KoAT: Automatic Complexity and Termination Analysis of Integer Programs

Nils Lommen, Éléanore Meyer, and Jürgen Giesl

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [ x  ]   [x − 1]
  [ y1 ] ← [  x  ]
  [ y2 ]   [  1  ]

end
```

- Does this program terminate over $\mathbb{Z}$?

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [ x  ]   [x − 1]
  [y1 ] ← [  x  ]
  [y2 ]   [  1  ]
end
```

- Does this program terminate over $\mathbb{Z}$?
- What is the runtime complexity?

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [ x  ]   [ x − 1 ]
  [ y1 ] ← [   x   ]
  [ y2 ]   [   1   ]

end
```

- Does this program terminate over $\mathbb{Z}$?
- What is the runtime complexity?
  - Outer loop: $\mathcal{O}(x)$

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [x ]   [x − 1]
  [y1] ← [ x  ]
  [y2]   [ 1  ]
  while (y1 > y2 ∧ y1 > 0) do
    [y1] ← [2 · y1]
    [y2]   [3 · y2]
  end
end
```

- Does this program terminate over $\mathbb{Z}$?
- What is the runtime complexity?
  - Outer loop: $\mathcal{O}(x)$
  - Inner loop: $\mathcal{O}(x \cdot \log(x))$

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [x ]   [x − 1]
  [y1] ← [ x  ]
  [y2]   [ 1  ]
  while (y1 > y2 ∧ y1 > 0) do
    [y1] ← [2 · y1]
    [y2]   [3 · y2]
  end
end
```

- Does this program terminate over $\mathbb{Z}$?
- What is the runtime complexity?
  - Outer loop: $\mathcal{O}(x)$
  - Inner loop: $\mathcal{O}(x \cdot \log(x))$
- Solution: Use KoAT!

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [x ]   [x − 1]
  [y1] ← [ x  ]
  [y2]   [ 1  ]
  while (y1 > y2 ∧ y1 > 0) do
    [y1]   [2 · y1]
    [y2] ← [3 · y2]
  end
end
```

- Does this program terminate over $\mathbb{Z}$?
- What is the runtime complexity?
  - Outer loop: $\mathcal{O}(x)$
  - Inner loop: $\mathcal{O}(x \cdot \log(x))$
- Solution: Use KoAT!
  - open-source complexity and termination analysis tool

Motivation

Goal: Infer (upper) runtime bounds and prove termination for integer programs

```
while (x > 0) do
  [x]   [x - 1]
  [y1] ← [ x  ]
  [y2]   [ 1  ]
  while (y1 > y2 ∧ y1 > 0) do
    [y1]   [2 · y1]
    [y2] ← [3 · y2]
  end
end
```

Bounds: (Ω(1), O(log(n)*n))

Took less than 1 Second

This number is the total time the Docker container was running. The overhead is about 1 second in most cases.

