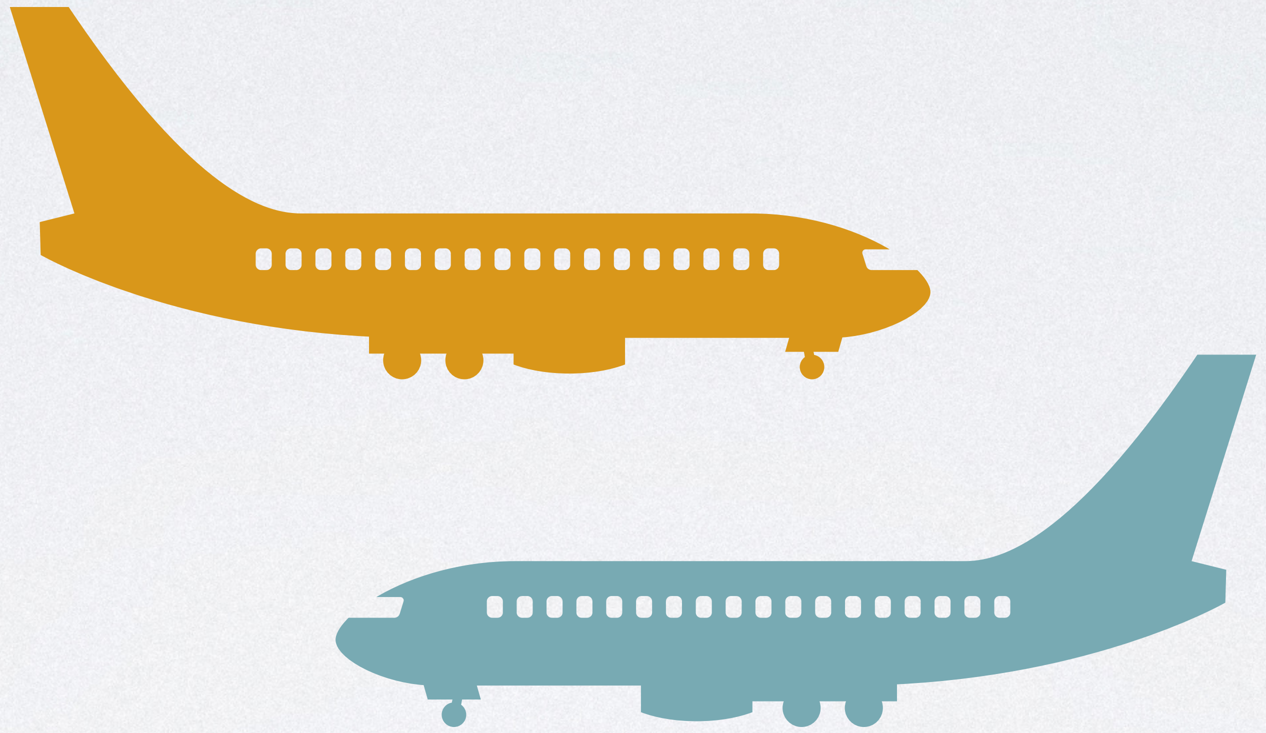
概率程序的代数程序分析

王迪
北京大学
wangdi95@pku.edu.cn
2023 年 12 月 2 日

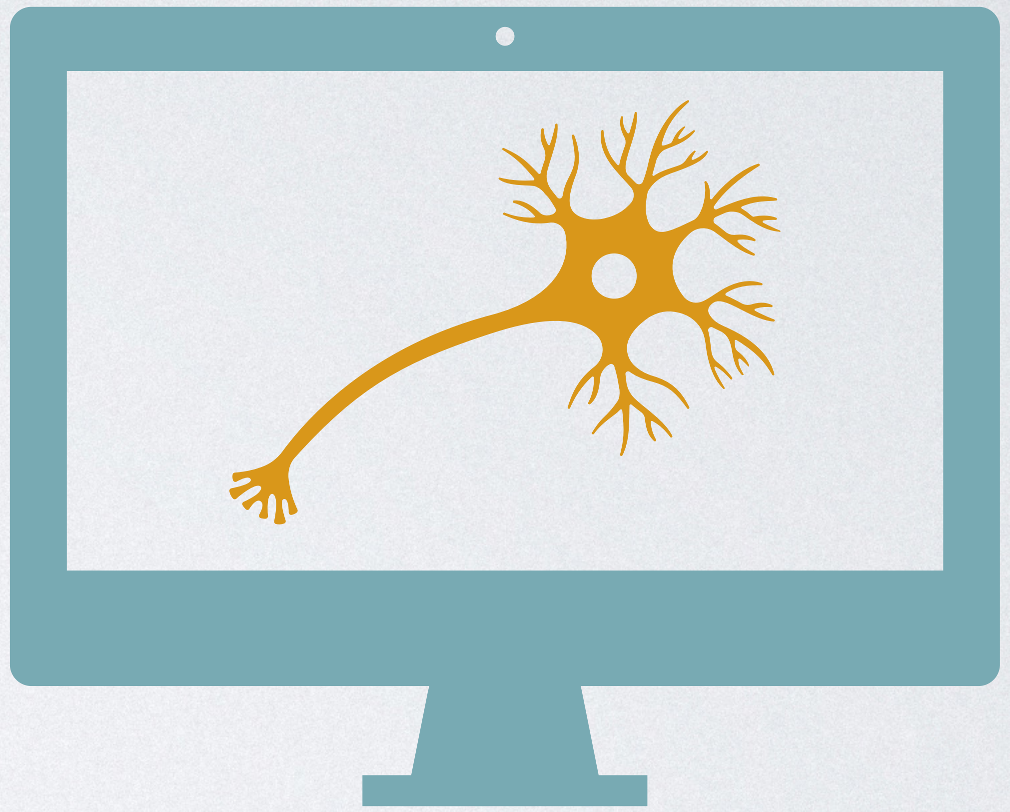
与 Jan Hoffmann 和 Thomas Reps 的合作工作

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.

