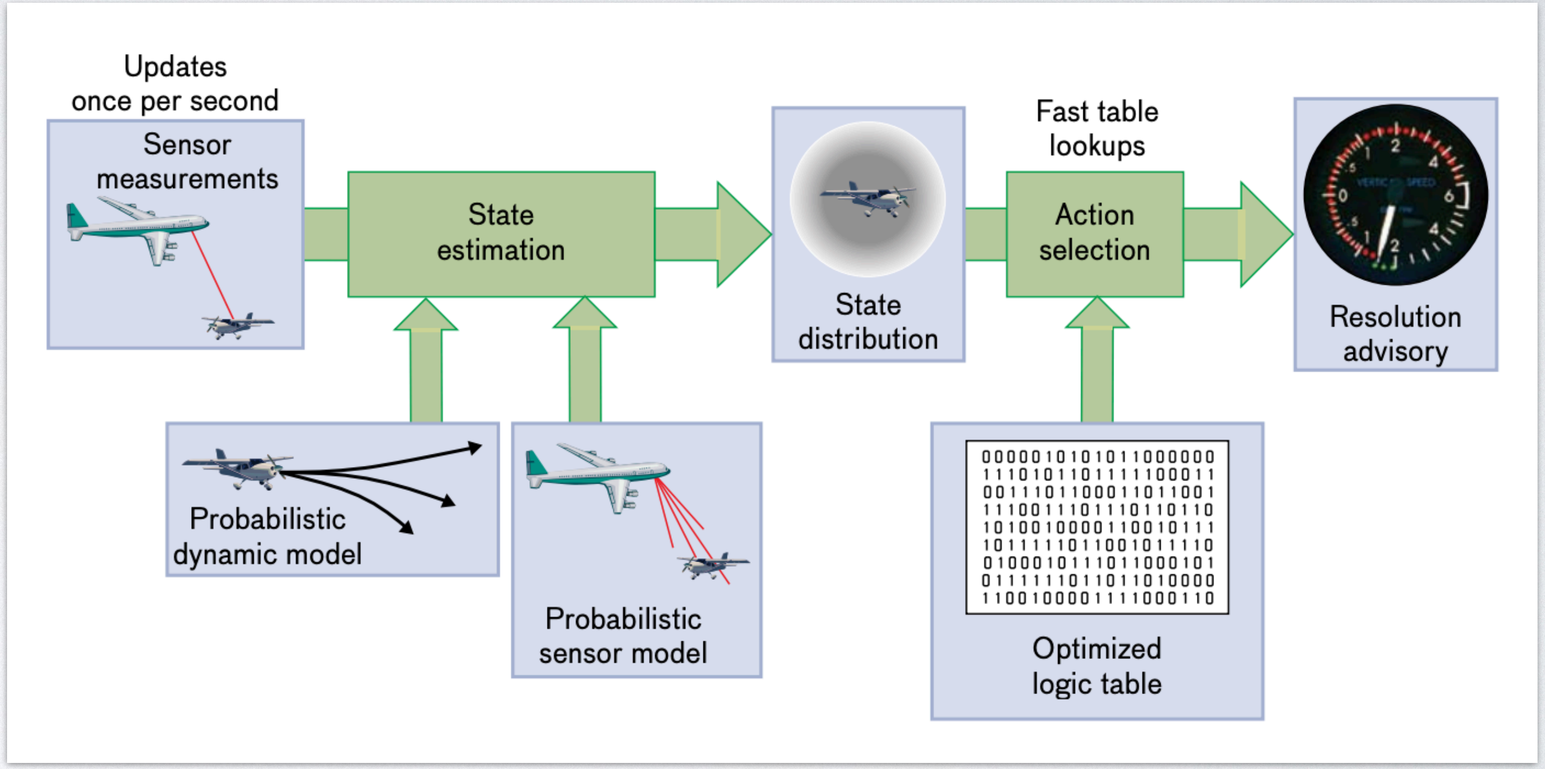
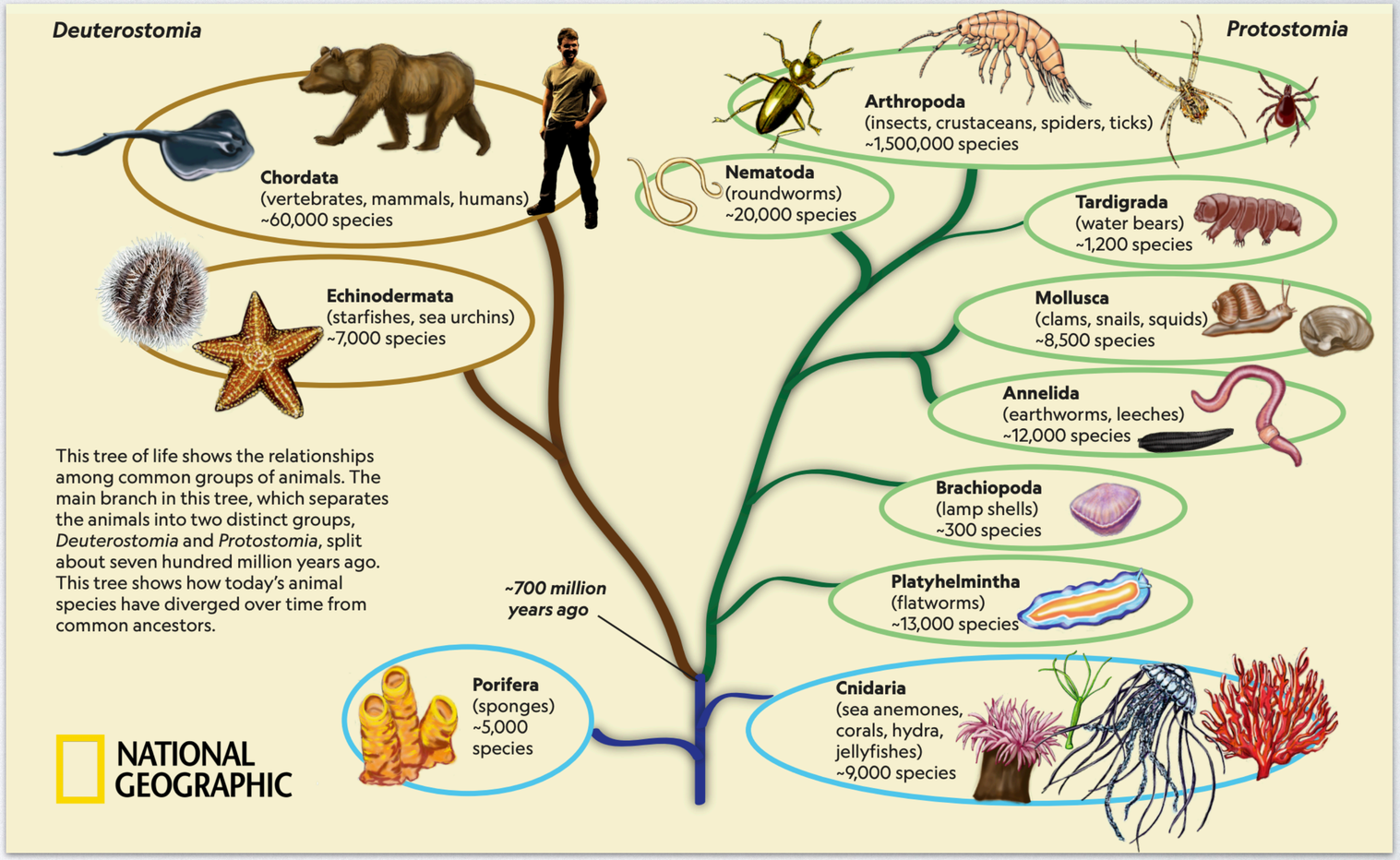


Image source: Kochenderfer, et al. “Next-Generation Airborne Collision Avoidance System.” Lincoln Laboratory Journal (2012).

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

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

```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi
```

Probabilistic Programs

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

```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi
```

```
cc :∈ (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 → pc := 1
| true → pc := 2
| true → pc := 3
fi
```

Demonic Programs

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

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

```
pc :∈ {1,2,3}
```

Example: Monty Hall

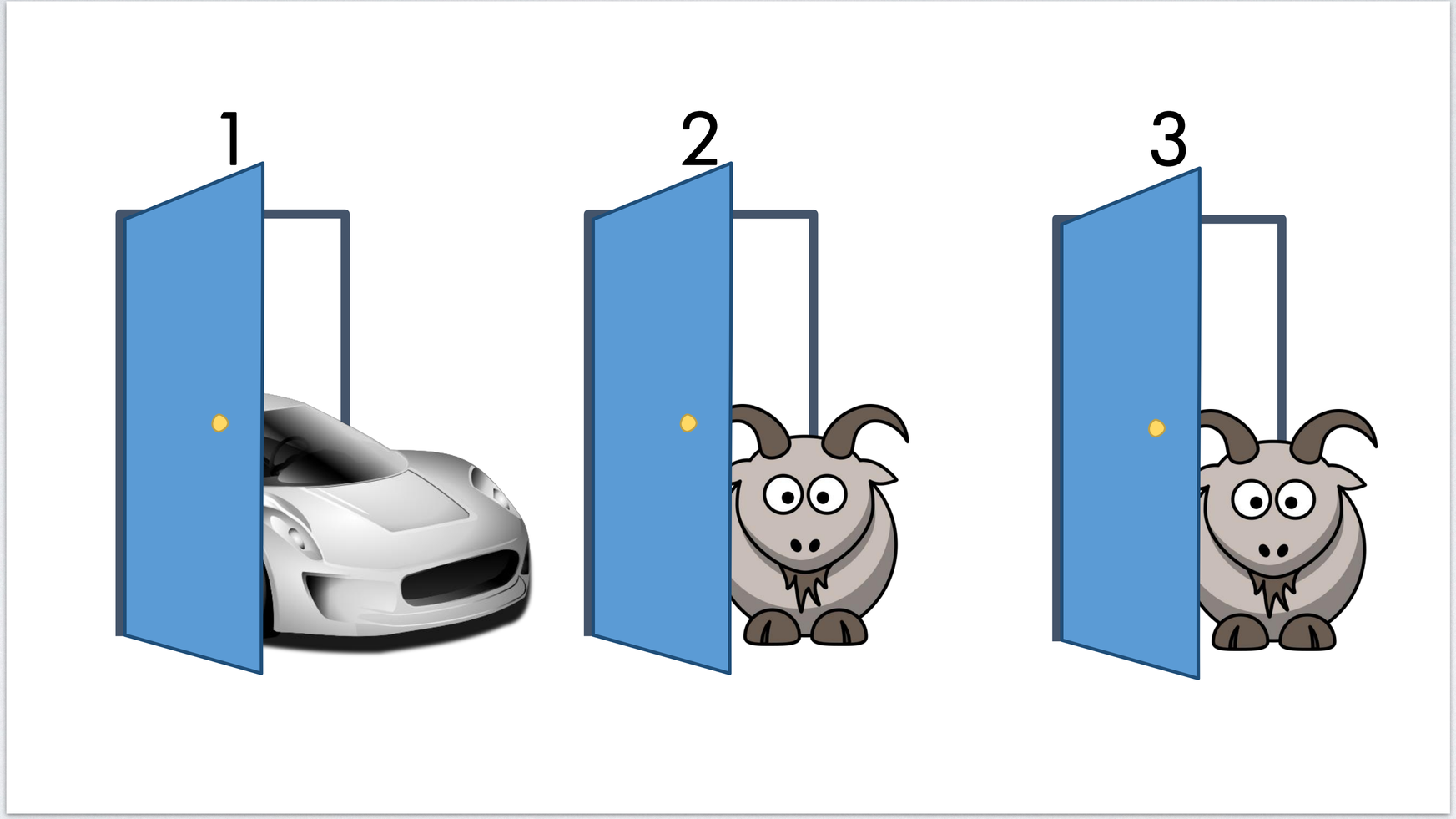


Image source: Maria Gorinova's Advances in Programming Languages (Guest Lecture) slides on Probabilistic Programming.

