Central Moment Analysis For Cost Accumulators In Probabilistic Programs

Di Wang[1], Jan Hoffmann[1], Thomas Reps[2]

[1] Carnegie Mellon University
[2] University of Wisconsin-Madison

Probabilistic Programs

Probabilistic Programs

Standard programs +

Probabilistic Programs

Standard programs + Probability distributions

Probabilistic Programs

Standard programs + Probability distributions

Random control flows

PROBABILISTIC PROGRAMS

Standard programs + Probability distributions / Random control flows

Can be used to implement and analyze

- Randomized algorithms
- Cryptographic protocols
- Machine-learning algorithms

Cost Accumulators

Cost Accumulators

Quantities that can only be incremented or decremented

Cost Accumulators

Quantities that can only be incremented or decremented

Termination time

Cost Accumulators

Quantities that can only be incremented or decremented

Termination time

Rewards in MDPs

Cost Accumulators

Quantities that can only be incremented or decremented

Termination time

Rewards in MDPs

Cash flow

Cost Accumulators

Quantities that can only be incremented or decremented

Termination time	Rewards in MDPs	Cash flow

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2); # sample t from a uniform distribution
    x := x - t;
    call rdwalk();
    tick(1)              # add one to the cost accumulator
  fi
end
```

Quantitative Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

QUANTITATIVE ANALYSIS

What can we know about the **accumulated cost**?

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

Quantitative Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

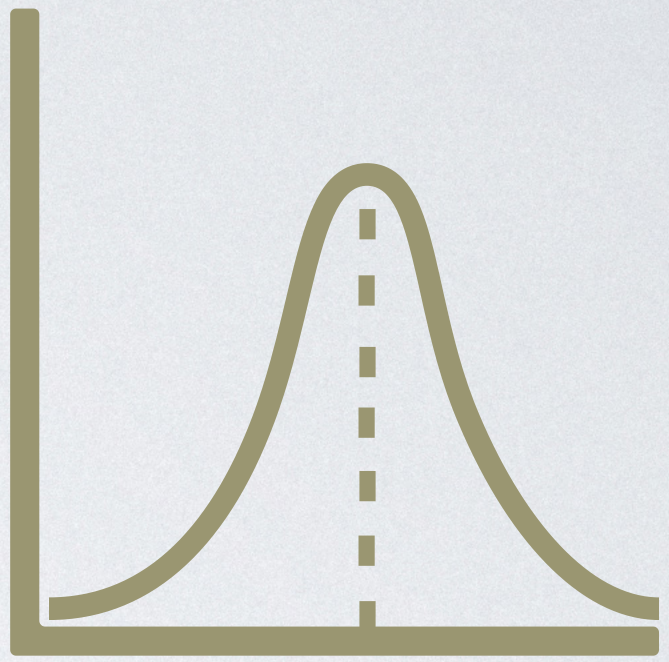
QUANTITATIVE ANALYSIS

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

By **simulation**, we can obtain an empirical estimation of this distribution.



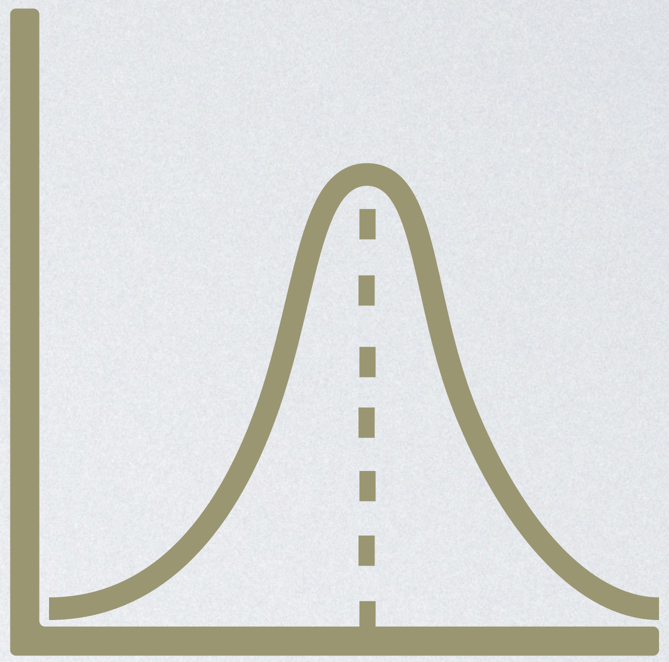
QUANTITATIVE ANALYSIS

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