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
```

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

Example: Monty Hall

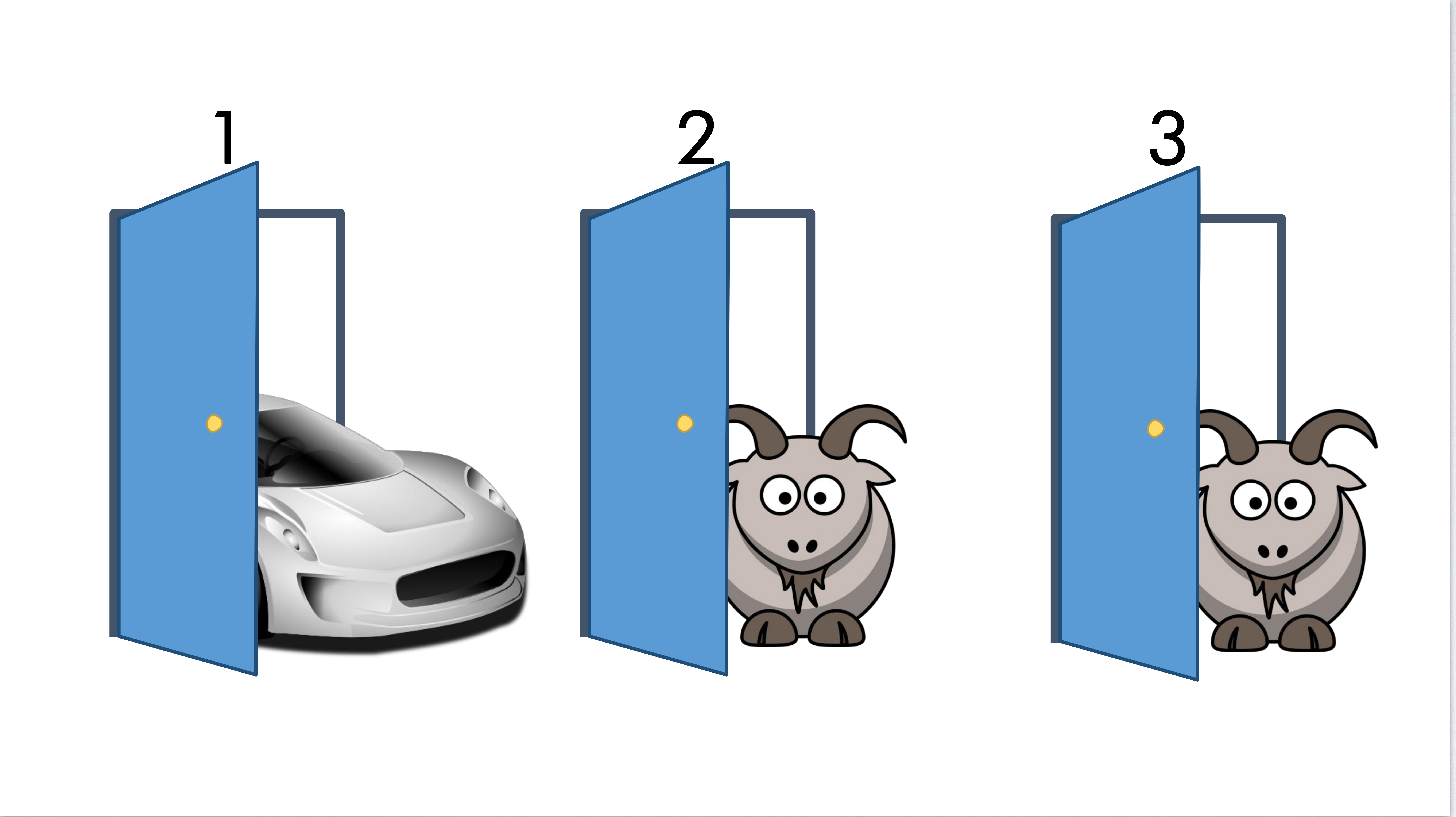


Image source: Maria Gorinova's Advances in Programming Languages (Guest Lecture) slides on Probabilistic Programming.

Example: Monty Hall

```
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
```

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

```
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

```
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

```
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;
```

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 quantitative reasoning about probabilistic programs?

How to automate such quantitative reasoning about probabilistic programs?

Examples

What is the probability that an assertion holds?

How to automate such 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 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
```

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

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

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

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
```

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

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

Do Kleene Algebras Suffice?

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

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

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

Do Kleene Algebras Suffice?

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

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

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

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**

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

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**

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

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**

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

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**

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

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**

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

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Easy to extend with more confluence operations!

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

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

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

$$a \sqcap a = a$$

...

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
```

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

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
```

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

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

Challenge II:
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
```

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

Iterative Program Analysis

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

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

- 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
```

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

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

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

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

$\cdots$

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

- 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
```

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

- 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 Program Analysis

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

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

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

Non-iterative Program Analysis

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

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

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

Equivalent to solve:

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

Analytical solution:

$$s = 2$$

No need for iteration!

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 $\mathbb{P}$[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