Example: Monty Hall

- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)
- Combine two forms of nondeterminism:
  - Probabilistic
  - Demonic
- What is the probability that `cc` equals to `pc`, if we switch to another door?
  - "Demons" minimize the probability

```
pc :∈ {1,2,3};
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
ac :∈ {1,2,3} \ {pc,cc};
if switch then
  cc :∈ {1,2,3} \ {cc,ac}
fi
```

Reasoning about pGCL Programs

Reasoning about pGCL Programs

- Dijkstra's GCL
  - Weakest pre-condition calculus
  - $\varphi_{pre} \Rightarrow \mathrm{wp}\,.\,prog\,.\,\varphi_{post}$

Reasoning about pGCL Programs

- Dijkstra's GCL
  - Weakest pre-condition calculus
  - $\varphi_{pre} \Rightarrow \mathrm{wp}\,.\,prog\,.\,\varphi_{post}$
- McIver and Morgan's pGCL
  - Weakest pre-expectation calculus
  - $f_{pre} \Rrightarrow \mathrm{wp}\,.\,prog\,.f_{post}$
  - $\mathrm{wp}\,.\,prog\,.\,[\varphi_{post}]$ gives the **greatest guaranteed** probability that φ_{post} holds

Reasoning about pGCL Programs

- Dijkstra's GCL
  - Weakest pre-condition calculus
  - $\varphi_{pre} \Rightarrow \mathrm{wp}\,.\,prog\,.\,\varphi_{post}$
- McIver and Morgan's pGCL
  - Weakest pre-expectation calculus
  - $f_{pre} \Rrightarrow \mathrm{wp}\,.\,prog\,.f_{post}$
  - $\mathrm{wp}\,.\,prog\,.\,[\varphi_{post}]$ gives the **greatest guaranteed** probability that φ_{post} holds

$$\{$$
$$1/3 * [1 \geq 2] + 1/3 * [2 \geq 2] + 1/3 * [3 \geq 2]$$
$$= 2/3$$
$$\}$$

```
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

$$\{\ [cc \geq 2]\ \}$$

Reasoning about pGCL Programs

- Dijkstra's GCL
  - Weakest pre-condition calculus
  - $\varphi_{pre} \Rightarrow \mathrm{wp}\,.\,prog\,.\,\varphi_{post}$
- McIver and Morgan's pGCL
  - Weakest pre-expectation calculus
  - $f_{pre} \Rrightarrow \mathrm{wp}\,.\,prog\,.f_{post}$
  - $\mathrm{wp}\,.\,prog\,.\,[\varphi_{post}]$ gives the **greatest guaranteed** probability that φ_{post} holds

$$\{\ 1/3 * [1 \geq 2] + 1/3 * [2 \geq 2] + 1/3 * [3 \geq 2] = 2/3\ \}$$

```
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

$$\{\ [cc \geq 2]\ \}$$

$$\{\ \min([1 \geq 2], [2 \geq 2], [3 \geq 2]) = 0\ \}$$

```
pc :∈ {1,2,3}
```

$$\{\ [pc \geq 2]\ \}$$

Example: Monty Hall

$\{\ [\neg switch] * 1/3 + [switch] * 2/3\ \}$

```
pc :∈ {1,2,3};
```

$\{$

$[\neg switch]/3 * ([pc = 1] + [pc = 2] + [pc = 3]) + [switch]/3 * ([pc \neq 1] + [pc \neq 2] + [pc \neq 3])$

$\}$

```
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
```

$\{\ [\neg switch] * [cc = pc] + [switch] * [cc \neq pc]\ \}$

```
ac :∈ {1,2,3} \ {pc,cc};
```

$\{\ [\neg switch] * [cc = pc] + [switch] * [\{cc, pc, ac\} = \{1,2,3\}]\ \}$

```
if switch then
  cc :∈ {1,2,3} \ {cc,ac}
fi
```

$\{\ [cc = pc]\ \}$

How to automate such quantitative reasoning about probabilistic programs?

How to automate such quantitative reasoning about probabilistic programs?

Examples

What is the probability that an assertion holds?

Is there any expectation invariant for a probabilistic loop?

What is the expected time complexity of a program?

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

Semantics: Support different semantics for nondeterminism

Representation: Extend pGCL with unstructured control-flow

Algorithm: Solve program analyses in a non-iterative way

Our Papers

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

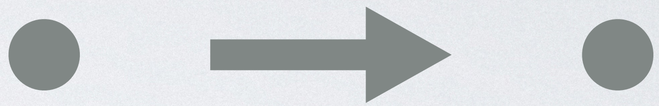
- Di Wang, Jan Hoffmann, Thomas Reps (2019). **A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism**. In *MFPS'19*.

- Di Wang, Thomas Reps (2023). **Newtonian Program Analysis of Probabilistic Programs**. *Working Paper*.

Denotational Semantics

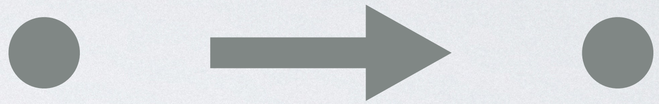
Denotational Semantics

Standard: $State \rightarrow State$



Denotational Semantics

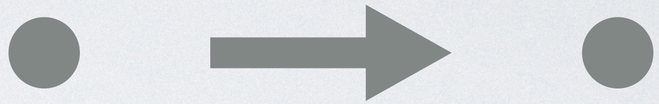
Standard: $State \rightarrow State$



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

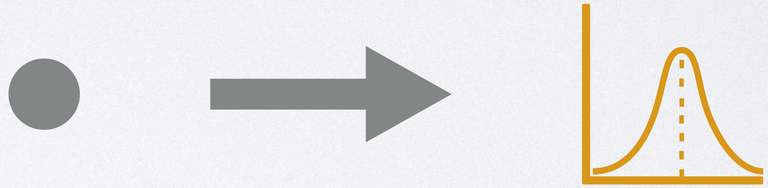
Denotational Semantics

Standard: $State \to State$



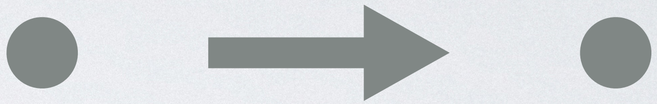
GCL: $State \to \wp(State)$

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



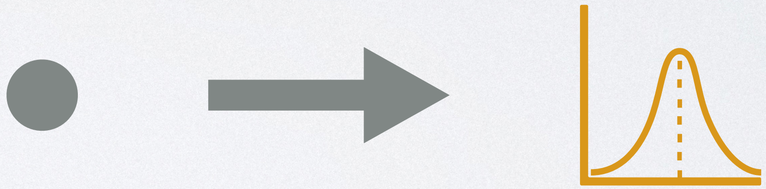
Denotational Semantics

Standard: $State \to State$

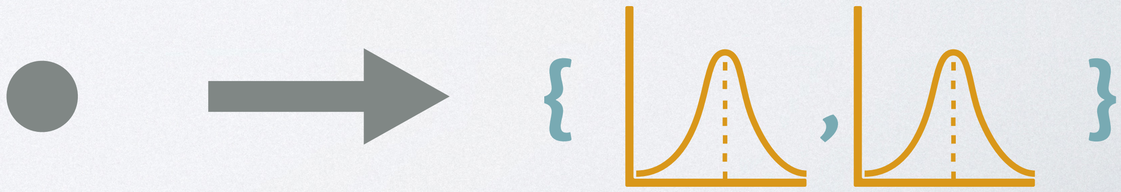


GCL: $State \to \wp(State)$

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



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

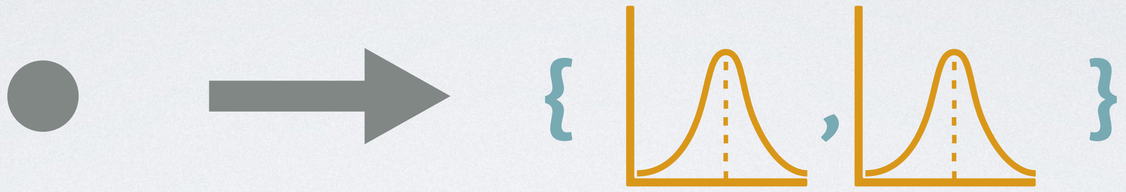


Different Models for Combining Nondeterminism

Different Models for Combining Nondeterminism

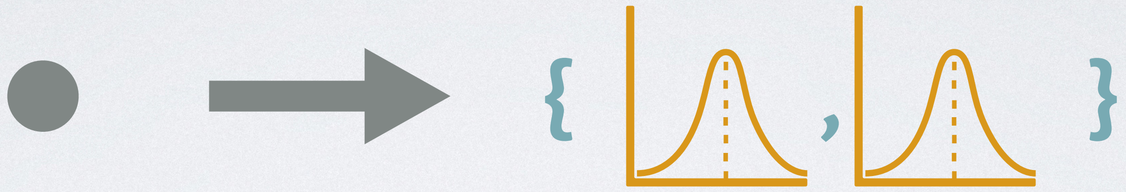
- pGCL:

$$State \to \wp(\mathbb{D}(State))$$

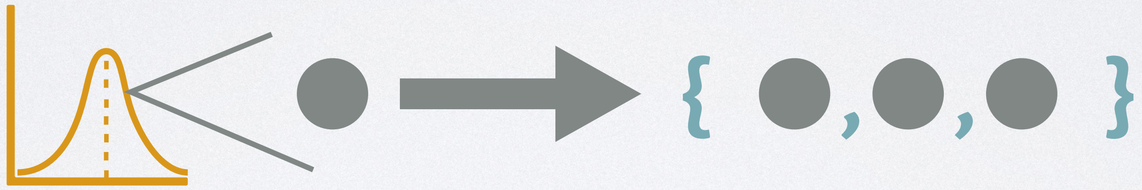


Different Models for Combining Nondeterminism

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

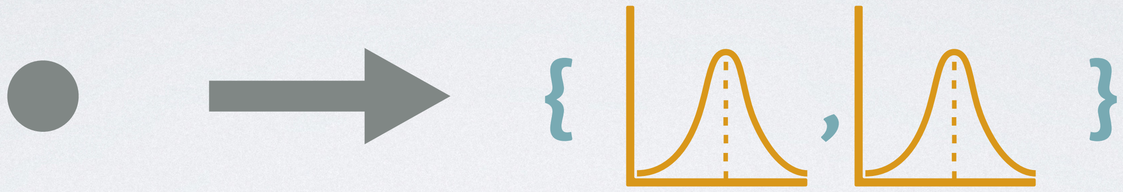


- Cousot's **Probabilistic Abstract Interpretation** (PAI): $\mathbb{D}(State \to \wp(State))$

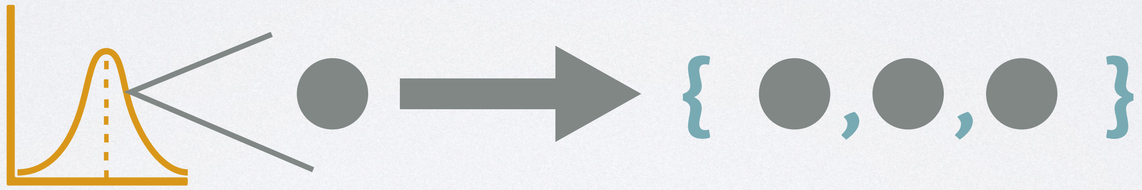


Different Models for Combining Nondeterminism

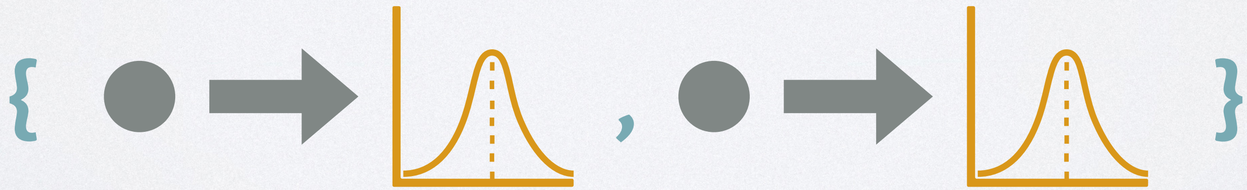
- pGCL: $State \to \wp(\mathbb{D}(State))$



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



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



Our Approach: Markov Algebras

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

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

Our Approach: Markov Algebras

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

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

Program state transformers
form a complete partial order

Our Approach: Markov Algebras

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

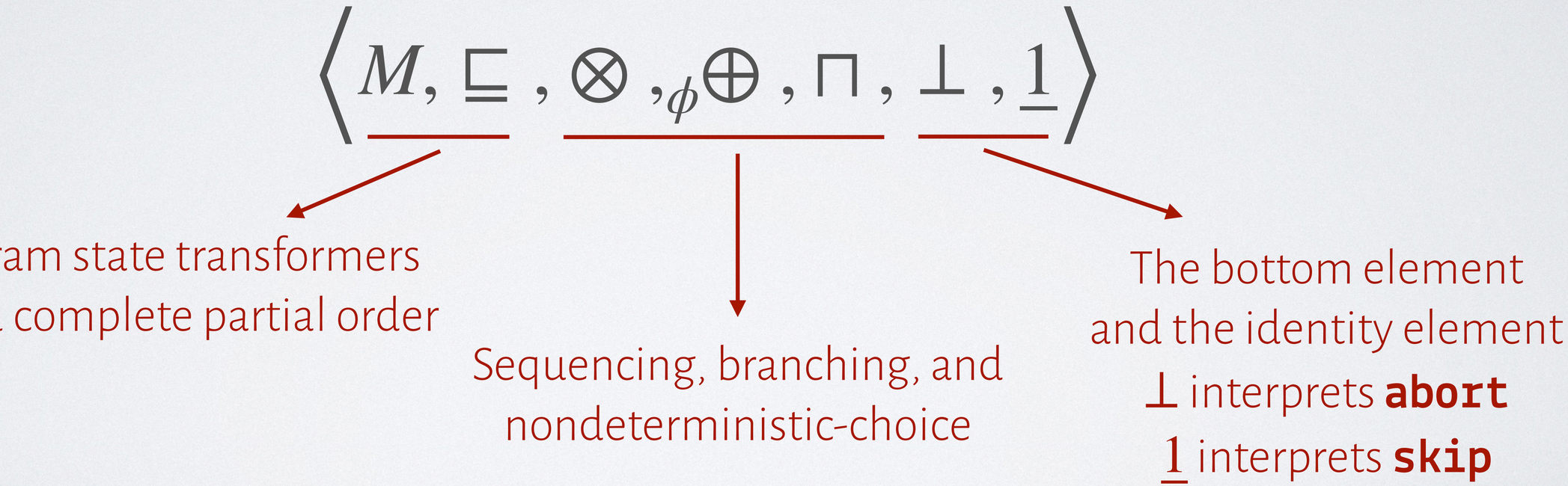
$$\left\langle M, \sqsubseteq, \otimes, {}_{\phi}\oplus, \sqcap, \bot, \underline{1} \right\rangle$$

Program state transformers form a complete partial order

Sequencing, branching, and nondeterministic-choice

Our Approach: Markov Algebras

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



Our Approach: Markov Algebras

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

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

Program state transformers form a complete partial order

Sequencing, branching, and nondeterministic-choice

Our Approach: Markov Algebras

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

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

Program state transformers form a complete partial order

Sequencing, branching, and nondeterministic-choice

$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$
$$a \otimes \underline{1} = \underline{1} \otimes a = a$$
$$a {}_{\phi}\oplus b = b {}_{\overline{\phi}}\oplus a$$
$$a \sqcap a = a$$
$$\dots$$

Markov Algebras Suffice!