By **simulation**, we can obtain an empirical estimation of this distribution.



How to **rigorously** reasoning about this distribution?

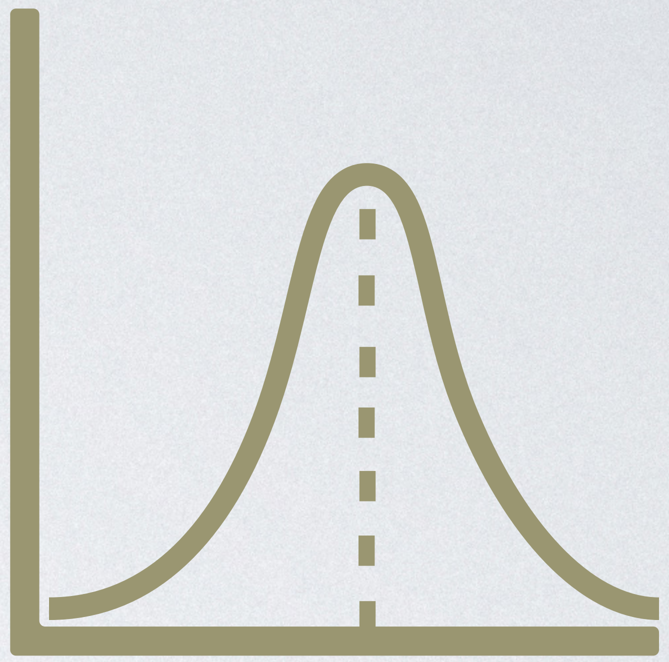
QUANTITATIVE ANALYSIS

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

By **simulation**, we can obtain an empirical estimation of this distribution.



How to **rigorously** reasoning about this distribution?

Usually it is **intractable** to analyze the result distribution precisely.

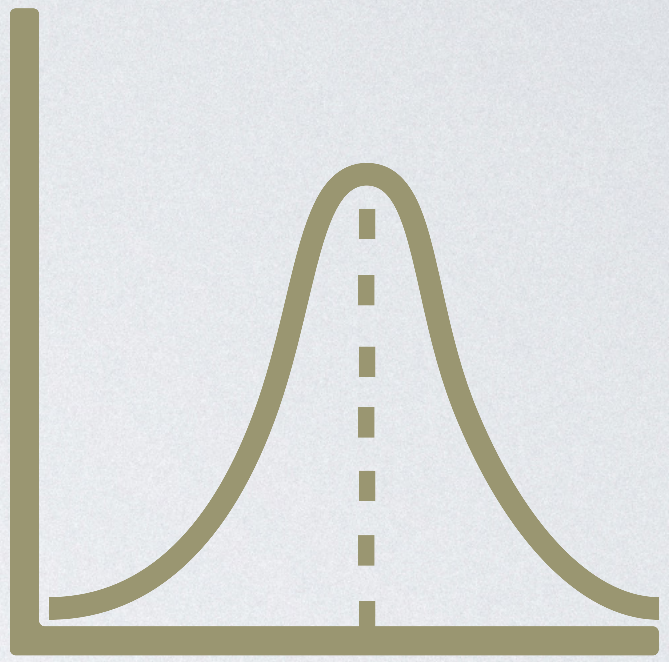
QUANTITATIVE ANALYSIS

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

By **simulation**, we can obtain an empirical estimation of this distribution.



How to **rigorously** reasoning about this distribution?

Usually it is **intractable** to analyze the result distribution precisely.

Static analysis can leverage **quantitative aggregate information** of the distribution.

QUANTITATIVE ANALYSIS

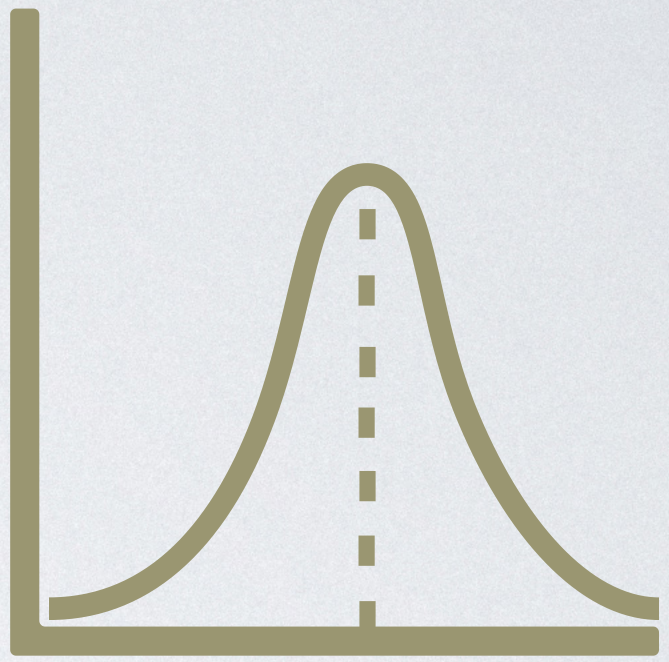
```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