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

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

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

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

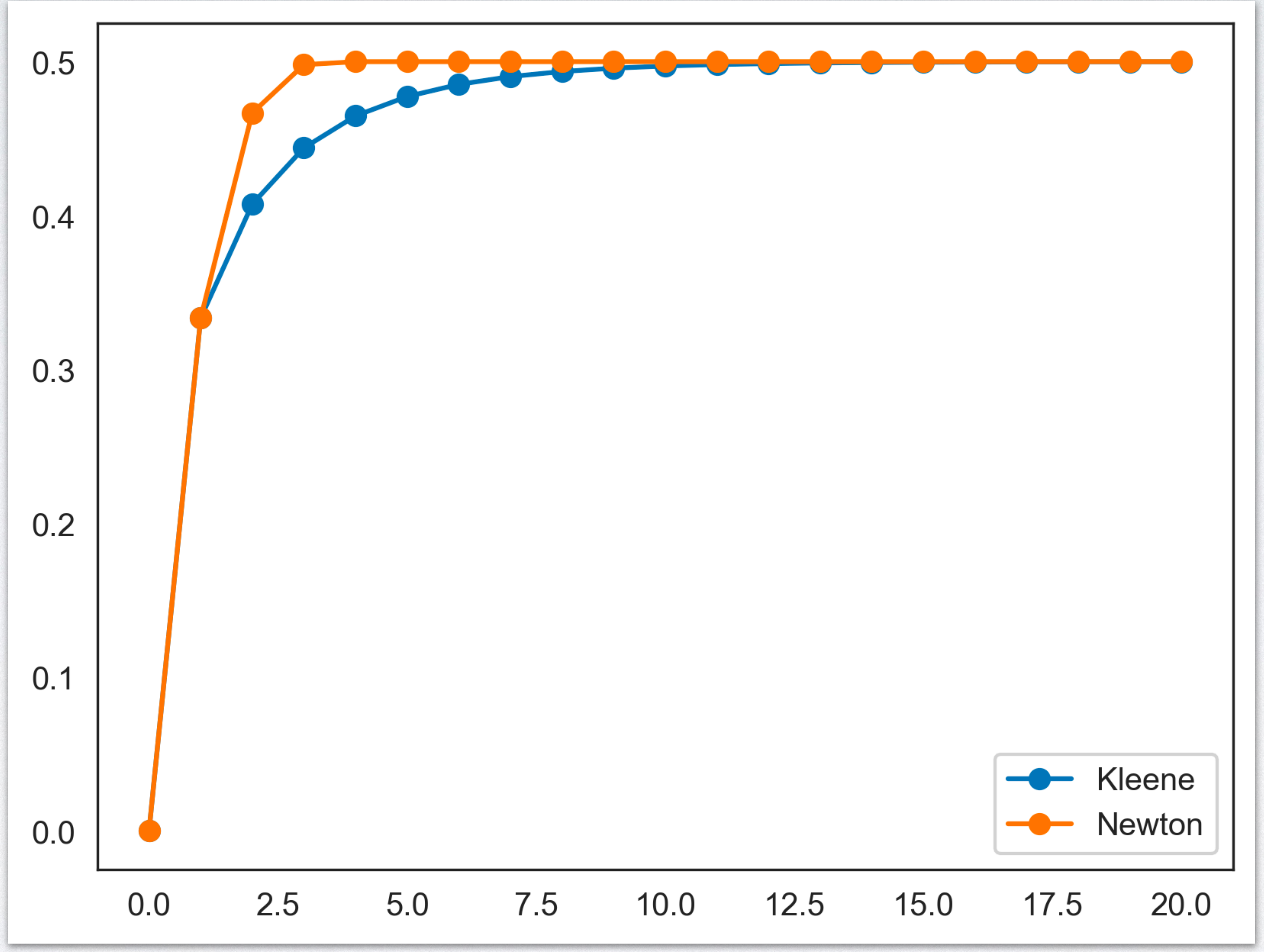
$$f(x) = 1/3 * 1 + 2/3 * (x * x)$$

Linear!

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

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

Newton's Method Converges Faster