Markov Algebras Suffice!

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

Markov Algebras Suffice!

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

$$(x \coloneqq 1 \oplus_{1/2} x \coloneqq 2) \sqcap (x \coloneqq 3 \oplus_{1/2} x \coloneqq 4)$$

Markov Algebras Suffice!

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

$$(x \coloneqq 1 \oplus_{1/2} x \coloneqq 2) \sqcap (x \coloneqq 3 \oplus_{1/2} x \coloneqq 4)$$

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

Markov Algebras Suffice!

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

$$(x \coloneqq 1 \oplus_{1/2} x \coloneqq 2) \sqcap (x \coloneqq 3 \oplus_{1/2} x \coloneqq 4)$$

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

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

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

☑ **Semantics:** Markov Algebras for Multiple Kinds of Confluence

Representation: Extend pGCL with unstructured control-flow

Algorithm: Solve program analyses in a non-iterative way

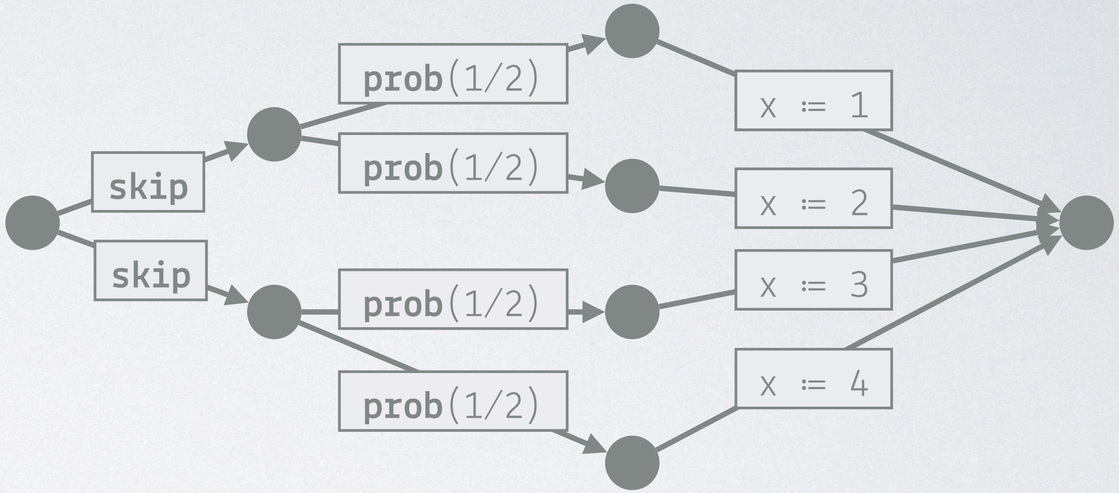
Do Control-flow Graphs Suffice?

Do Control-flow Graphs Suffice?

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

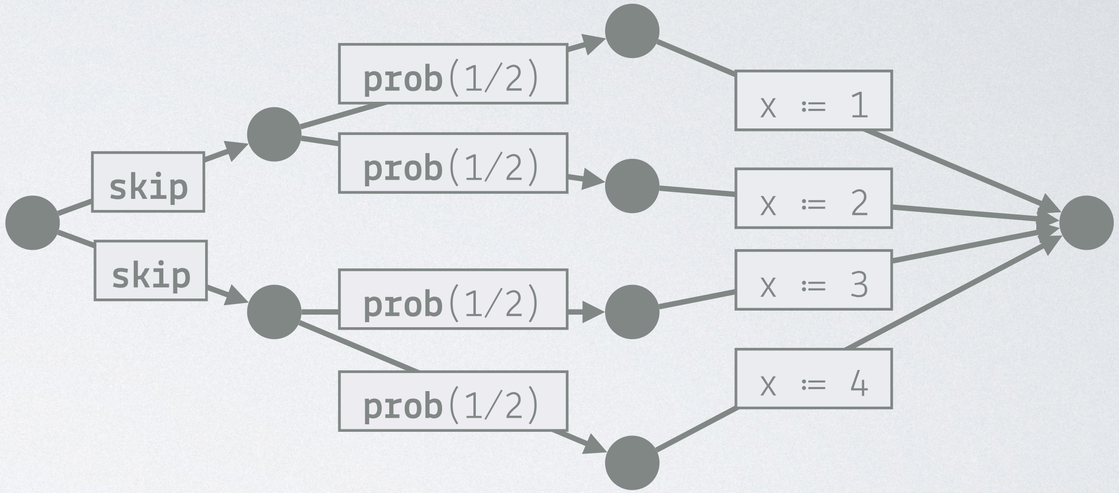
Do Control-flow Graphs Suffice?

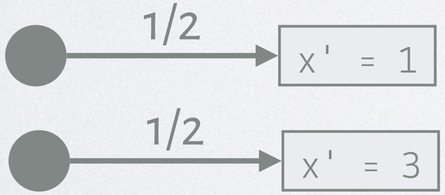
```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```

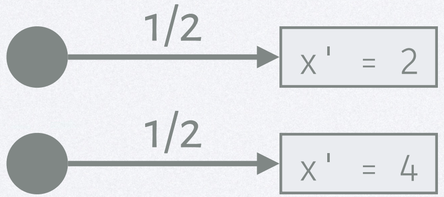


Do Control-flow Graphs Suffice?

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



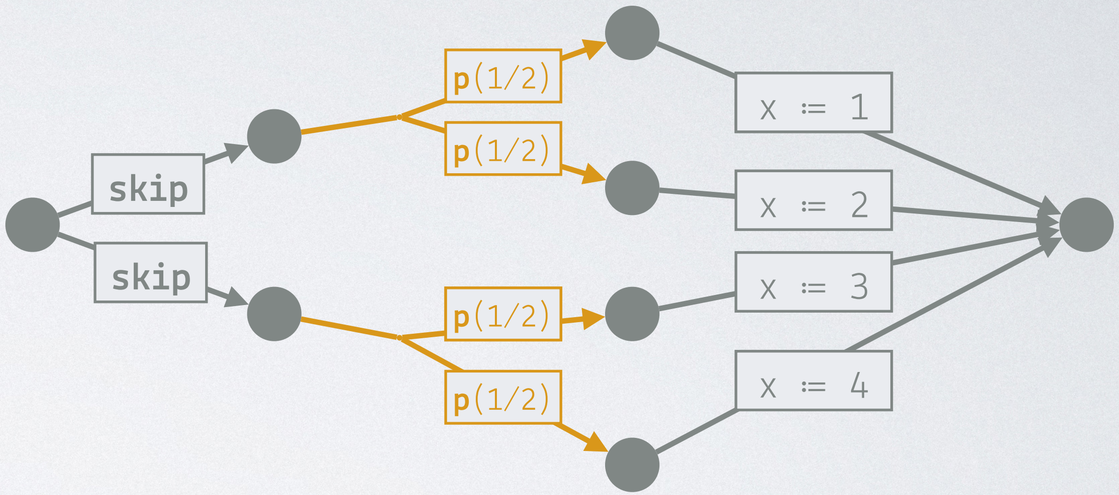




Probabilities sum up to 2!

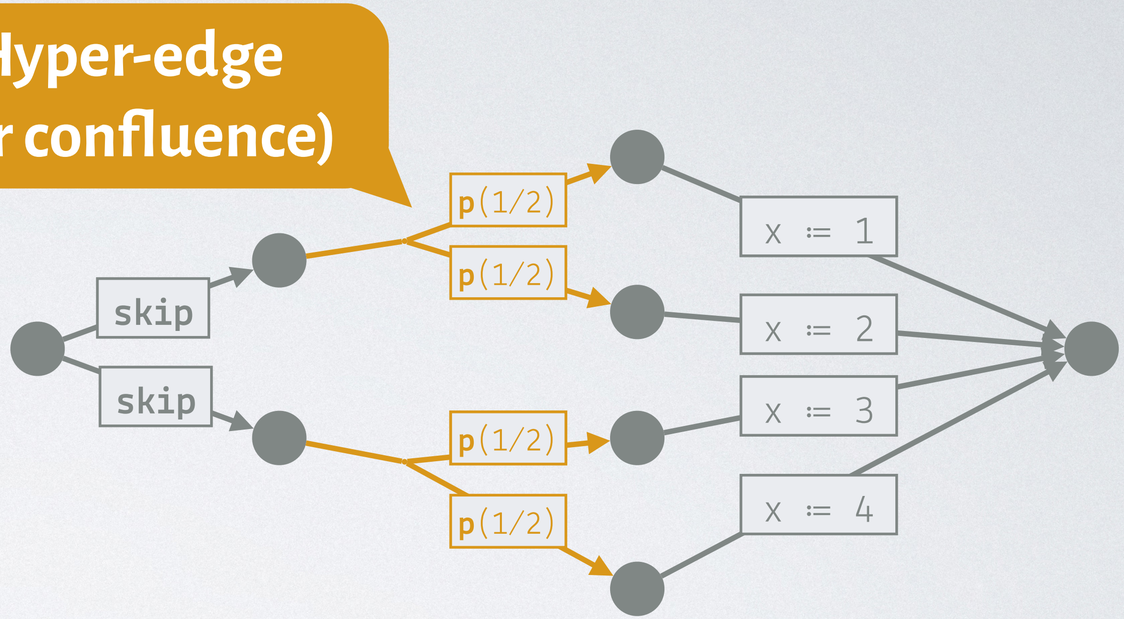
Our Approach: Control-flow Hyper-graphs

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



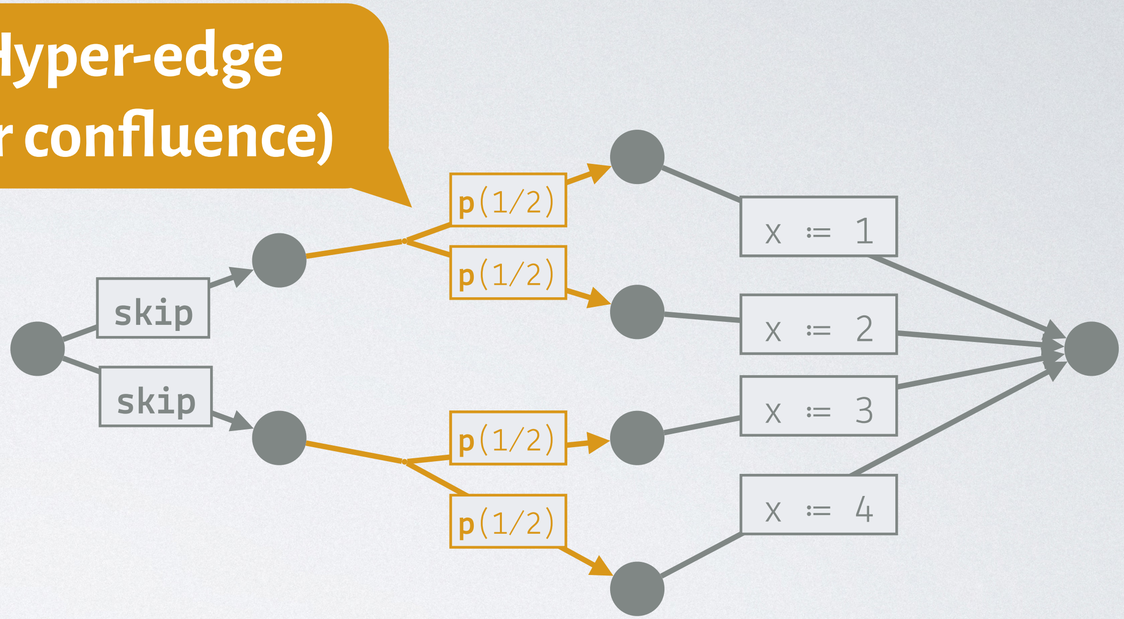
Our Approach: Control-flow Hyper-graphs

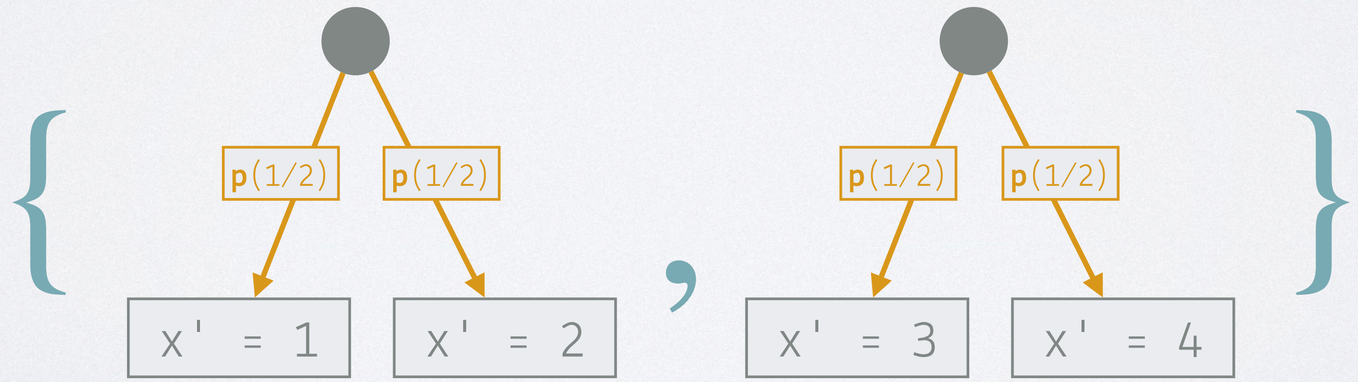
```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```