Such as Moments!

What can we know about the **accumulated cost**?

The program produces a **distribution** on possible accumulated costs.

By **simulation**, we can obtain an empirical estimation of this distribution.



How to **rigorously** reasoning about this distribution?

Usually it is **intractable** to analyze the result distribution precisely.

Static analysis can leverage **quantitative aggregate information** of the distribution.

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

- Let T denote the cost accumulator

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$\mathbb{E}[T] \leq 2x + 4$

$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$\mathbb{E}[T] \leq 2x + 4$

$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$
  - Central **moments**: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \geq 2$

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$$\mathbb{E}[T] \leq 2x + 4$$

$$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$$

$$\mathbb{E}[(T - \mathbb{E}[T])^2] \leq 22x + 28$$

- Let T denote the cost accumulator
  - Raw moments: $\mathbb{E}[T^m]$ for $m \geq 1$
  - Central moments: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \geq 2$

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$$\mathbb{E}[T] \leq 2x + 4$$

$$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$$

$$\mathbb{E}[(T - \mathbb{E}[T])^2] \leq 22x + 28$$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$
  - Central **moments**: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \geq 2$
- Moments can indicate **the shape of a distribution**

Central Moment Analysis

```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$$\mathbb{E}[T] \leq 2x + 4$$

$$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$$

$$\mathbb{E}[(T - \mathbb{E}[T])^2] \leq 22x + 28$$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$
  - Central **moments**: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \geq 2$
- Moments can indicate **the shape of a distribution**
  - We can use moments to derive **tail bounds**, e.g., $\mathbb{P}[T \geq 4x]$

Central Moment Analysis

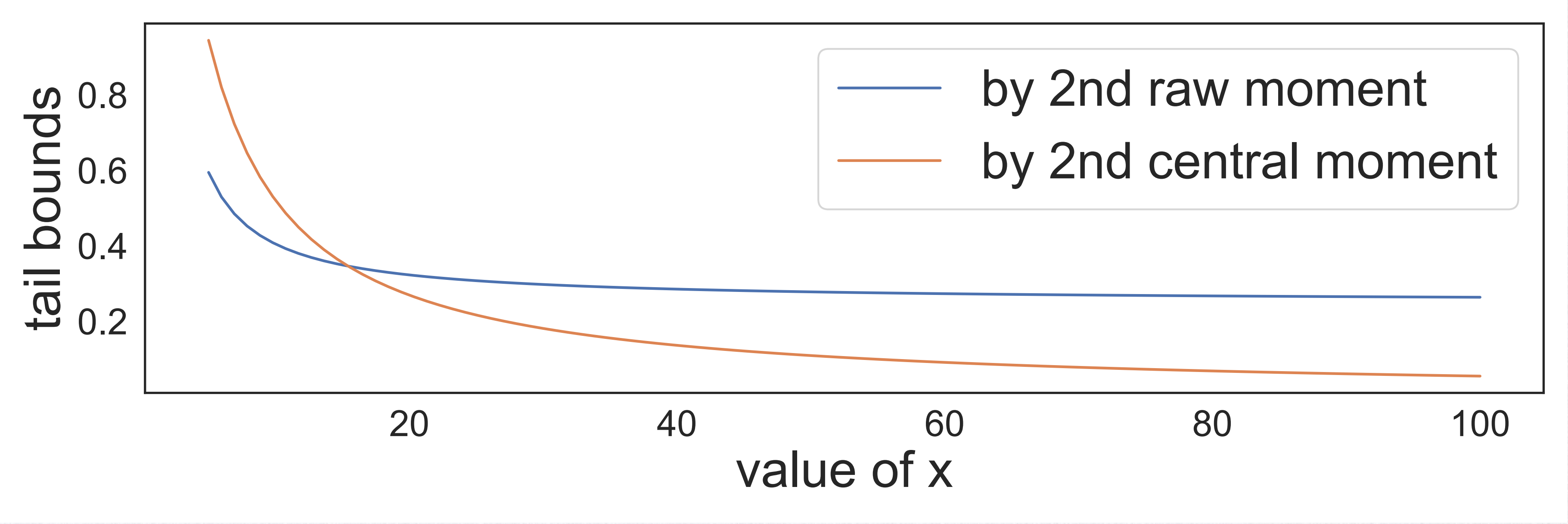
```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$\mathbb{E}[T] \leq 2x + 4$

$\mathbb{E}[T^2] \leq 4x^2 + 22x + 28$

$\mathbb{E}[(T - \mathbb{E}[T])^2] \leq 22x + 28$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \geq 1$
  - Central **moments**: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \geq 2$
- Moments can indicate **the shape of a distribution**
  - We can use moments to derive **tail bounds**, e.g., $\mathbb{P}[T \geq 4x]$



Lower is better!