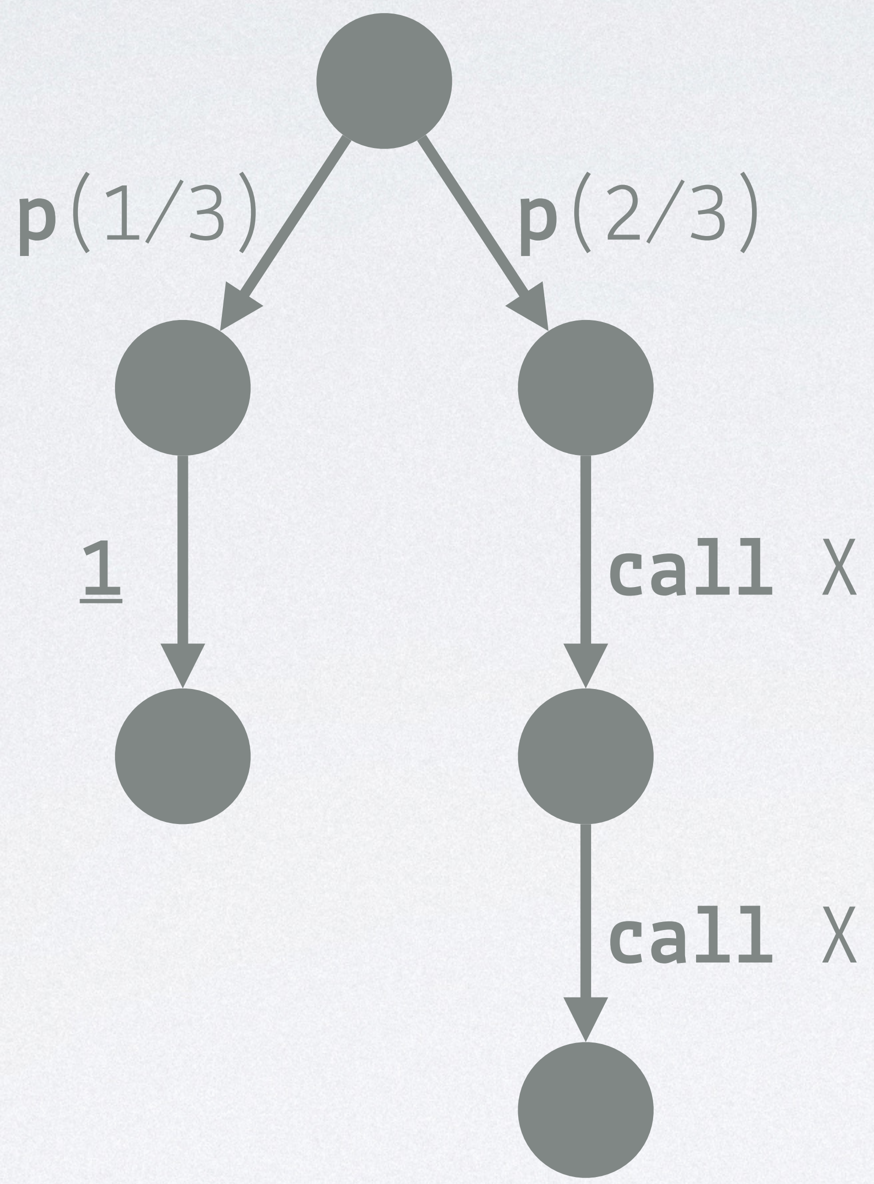
Newtonian Program Analysis (NPA)

```
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)

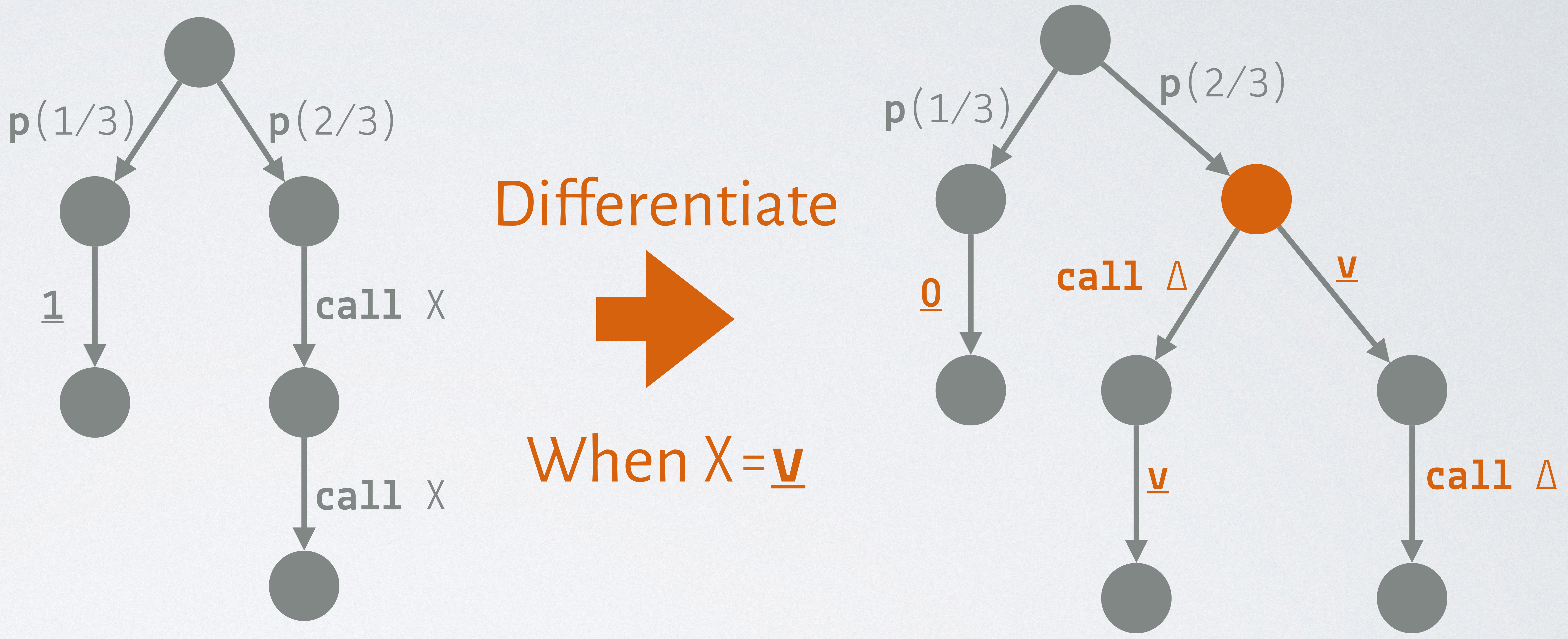
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



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)

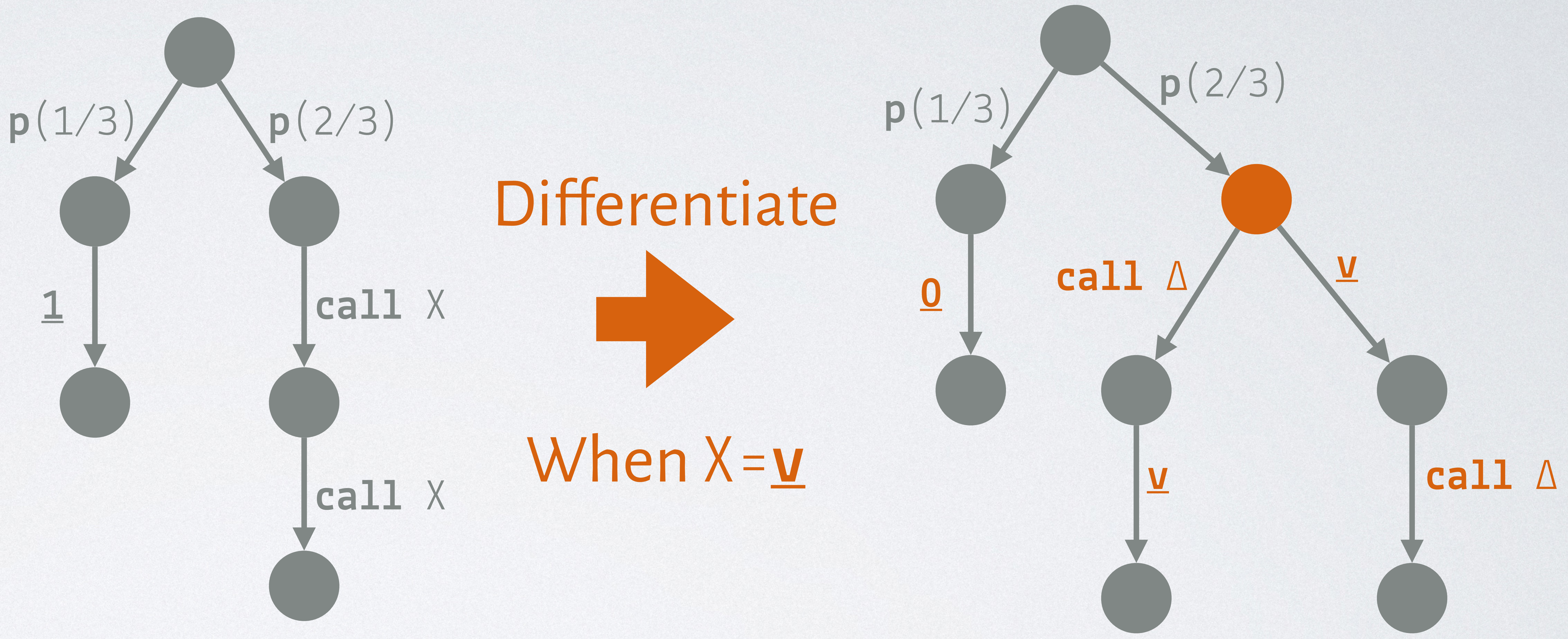
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



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
```