Our Approach: Control-flow Hyper-graphs

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

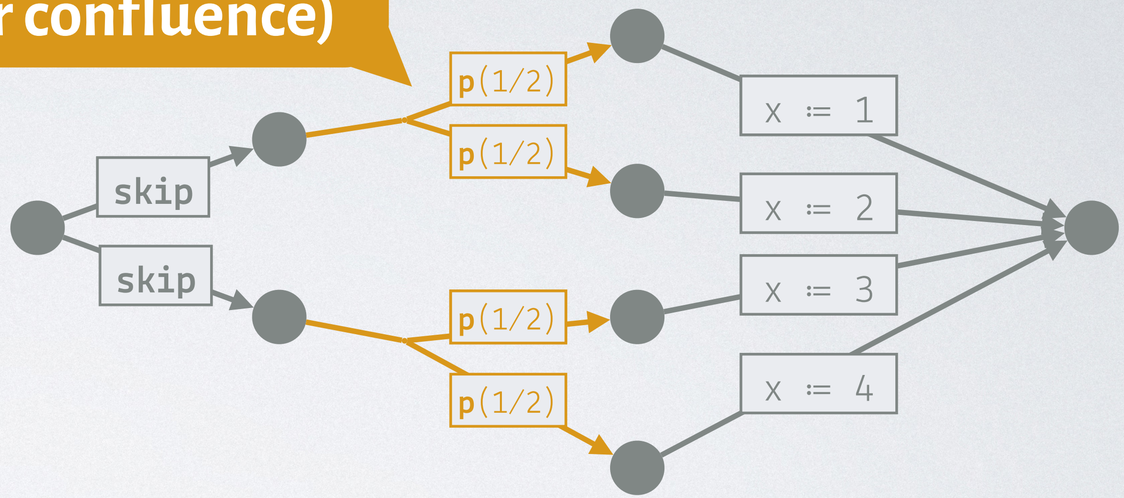




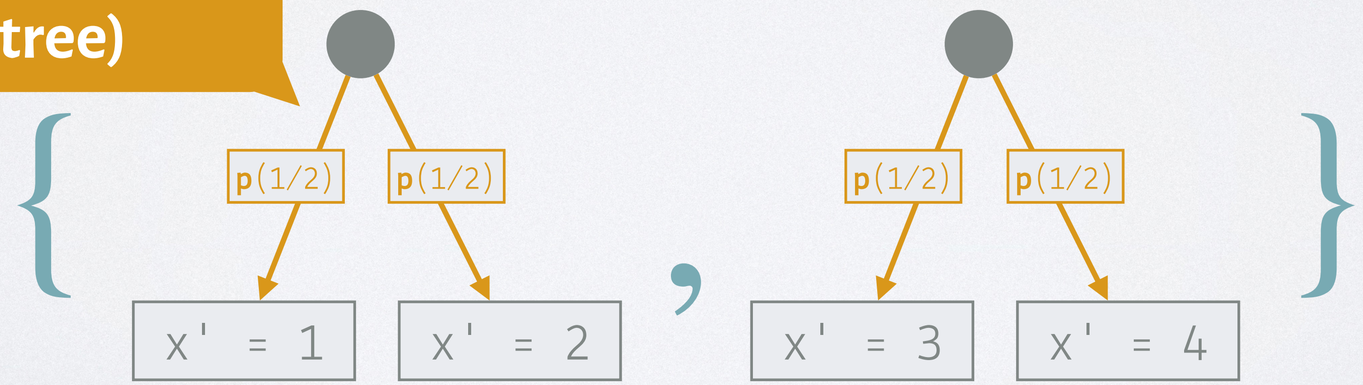
Our Approach: Control-flow Hyper-graphs

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

Hyper-edge (for confluence)

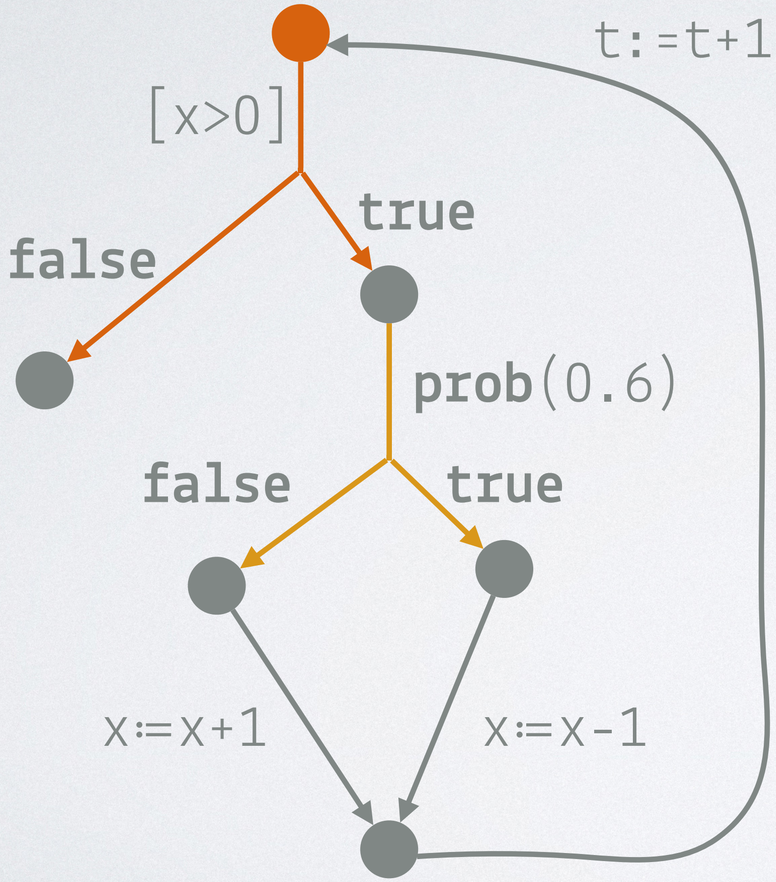


Hyper-path (like a tree)

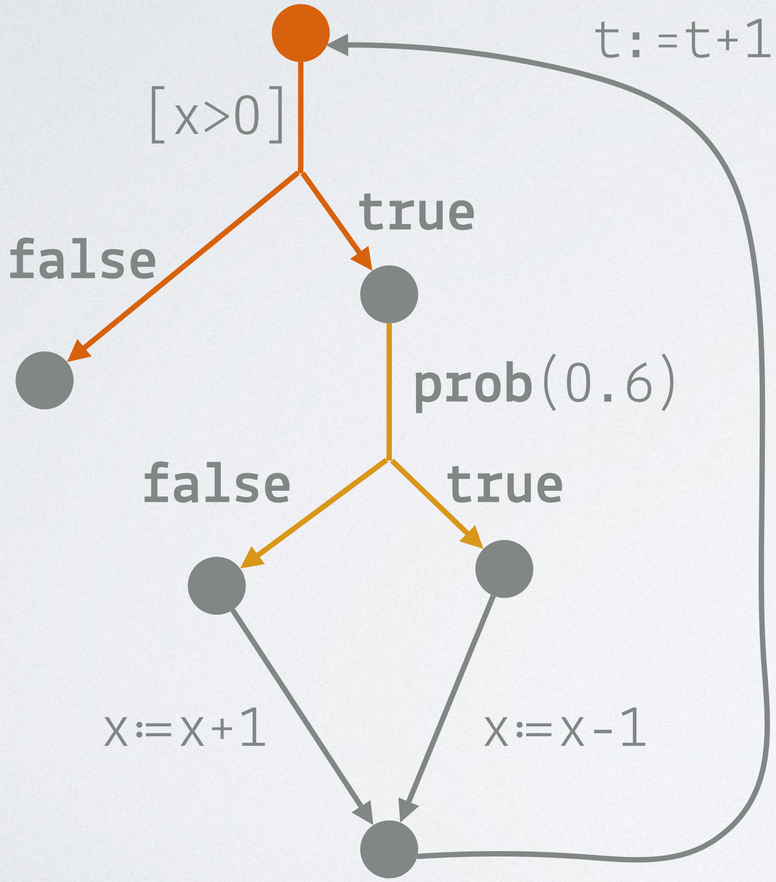


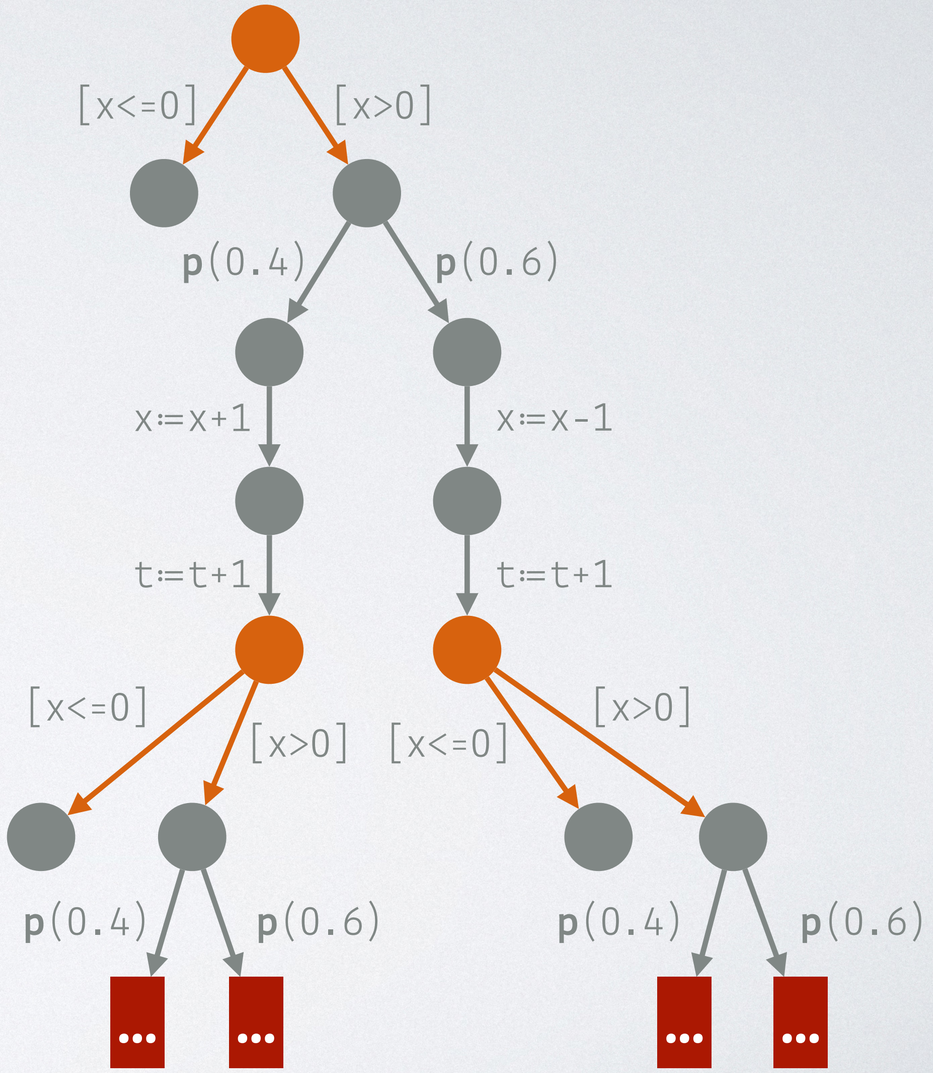
Hyper-paths are Infinite Trees!

Hyper-paths are Infinite Trees!

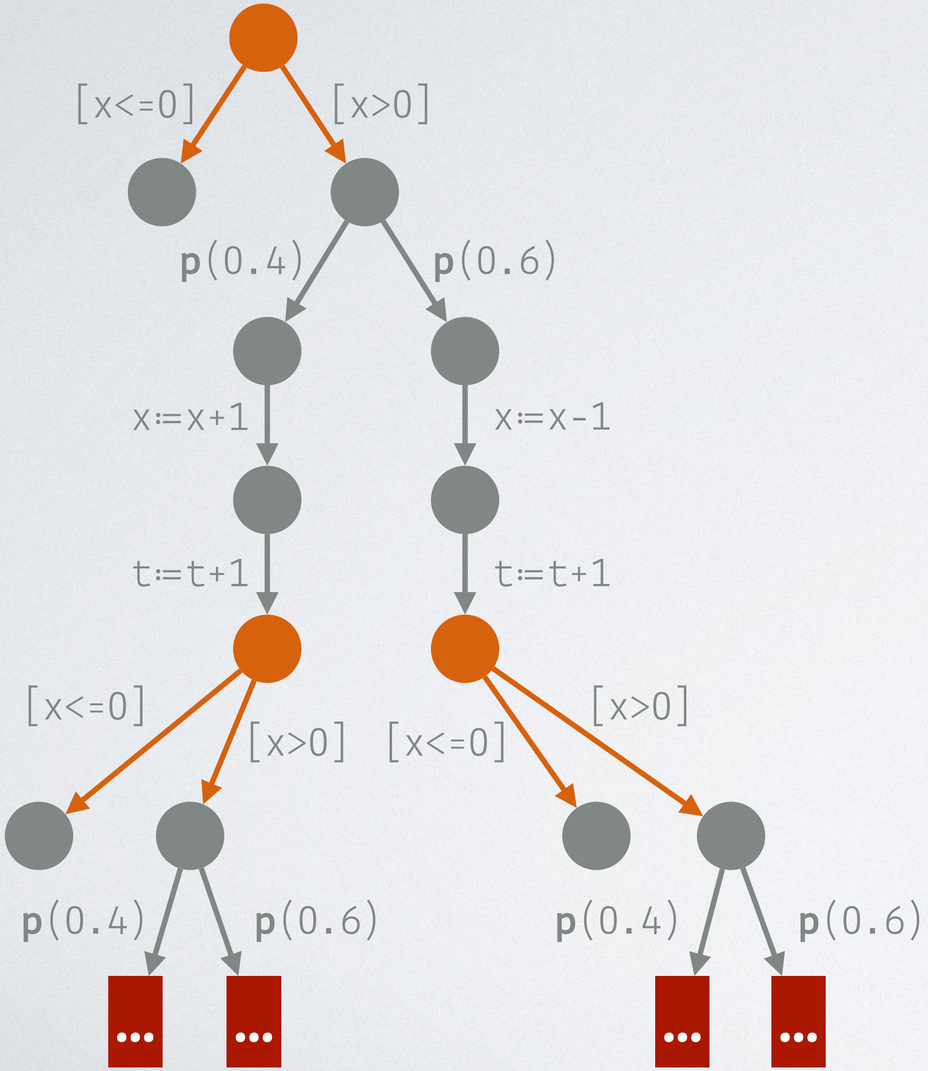


Hyper-paths are Infinite Trees!