Show result in a new tab

Preprocessing

Found invariant $1+2\cdot X_2 \leq 3\cdot X_1 \wedge 1 \leq X_2$ for location l2

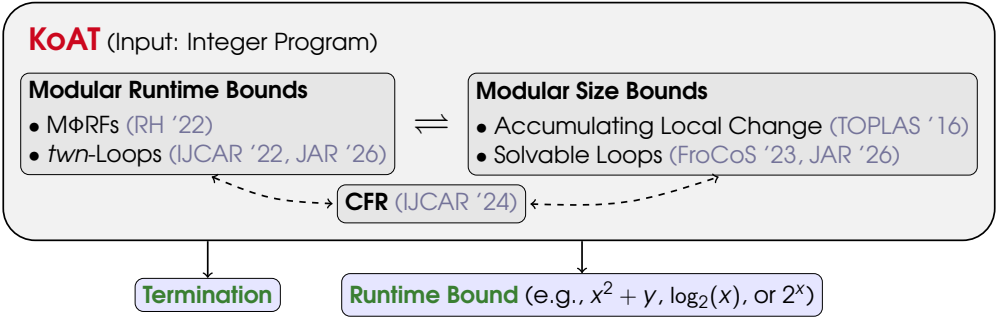
Problem after Preprocessing

Start: l0
Program_Vars: X_0, X_1, X_2
Temp_Vars:
Locations: l0, l1, l2
Return Locations:
Transitions:
t_0: $l0(X_0, X_1, X_2) \to l1(X_0, X_1, X_2)$
t_1: $l1(X_0, X_1, X_2) \to l2(X_0\text{-}1, X_0, 1)$:|: $1 \leq X_0$
t_3: $l2(X_0, X_1, X_2) \to l1(X_0\text{-}1, X_0, 1)$:|: $1+2\cdot X_2 \leq 3\cdot X_1 \wedge 1 \leq X_2$
t_2: $l2(X_0, X_1, X_2) \to l2(X_0, 2\cdot X_1, 3\cdot X_2)$:|: $1 \leq X_1 \wedge 1+X_2 \leq X_1 \wedge 1+2\cdot X_2 \leq 3\cdot X_1 \wedge 1 \leq X_2$

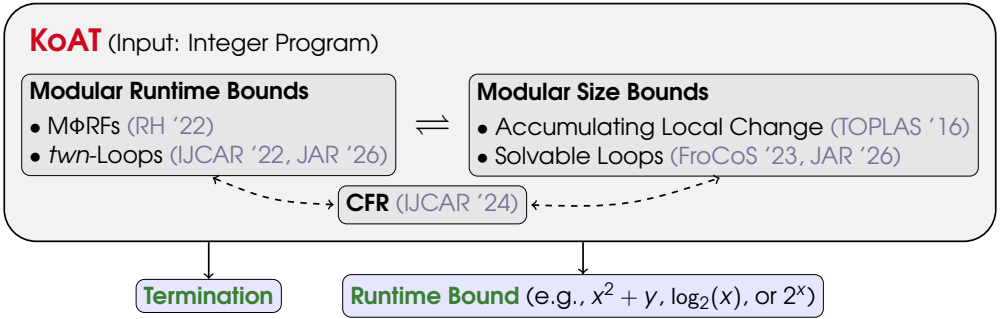
Show Graph | Show Dependency Graph

MPRF for transition t_1: $l1(X_0, X_1, X_2) \to l2(X_0\text{-}1, X_0, 1)$:|: $1 \leq X_0$ of depth 1:

Overview: KoAT

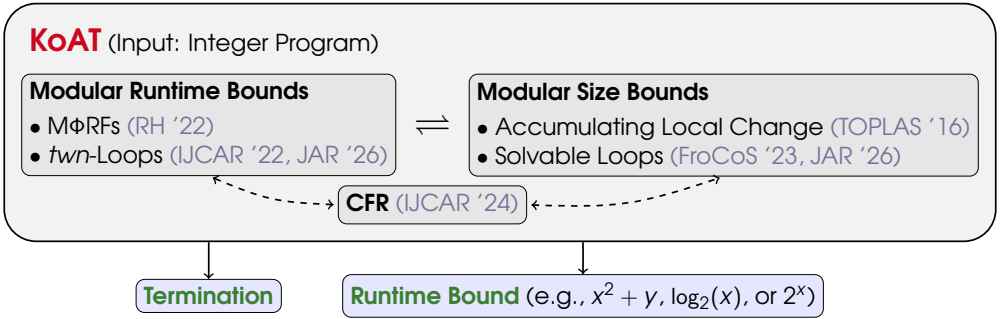


Overview: KoAT



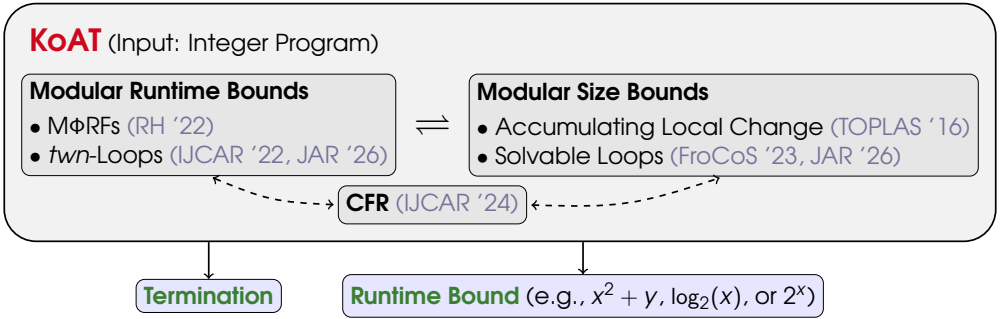
- **Alternating Modular Inference**: Analyze subprograms in topological order.