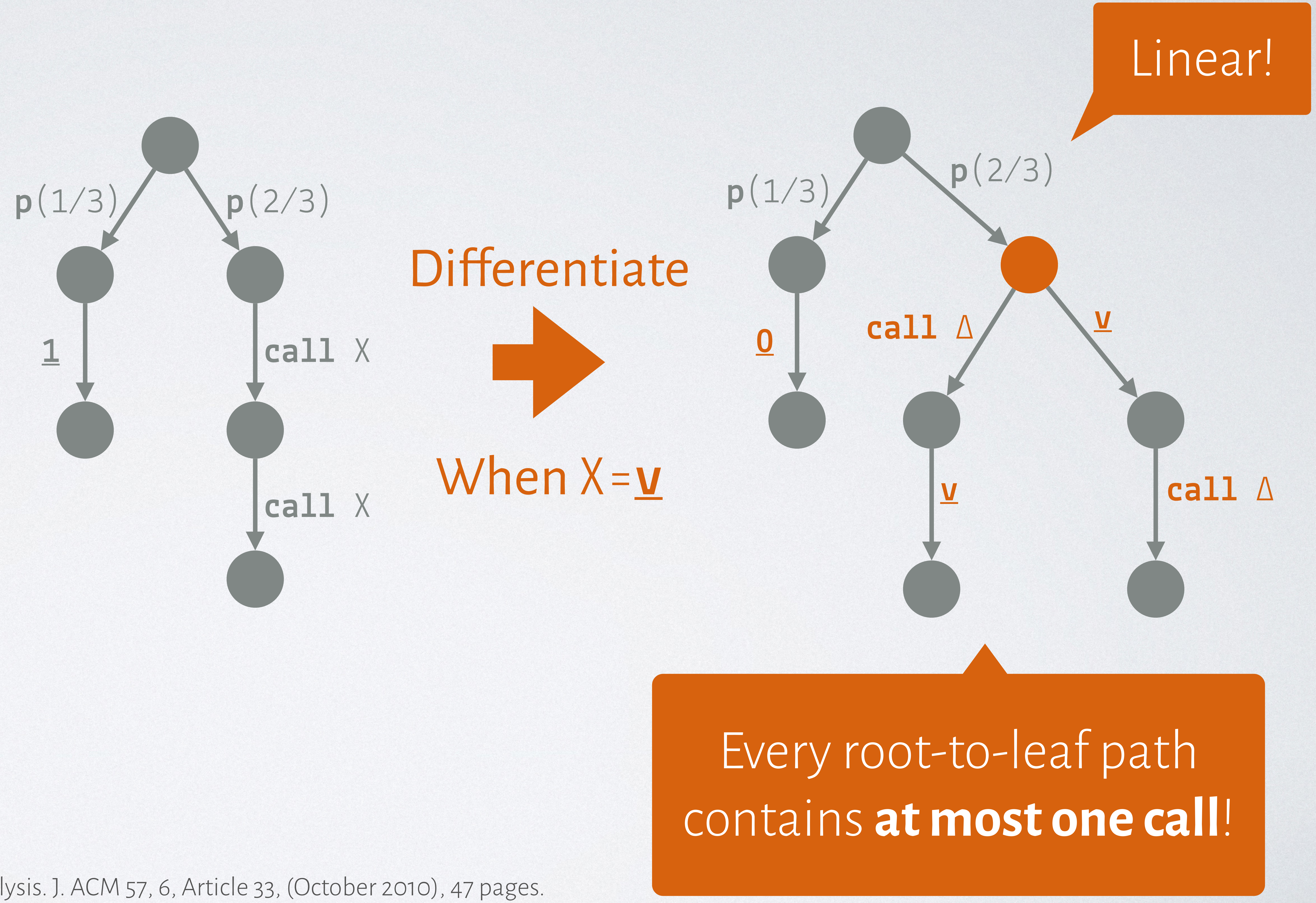


Every root-to-leaf path contains **at most one call**!

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)