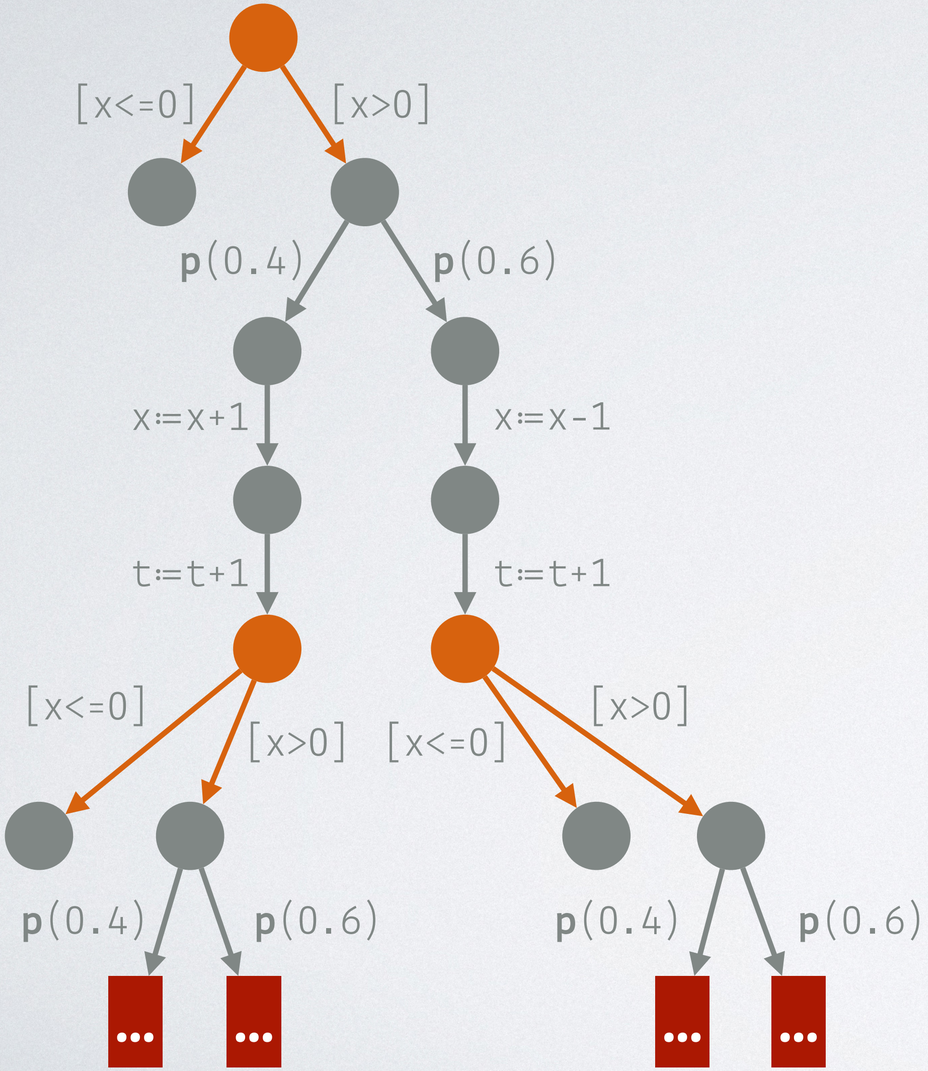


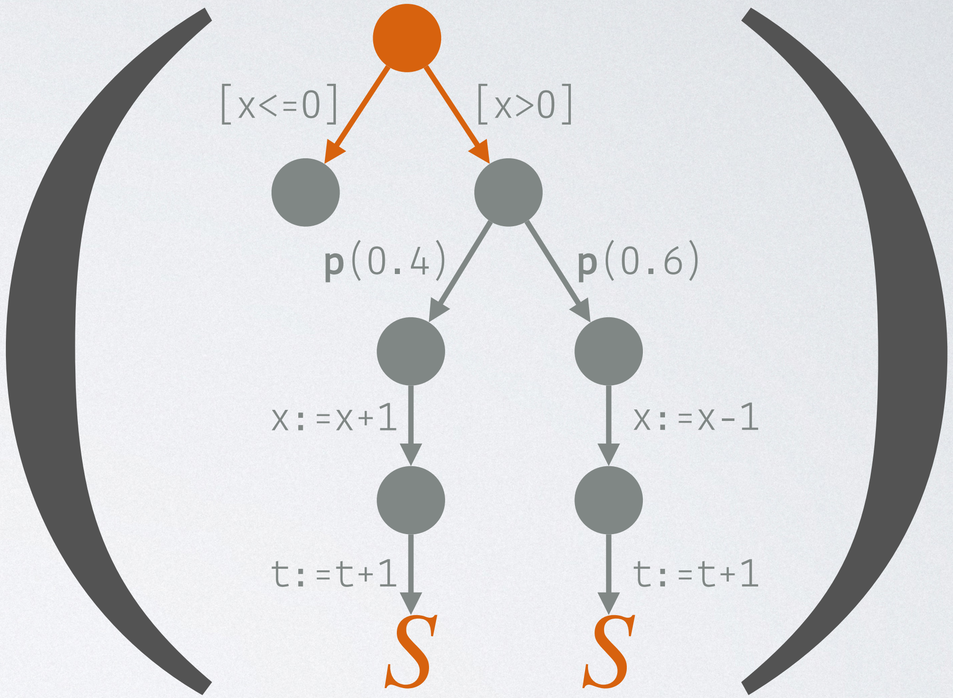


Regular Infinite-tree Expressions

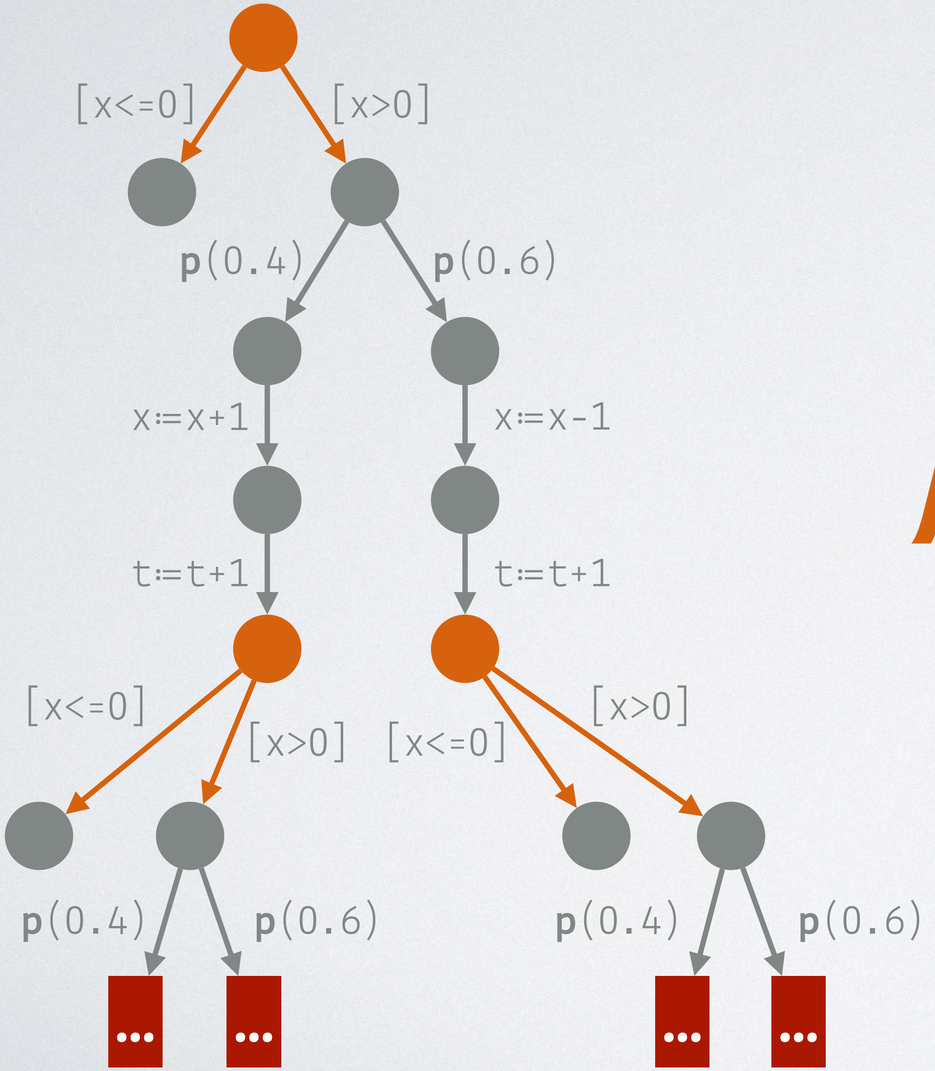


Regular Infinite-tree Expressions

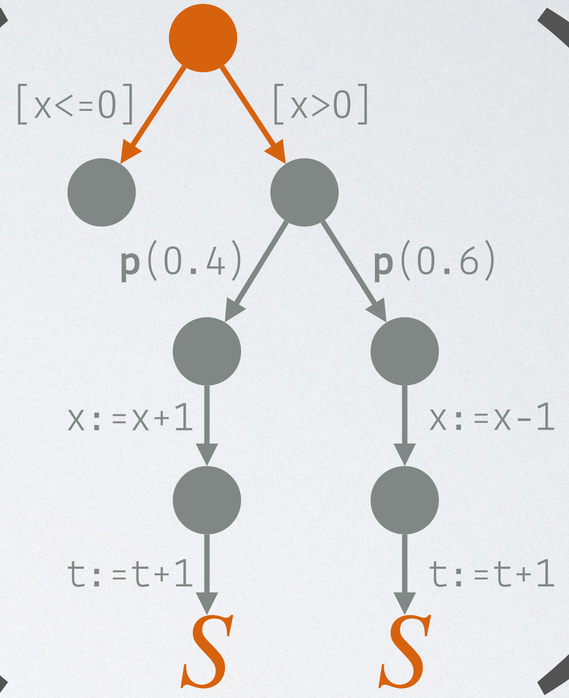




Regular Infinite-tree Expressions

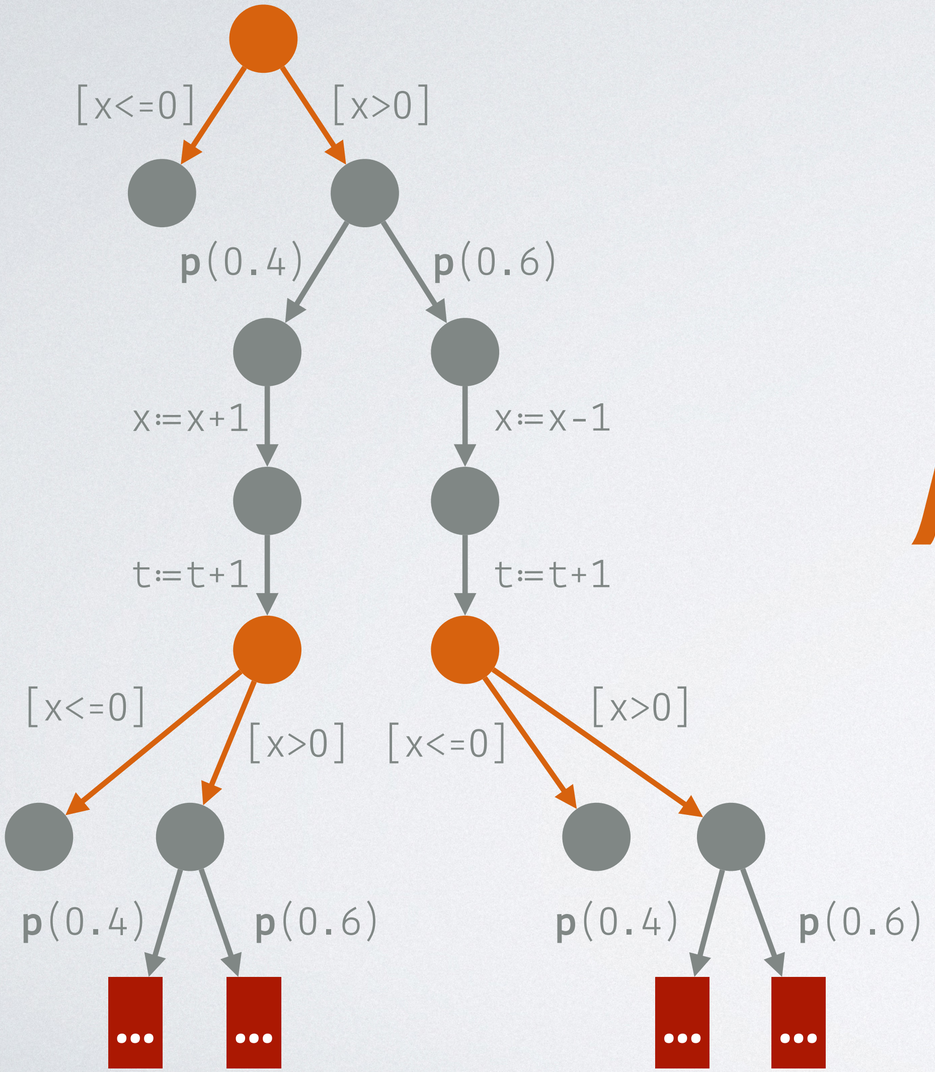


$$\mu S . \left(\right.$$

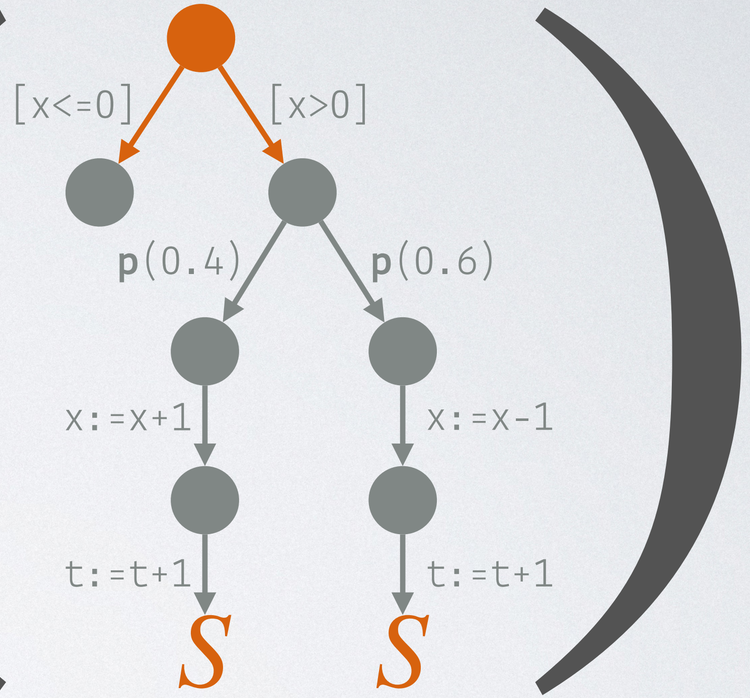


$$\left. \right)$$

Regular Infinite-tree Expressions



$\mu S .$



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

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

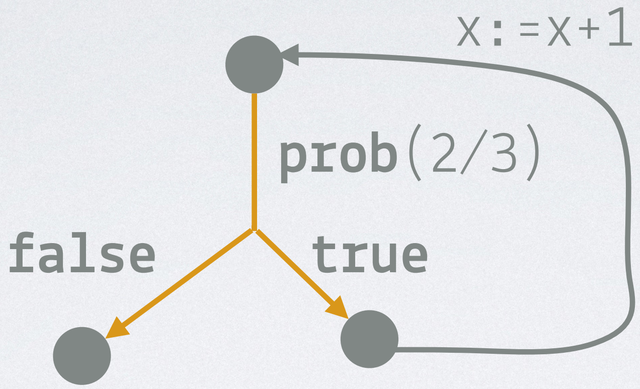
☑ **Semantics:** Markov Algebras for Multiple Kinds of Confluence

☑ **Representation:** Control-flow Hyper-graphs and Tree Expressions

Algorithm: Solve program analyses in a non-iterative way

Iterative Program Analysis

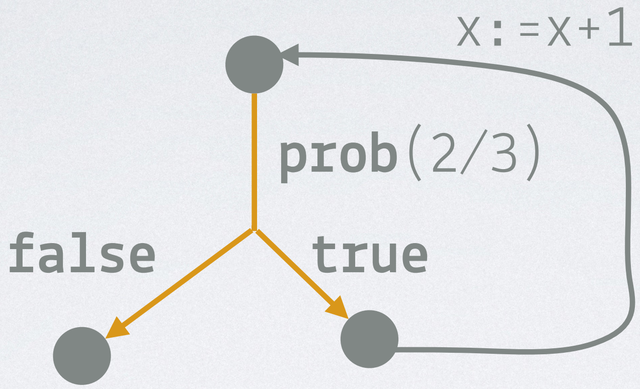
```
while prob(2/3) do
   x := x + 1
od
```



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

Iterative Program Analysis

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

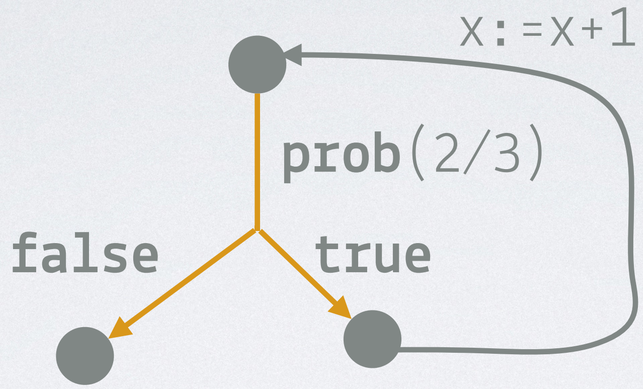


$$\mu S \,.\, ((\mathsf{x} \coloneqq \mathsf{x}{+}1) \otimes S)_{[2/3]} \oplus \mathbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r_1 \otimes r_2 \triangleq r_1 + r_2$
  - Branching: $r_{1p} \otimes r_2 \triangleq p * r_1 + (1-p) * r_2$

Iterative Program Analysis

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



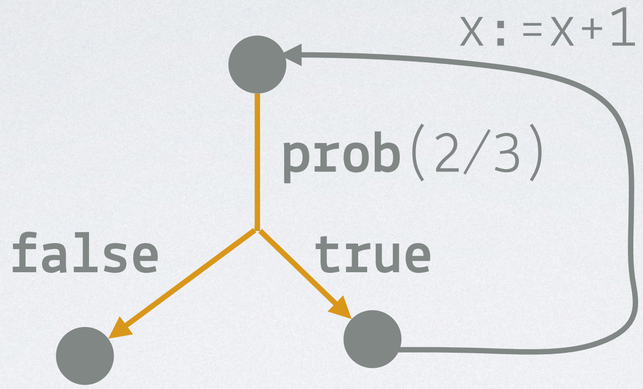
$$\mu S\,.\,((\mathtt{x} \coloneqq \mathtt{x}{+}1) \otimes S)_{[2/3]} \oplus \mathbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r_1 \otimes r_2 \triangleq r_1 + r_2$
  - Branching: $r_{1p} \otimes r_2 \triangleq p * r_1 + (1-p) * r_2$

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

Iterative Program Analysis

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



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

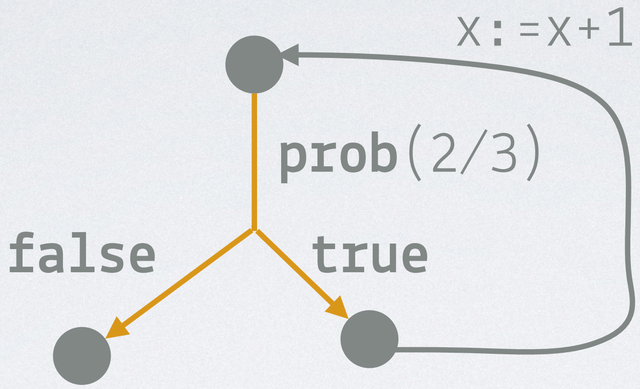
- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r_1 \otimes r_2 \triangleq r_1 + r_2$
  - Branching: $r_{1p} \otimes r_2 \triangleq p * r_1 + (1-p) * r_2$

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

Need ∞ iterations to converge!

Non-iterative Program Analysis

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

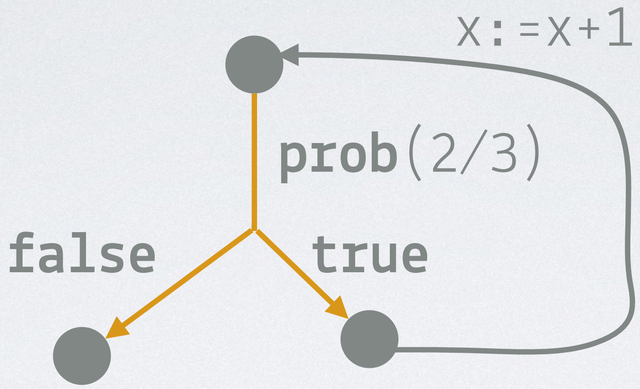


$$\mu S\,.\,((\mathsf{x} \coloneqq \mathsf{x}{+}1) \otimes S)_{[2/3]} \oplus \mathbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r_1 \otimes r_2 \triangleq r_1 + r_2$
  - Branching: $r_{1p} \otimes r_2 \triangleq p * r_1 + (1-p) * r_2$

Non-iterative Program Analysis

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



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

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r_1 \otimes r_2 \triangleq r_1 + r_2$
  - Branching: $r_{1p} \otimes r_2 \triangleq p * r_1 + (1-p) * r_2$

This analysis is equivalent to solve

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

which can be solve analytically:

$$s = 2$$

No need for iteration!

Inter-procedural Analysis

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

Inter-procedural Analysis

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

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

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]

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

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]

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

Solve the quadratic equation analytically: $p = 1/2$

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]

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

Solve the quadratic equation analytically: $p = 1/2$

Newtons's method!

Newton's Method Converges Faster!

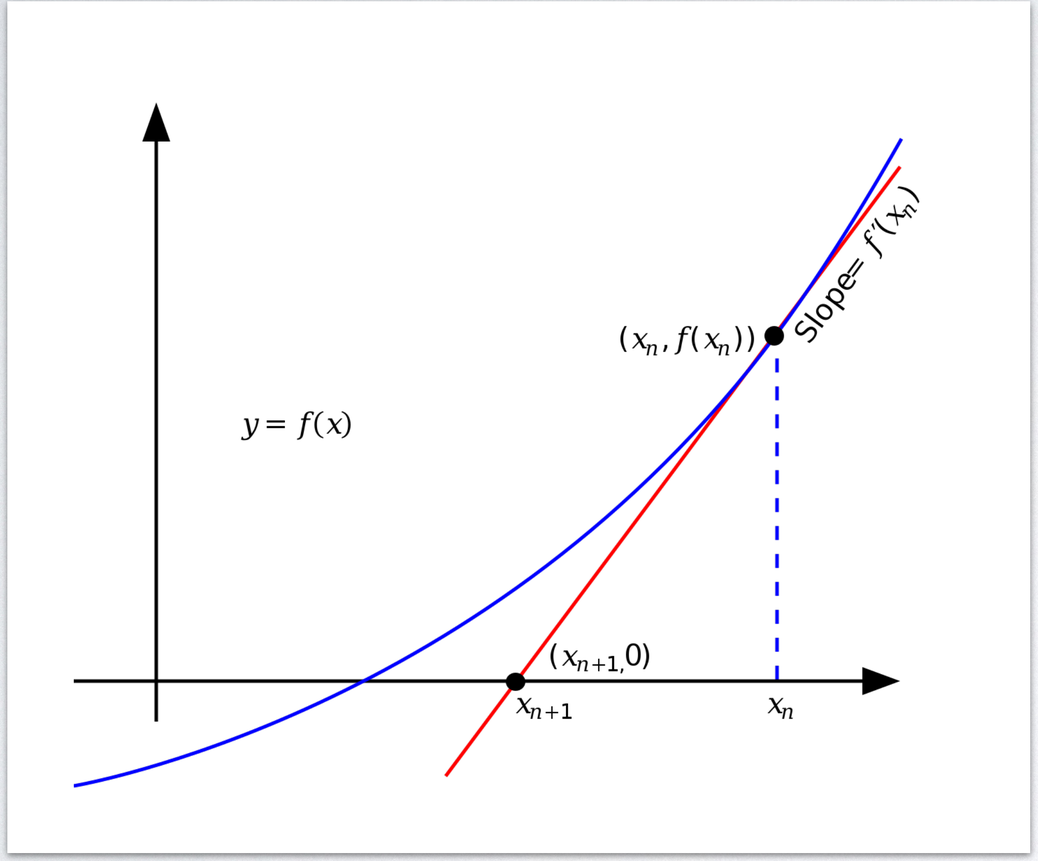
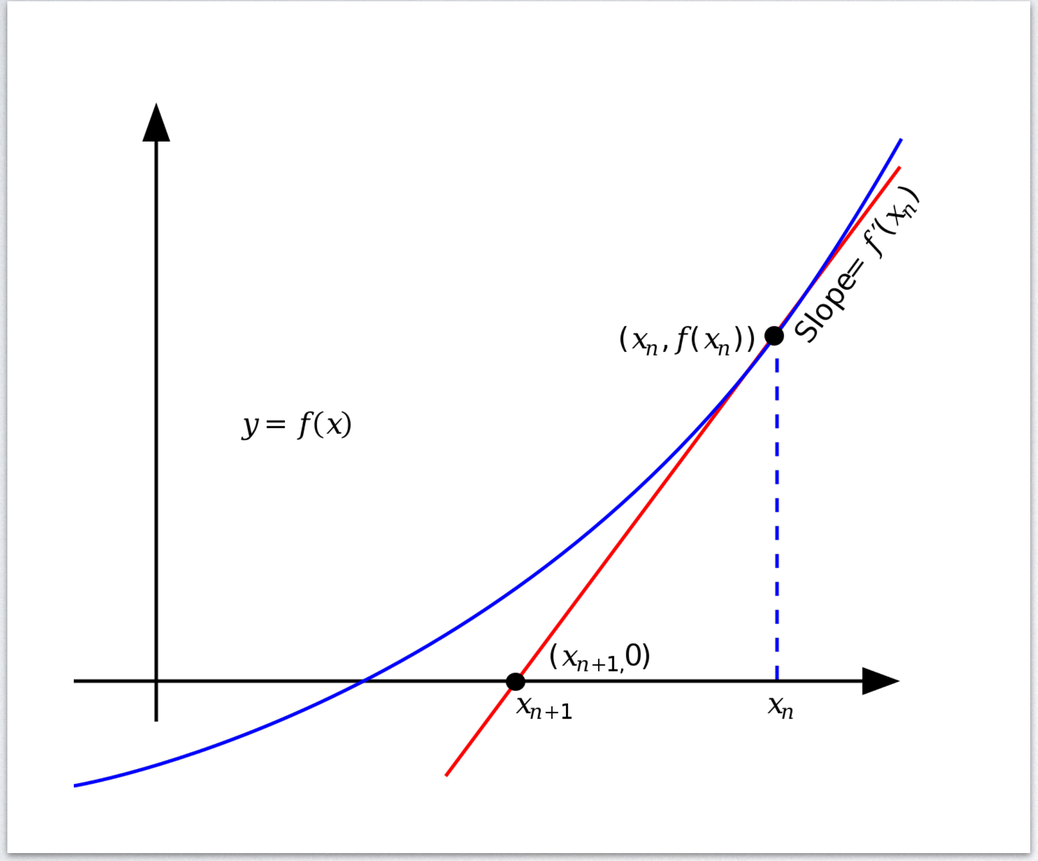


Image source: https://commons.wikimedia.org/wiki/File:Newton_iteration.svg.