Central Moment Analysis

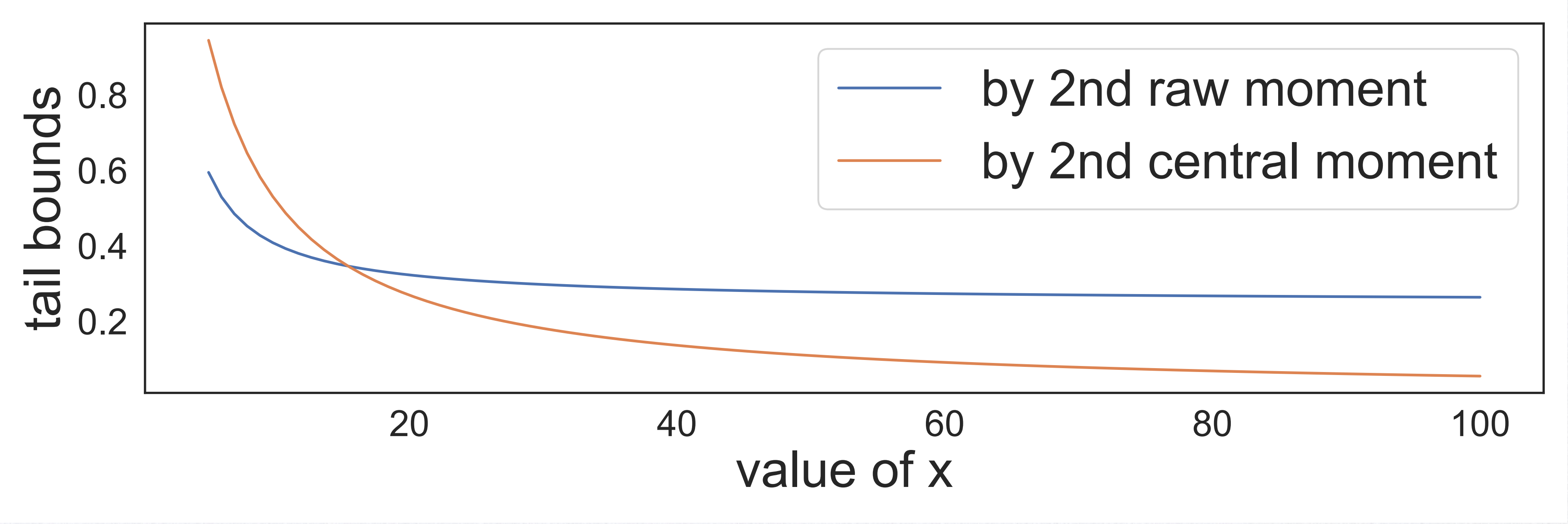
```
# variables: x, t
# pre-condition: x > 0
func rdwalk() begin
  if x > 0 then
    t ~ uniform(-1, 2);
    x := x - t;
    call rdwalk();
    tick(1)
  fi
end
```

$$\mathbb{E}[T] \le 2x + 4$$

$$\mathbb{E}[T^2] \le 4x^2 + 22x + 28$$

$$\mathbb{E}[(T - \mathbb{E}[T])^2] \le 22x + 28$$

- Let T denote the cost accumulator
  - Raw **moments**: $\mathbb{E}[T^m]$ for $m \ge 1$
  - Central **moments**: $\mathbb{E}[(T - \mathbb{E}[T])^m]$ for $m \ge 2$
- Moments can indicate **the shape of a distribution**
  - We can use moments to derive **tail bounds**, e.g., $\mathbb{P}[T \ge 4x]$



Lower is better!

Central moments can provide more information!

Our Work

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

OUR WORK

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

OUR WORK

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis

Our Work

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming

OUR WORK

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming
- Optional stopping theorem for reasoning about soundness

OUR WORK

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming
- Optional stopping theorem for reasoning about soundness

Our contributions

Our Work

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming
- Optional stopping theorem for reasoning about soundness

Our contributions

- An algebraic and systematic approach for composing moments

Our Work

We present the first fully automatic analysis
for deriving symbolic interval bounds
on central moments for cost accumulators
in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming
- Optional stopping theorem for reasoning about soundness

Our contributions

- An algebraic and systematic approach for composing moments
- Moment-polymorphic recursion that handles non-trivial recursion patterns

Our Work

We present the first fully automatic analysis for deriving symbolic interval bounds on central moments for cost accumulators in probabilistic programs with general recursion

Ideas from the literature

- Potential method for cost analysis
- Automation via linear programming
- Optional stopping theorem for reasoning about soundness

Our contributions

- An algebraic and systematic approach for composing moments
- Moment-polymorphic recursion that handles non-trivial recursion patterns
- A program logic for moment inference, and proof of its soundness

Case Study: Timing Attack

password checker

CASE STUDY: TIMING ATTACK

password checker

- It checks the N-bit user input against the secret bit-by-bit

Case Study: Timing Attack

password checker

- It checks the N-bit user input against the secret bit-by-bit
- Its **running time** might reveal the length of the **common prefix** of the input and the secret

CASE STUDY: TIMING ATTACK

password checker

- It checks the N-bit user input against the secret bit-by-bit
- Its **running time** might reveal the length of the **common prefix** of the input and the secret
- It adds some **random delays** through its computation as a defense

Case Study: Timing Attack

password checker

- It checks the N-bit user input against the secret bit-by-bit
- Its **running time** might reveal the length of the **common prefix** of the input and the secret
- It adds some **random delays** through its computation as a defense

Our tool still points out that a timing attack is very likely to succeed!