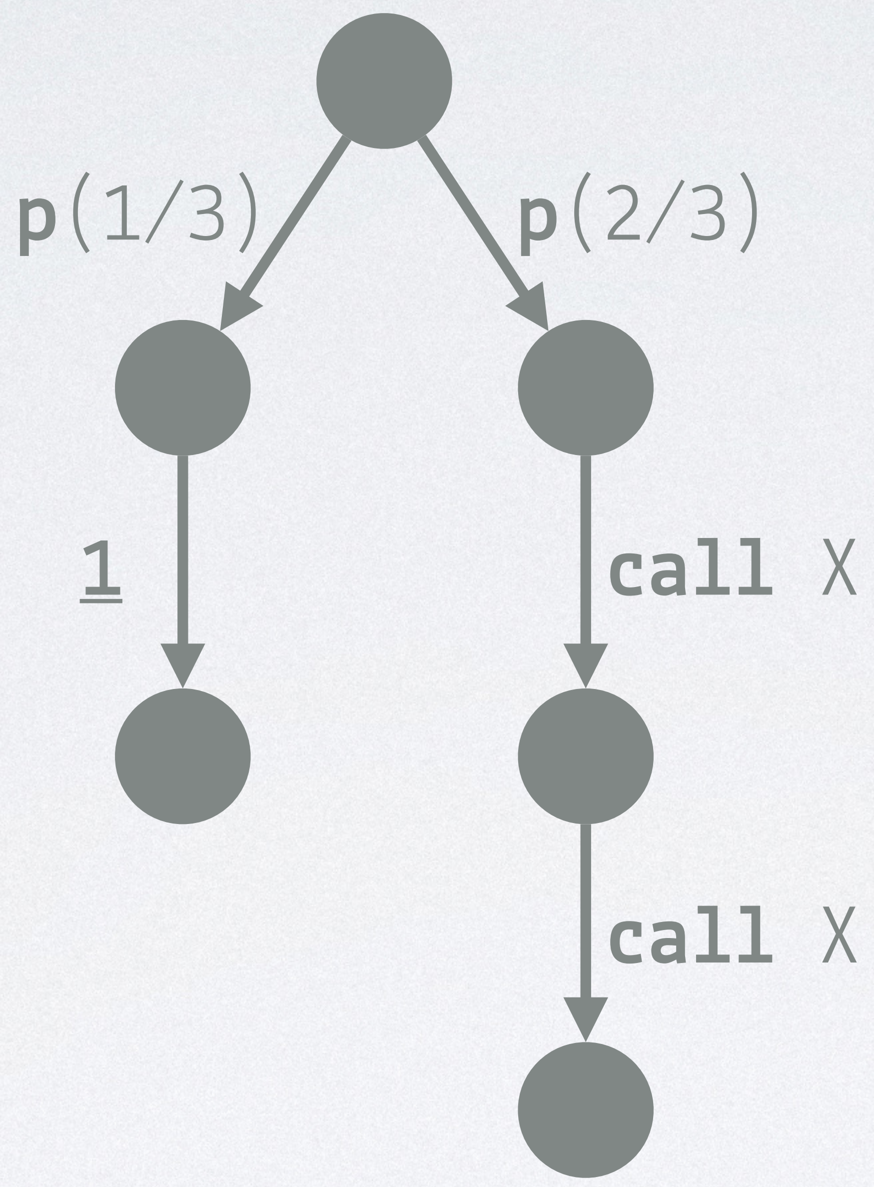
```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



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

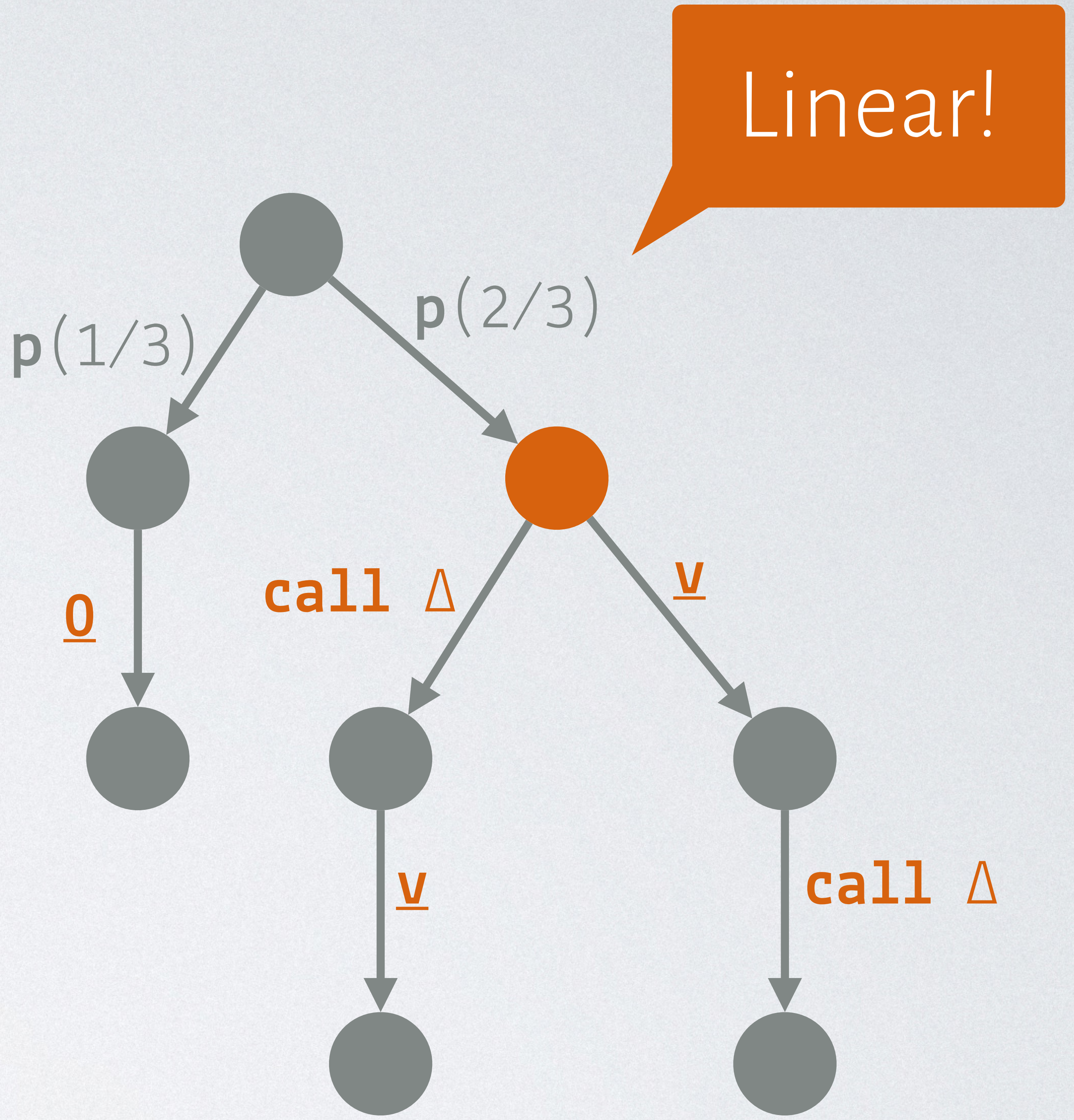
Newtonian Program Analysis (NPA)

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



Differentiate

When X = **v**



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

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

- 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**

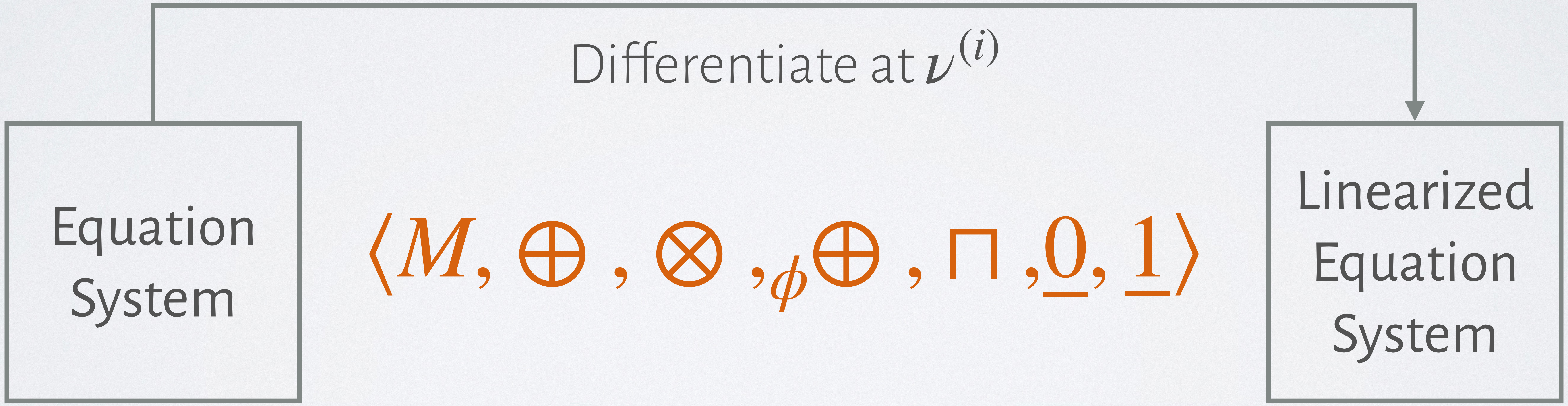
- We develop a differentiation routine for **recursive program schemes**

Equation System