Newton's Method Converges Faster!



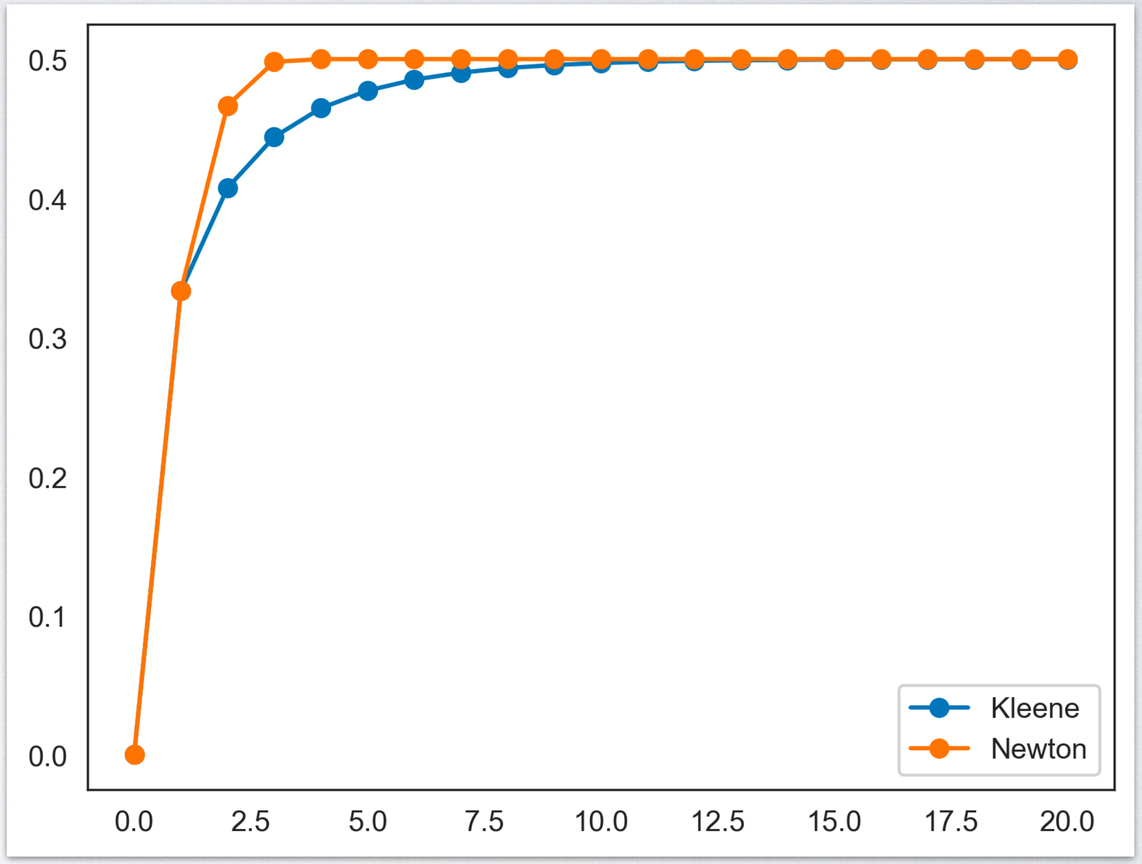


Image source: https://commons.wikimedia.org/wiki/File:Newton_iteration.svg.