Case Study: Timing Attack

Case Study: Timing Attack

$T_1 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Case Study: Timing Attack

$T_1 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Runtime Characteristics (Obtained via Our Tool)

$$13N - 5K \leq \mathbb{E}[T_0] \leq 13N - 3K \qquad\qquad 13N \leq \mathbb{E}[T_1] \leq 15N$$

Case Study: Timing Attack

$T_1 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Runtime Characteristics (Obtained via Our Tool)

$13N - 5K \leq \mathbb{E}[T_0] \leq 13N - 3K$ $\qquad$ $13N \leq \mathbb{E}[T_1] \leq 15N$

So if the runtime is less than $13N - 1.5K$, the next bit is likely to be zero

Case Study: Timing Attack

$T_1 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \stackrel{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Runtime Characteristics (Obtained via Our Tool)

$$13N - 5K \leq \mathbb{E}[T_0] \leq 13N - 3K \qquad 13N \leq \mathbb{E}[T_1] \leq 15N$$

So if the runtime is less than $13N - 1.5K$, the next bit is likely to be zero

$$\mathbb{E}[(T_0 - \mathbb{E}[T_0])^2] \leq 8N - 36K^2 + 52NK + 24K \qquad \mathbb{E}[(T_1 - \mathbb{E}[T_1])^2] \leq 26N^2 + 42N$$

CASE STUDY: TIMING ATTACK

$T_1 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Runtime Characteristics (Obtained via Our Tool)

$$13N - 5K \leq \mathbb{E}[T_0] \leq 13N - 3K \qquad 13N \leq \mathbb{E}[T_1] \leq 15N$$

So if the runtime is less than $13N - 1.5K$, the next bit is likely to be zero

$$\mathbb{E}[(T_0 - \mathbb{E}[T_0])^2] \leq 8N - 36K^2 + 52NK + 24K \qquad \mathbb{E}[(T_1 - \mathbb{E}[T_1])^2] \leq 26N^2 + 42N$$

By concentration inequalities and central moments, we can derive tail bounds on $\mathbb{P}[T_1 \leq 13N - 1.5K]$ and $\mathbb{P}[T_0 \geq 13N - 1.5K]$

Case Study: Timing Attack

$T_1 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is one

$T_0 \overset{\text{def}}{=}$ the runtime if we have obtained the first K bits and the next bit is zero

Runtime Characteristics (Obtained via Our Tool)

$$13N - 5K \leq \mathbb{E}[T_0] \leq 13N - 3K \qquad 13N \leq \mathbb{E}[T_1] \leq 15N$$

So if the runtime is less than $13N - 1.5K$, the next bit is likely to be zero

$$\mathbb{E}[(T_0 - \mathbb{E}[T_0])^2] \leq 8N - 36K^2 + 52NK + 24K \qquad \mathbb{E}[(T_1 - \mathbb{E}[T_1])^2] \leq 26N^2 + 42N$$

By concentration inequalities and central moments, we can derive tail bounds on $\mathbb{P}[T_1 \leq 13N - 1.5K]$ and $\mathbb{P}[T_0 \geq 13N - 1.5K]$

$\mathbb{P}[\text{The attack succeeds}] \geq 0.830561$ when $N = 32$

More in the Paper

- Full formalism of the program logic for moment inference
- How our system supports inter-procedural reasoning
- Proof of the soundness for the program logic
- An implementation and experiments on analysis capability and scalability
- Comparison with prior work on raw-moment analysis for termination time and expected-cost bound analysis