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

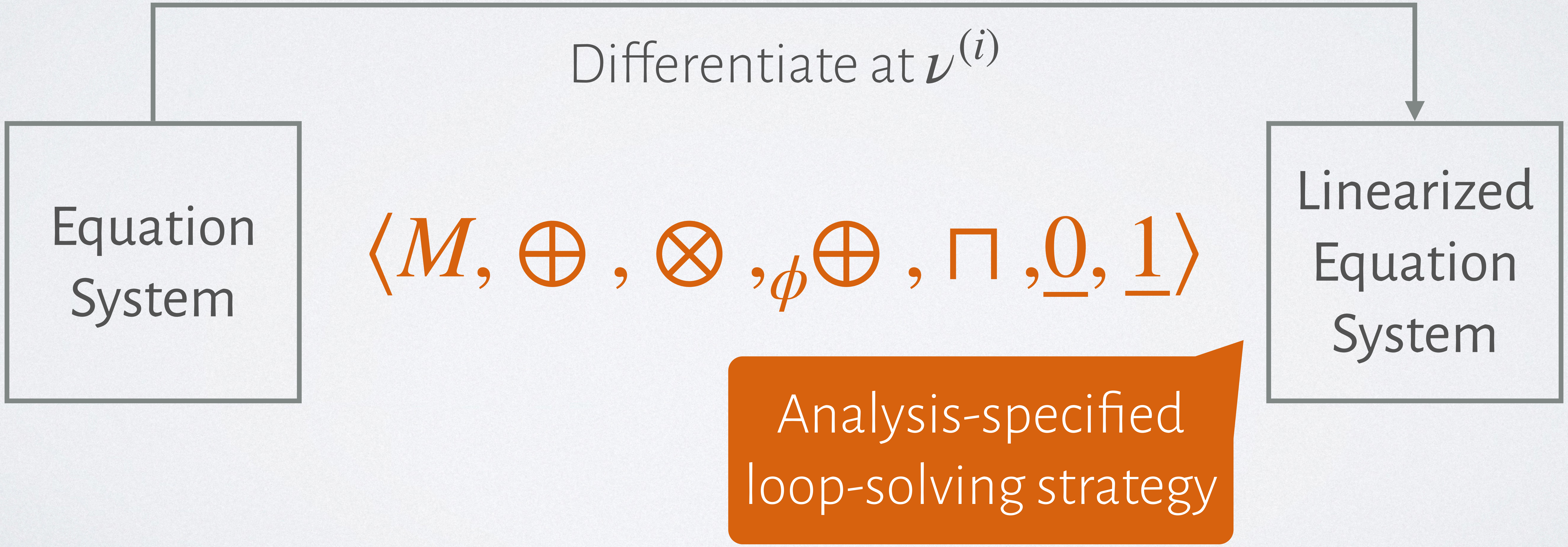
Our Approach: NPA for pre-Markov Algebras

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



Our Approach: NPA for pre-Markov Algebras

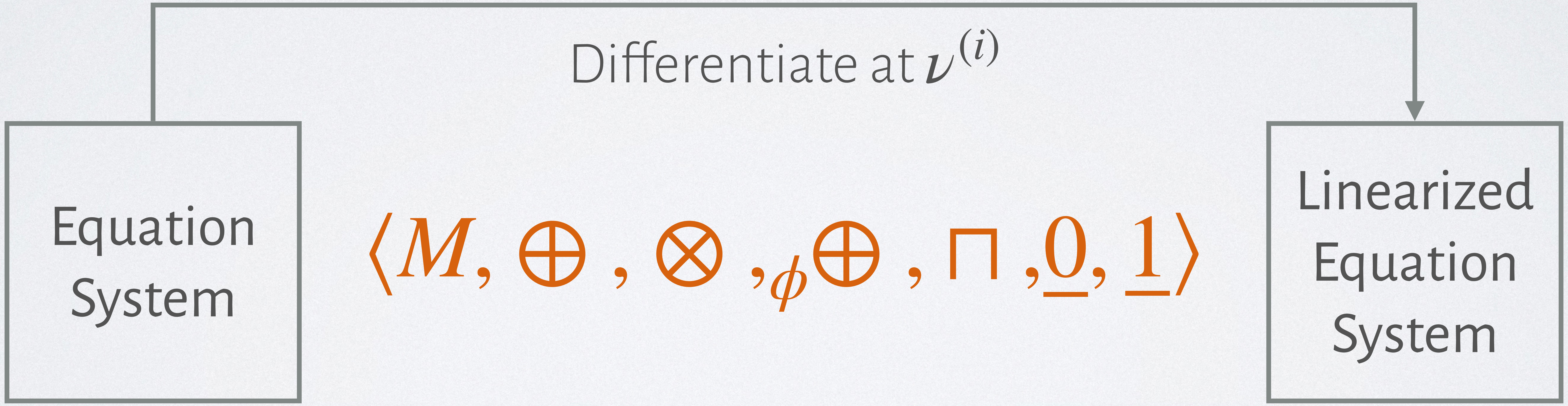
- Key idea: Apply Newton's method to **pre-Markov algebras**
- We develop a differentiation routine for **recursive program schemes**



Our Approach: NPA for pre-Markov Algebras

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

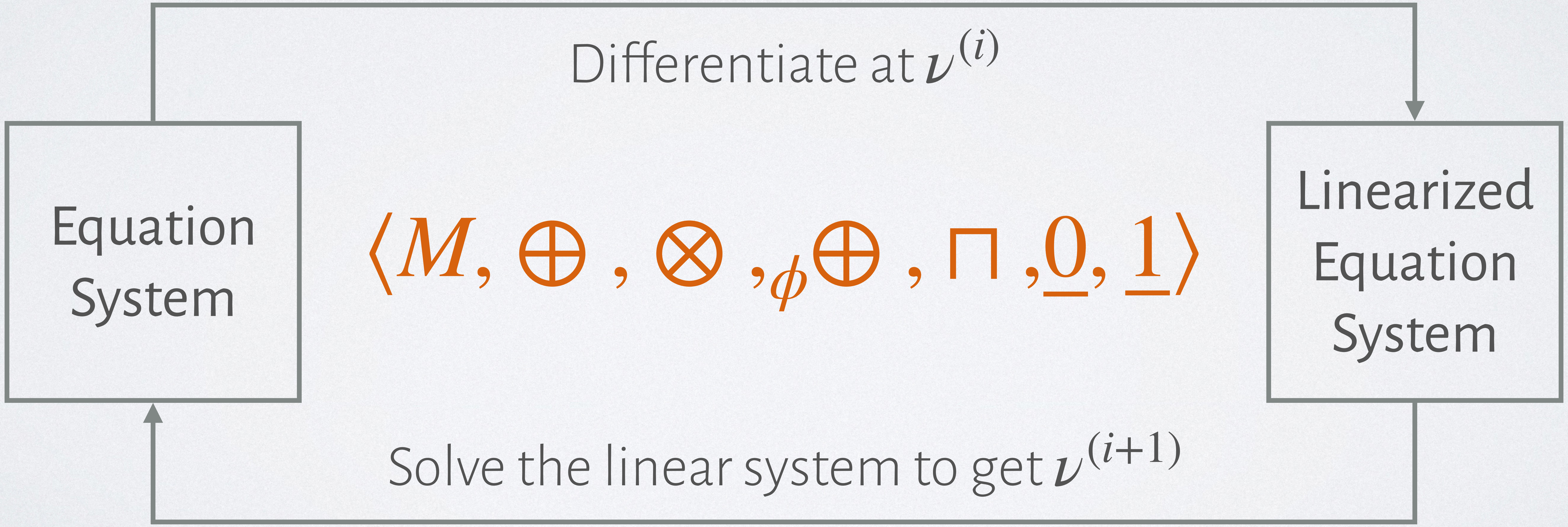
- We develop a differentiation routine for **recursive program schemes**



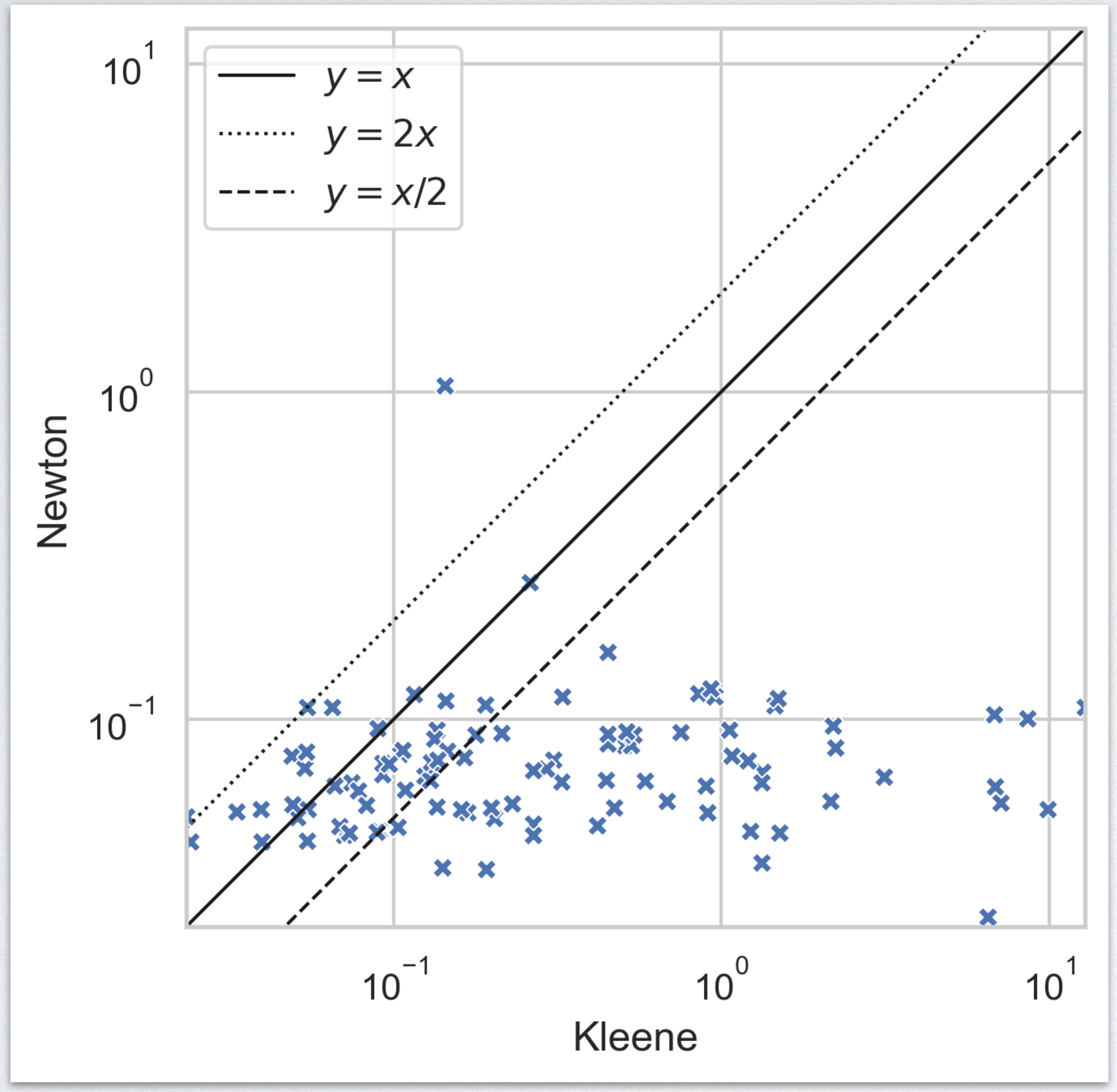