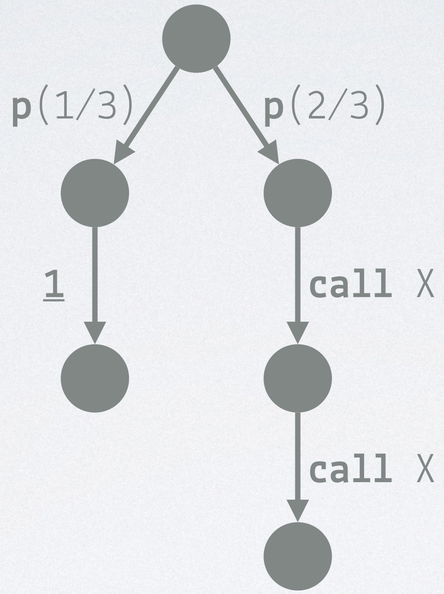
Newtonian Program Analysis (NPA)

```
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)

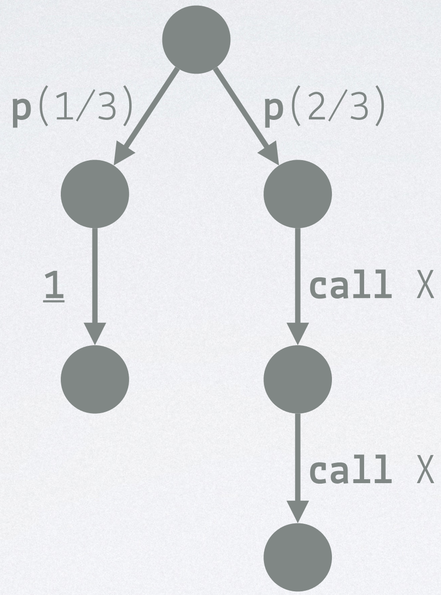
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



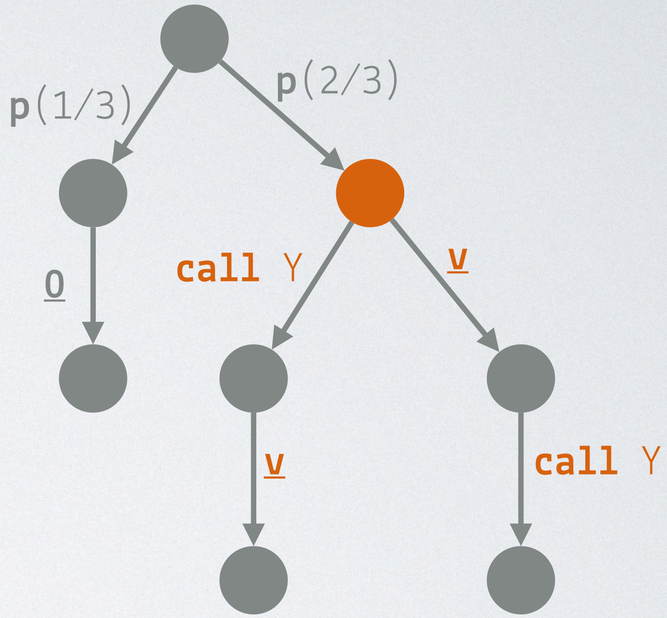
Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

Newtonian Program Analysis (NPA)

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



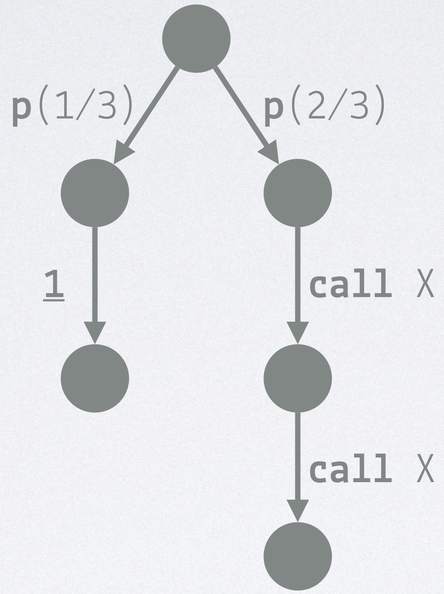
Differentiate



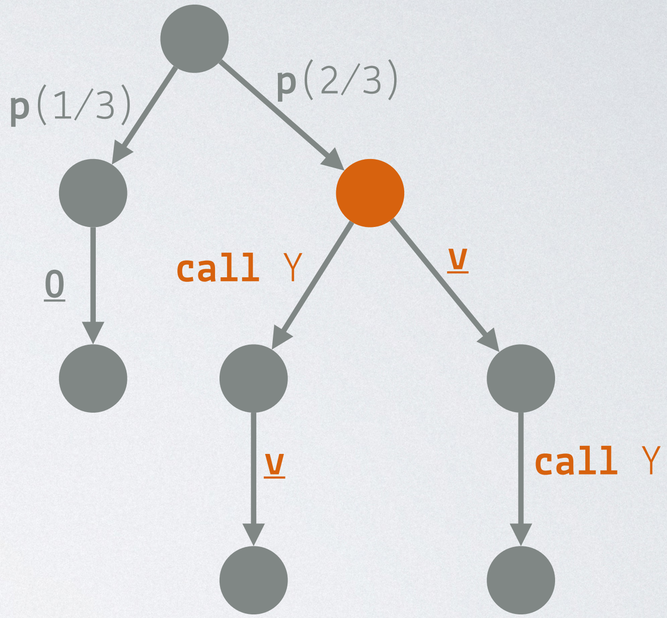
Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

Newtonian Program Analysis (NPA)

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



Differentiate

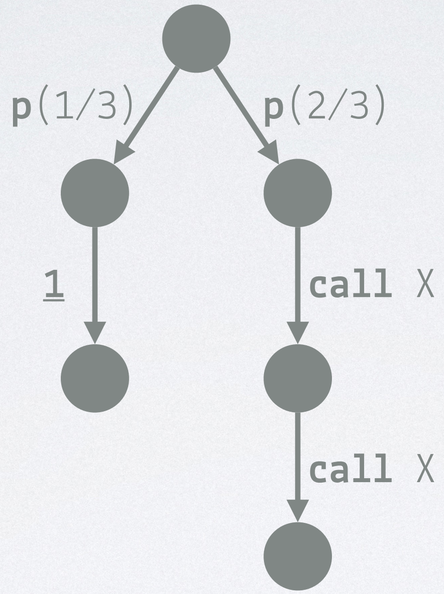


Every root-to-leaf path contains **at most one call**!

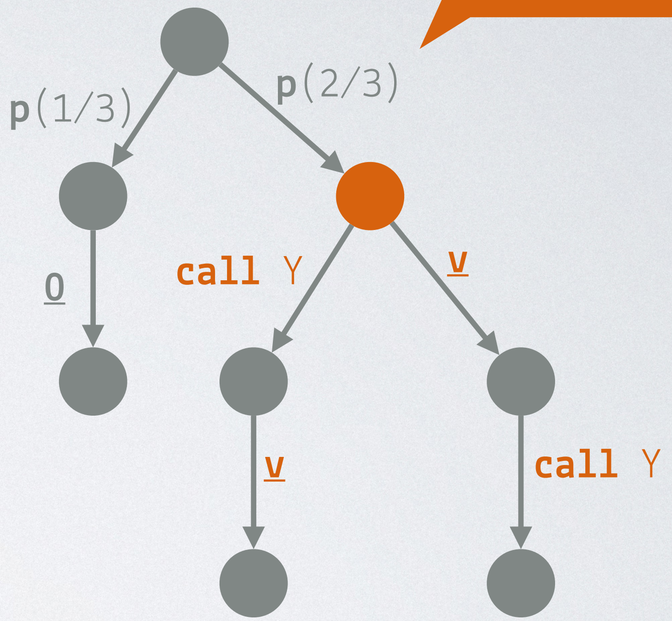
Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

Newtonian Program Analysis (NPA)

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



Differentiate



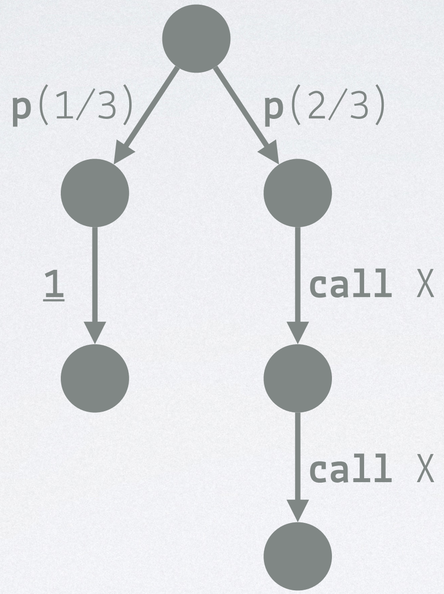
Linear!

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

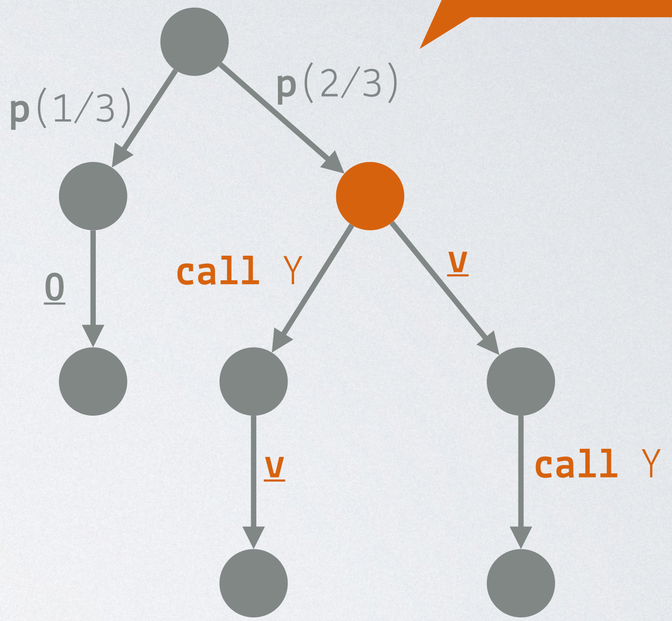
Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

Newtonian Program Analysis (NPA)

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



Differentiate



Linear!

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

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.

Our Approach: NPA for Pre-Markov Algebras

Our Approach: NPA for Pre-Markov Algebras

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

Our Approach: NPA for Pre-Markov Algebras

- Key idea: Apply Newton's method to **(pre-)Markov algebras**
- We develop a differentiation routine for **regular infinite-tree expressions**

Our Approach: NPA for Pre-Markov Algebras

- Key idea: Apply Newton's method to **(pre-)Markov algebras**
- We develop a differentiation routine for **regular infinite-tree expressions**

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

Our Approach: NPA for Pre-Markov Algebras

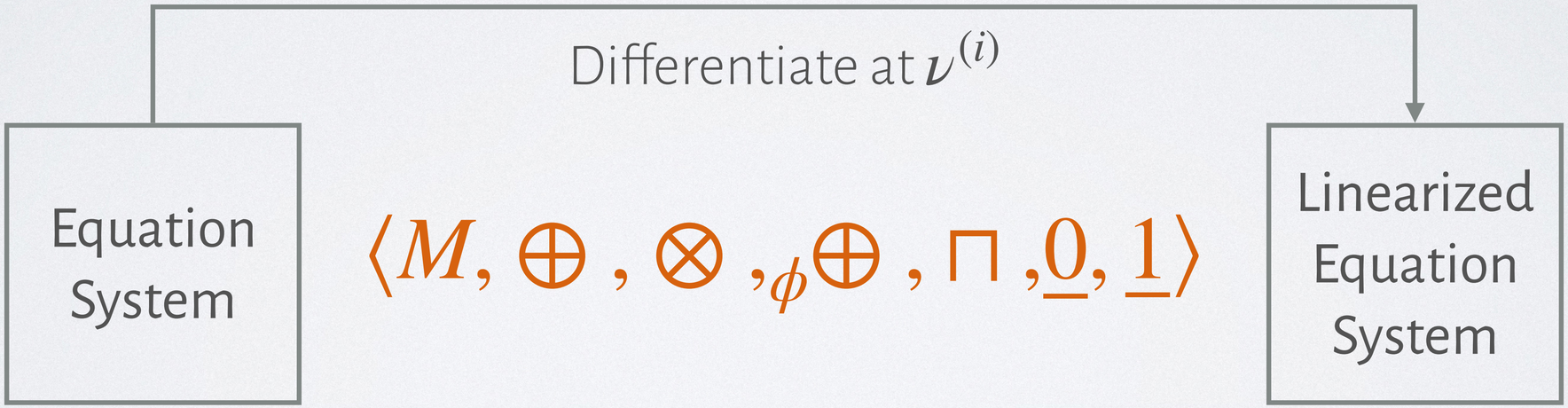
- Key idea: Apply Newton's method to **(pre-)Markov algebras**
- We develop a differentiation routine for **regular infinite-tree expressions**

Equation System

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

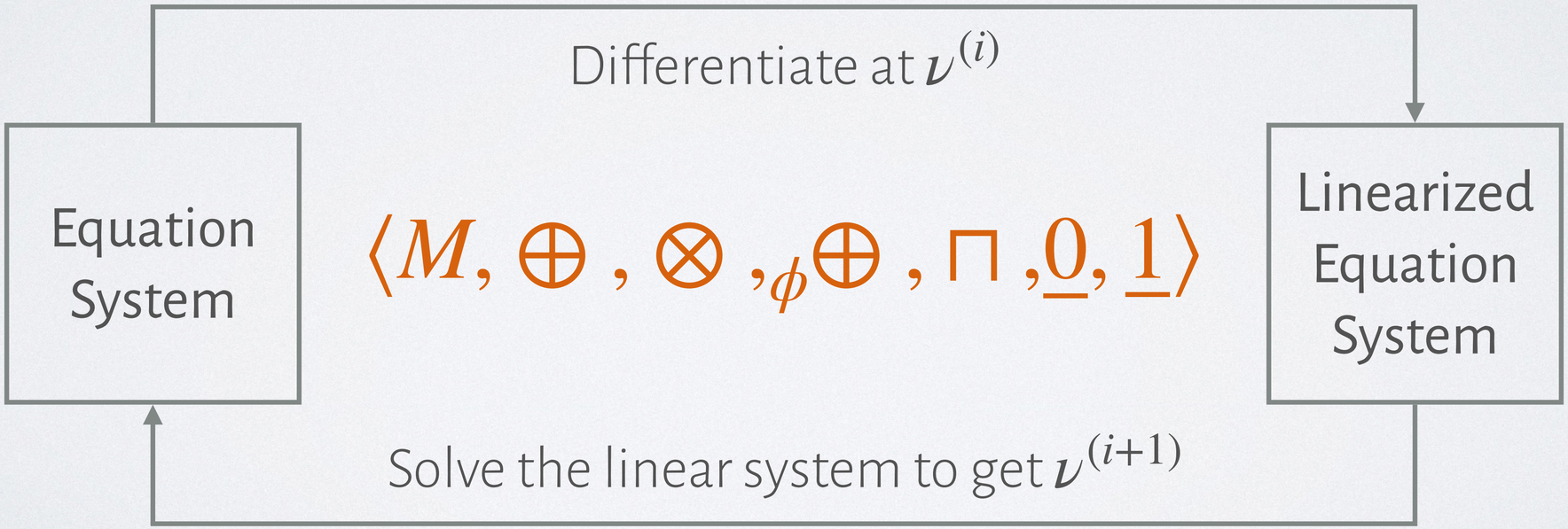
Our Approach: NPA for Pre-Markov Algebras

- Key idea: Apply Newton's method to **(pre-)Markov algebras**
- We develop a differentiation routine for **regular infinite-tree expressions**

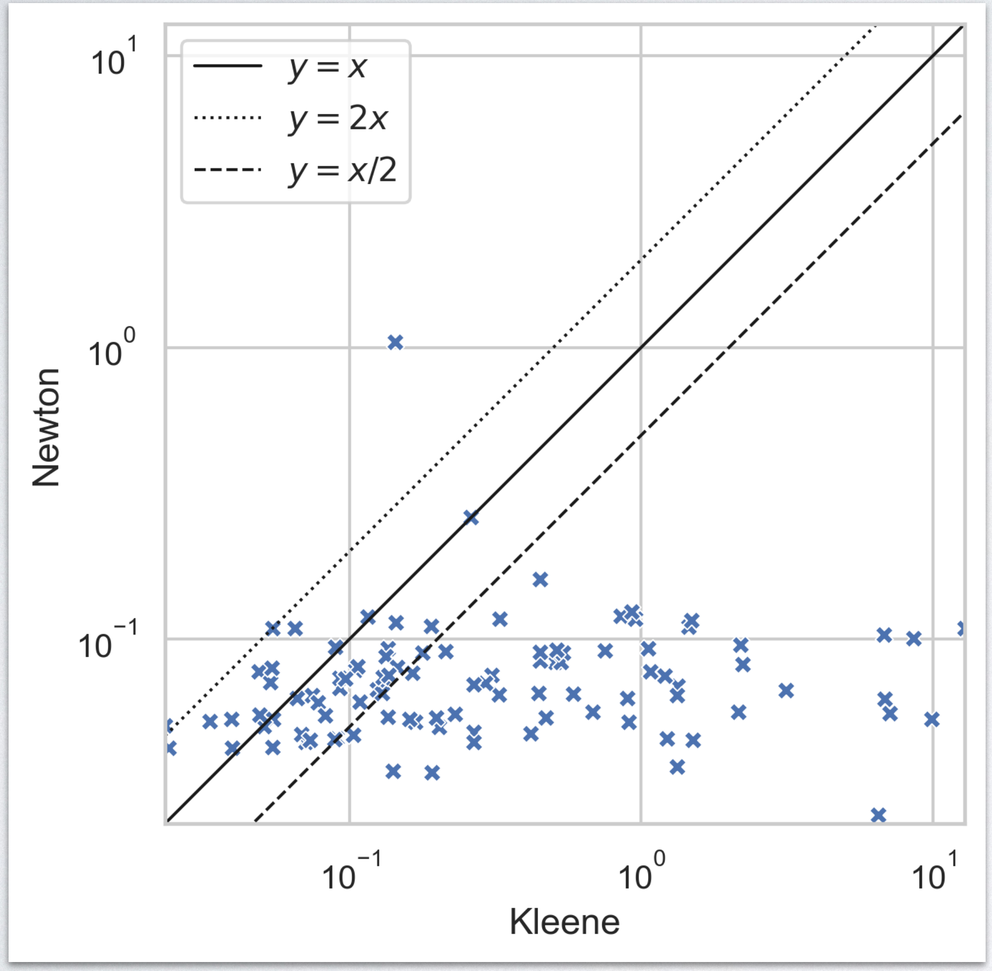


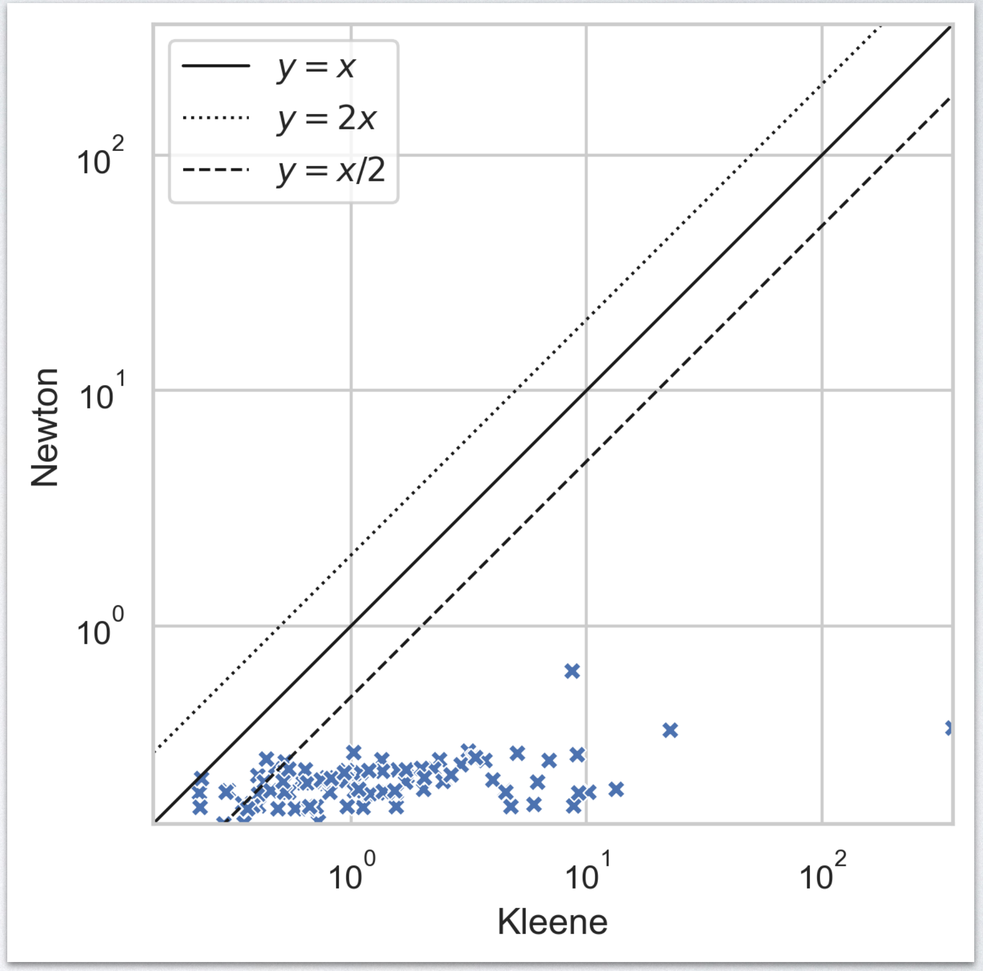
Our Approach: NPA for Pre-Markov Algebras

- Key idea: Apply Newton's method to **(pre-)Markov algebras**
- We develop a differentiation routine for **regular infinite-tree expressions**



Preliminary Evaluation





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

- ☑ **Semantics:** Markov Algebras for Multiple Kinds of Confluence
- ☑ **Representation:** Control-flow Hyper-graphs and Tree Expressions
- ☑ **Algorithm:** Newton's Method for Pre-Markov Algebras