Our Approach: NPA for pre-Markov Algebras

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

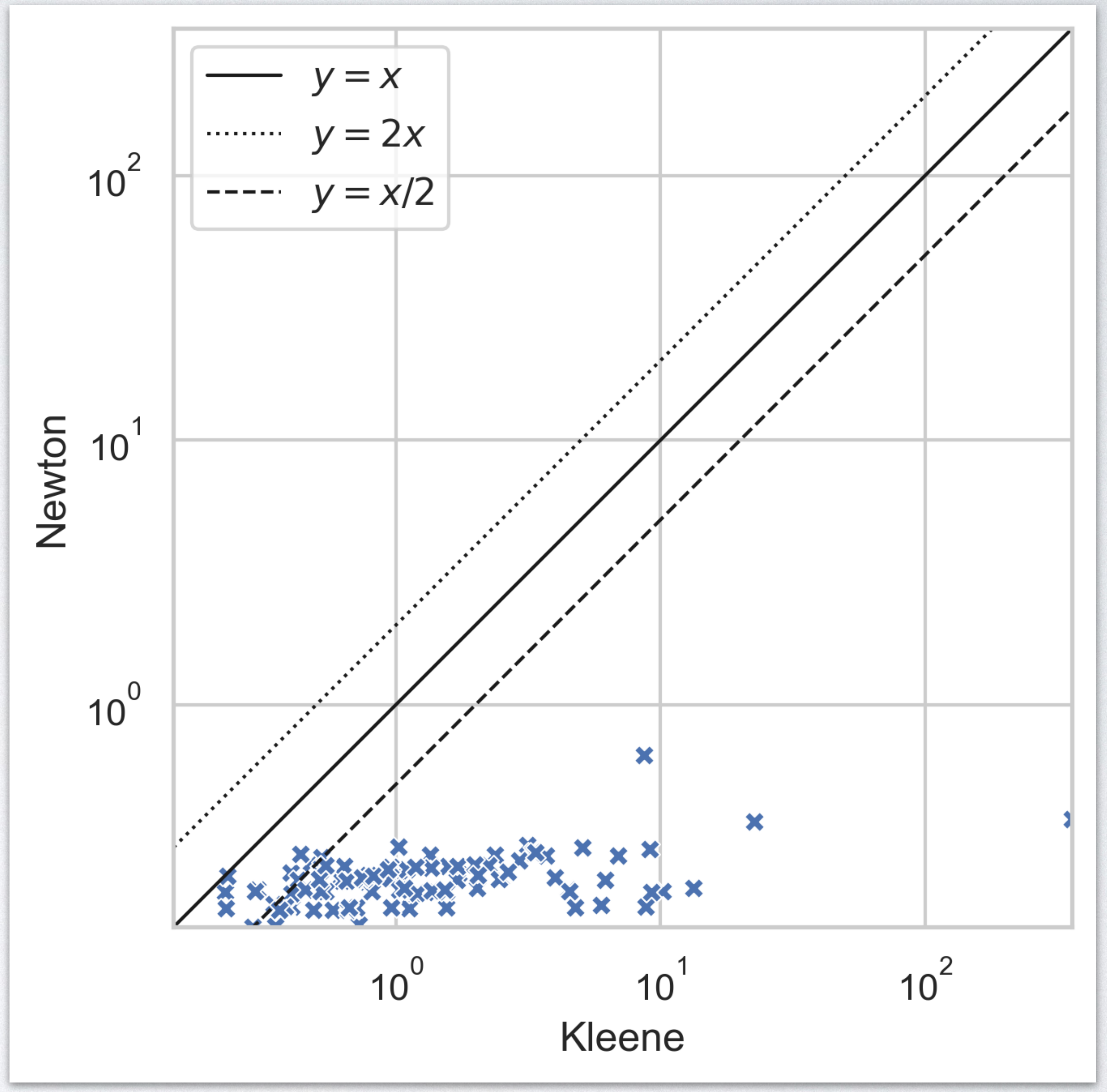
- We develop a differentiation routine for **recursive program schemes**



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 (2023). **Newtonian Program Analysis of Probabilistic Programs**. *Working Paper*.

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

- ☑ **Semantics:** Markov Algebras for Multiple Kinds of Confluences
- ☑ **Algorithm:** Newton's Method for pre-Markov Algebras

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

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