Overview: KoAT



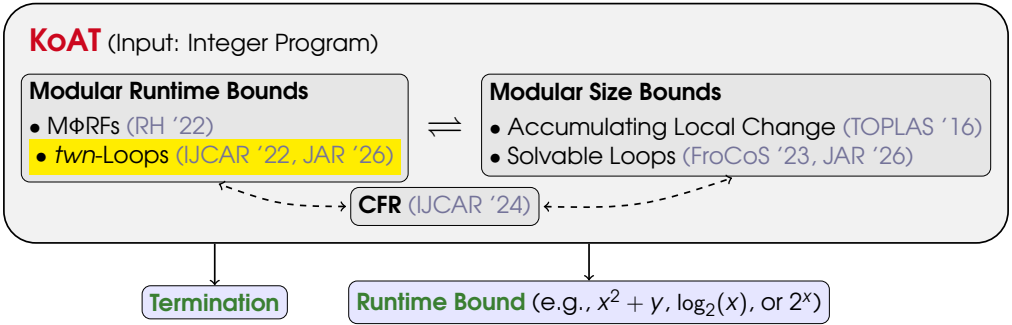
- **Alternating Modular Inference**: Analyze subprograms in topological order.
- **Control-Flow Refinement (CFR)**:

Overview: KoAT



- **Alternating Modular Inference**: Analyze subprograms in topological order.
- **Control-Flow Refinement (CFR)**:
  - Fallback technique to unroll/split loops and sort out paths.

Overview: KoAT



- **Alternating Modular Inference**: Analyze subprograms in topological order.
- **Control-Flow Refinement (CFR)**:
  - Fallback technique to unroll/split loops and sort out paths.

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate?

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate? **Yes!**

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate? **Yes!**
- y_2 eventually *outgrows* the value of y_1.

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate? **Yes!**
- y_2 eventually *outgrows* the value of y_1.

- Reduce termination to an existential formula over $\mathbb{Z}$.

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate? **Yes!**
- y_2 eventually *outgrows* the value of y_1.

- Reduce termination to an existential formula over $\mathbb{Z}$.
- For terminating twn-loops always polynomial runtime bounds.

Analyzing Triangular Weakly Non-Linear Loops

```
while (y1 > y2 ∧ y2 > 0) do
    [y1]      [2 · y1]
    [y2]  ←   [3 · y2]
end
```

- Does the loop terminate? **Yes!**
- y_2 eventually *outgrows* the value of y_1.

- Reduce termination to an existential formula over $\mathbb{Z}$.
- For terminating twn-loops always polynomial runtime bounds.
- KoAT infers the local runtime bound: $\log_{3/2}(y_1) + 1$

Lifting Bounds via Modularity

How do we get the global runtime of the nested program?

```
while (x > 0) do
  [x, y1, y2] ← [x − 1, x, 1]
  while (y1 > y2 ∧ y1 > 0) do
    [y1, y2] ← [2 · y1, 3 · y2]
  end
end
```

- Inner loop has local bound $\log_{3/2}(y_1) + 1$. How often is it executed in total?

$$\log_{3/2}(y_1) + 1$$

Lifting Bounds via Modularity

How do we get the global runtime of the nested program?

```
while (x > 0) do
  [x, y1, y2] ← [x − 1, x, 1]
  while (y1 > y2 ∧ y1 > 0) do
    [y1, y2] ← [2 · y1, 3 · y2]
  end
end
```

- Inner loop has local bound $\log_{3/2}(y_1) + 1$. How often is it executed in total?
- **Step 1 (Outer Runtime):** Outer loop executes at most x times by MΦRFs

$$x \cdot (\log_{3/2}(y_1) + 1)$$

Lifting Bounds via Modularity

How do we get the global runtime of the nested program?

```
while (x > 0) do
  [x, y1, y2] ← [x − 1, x, 1]
  while (y1 > y2 ∧ y1 > 0) do
    [y1, y2] ← [2 · y1, 3 · y2]
  end
end
```

- Inner loop has local bound $\log_{3/2}(y_1) + 1$. How often is it executed in total?
- **Step 1 (Outer Runtime):** Outer loop executes at most x times by MΦRFs
- **Step 2 (Size Bounds):** What is the size of y_1 before entering the inner loop?
  - Update is $y_1 \leftarrow x$.
  - Size bound for y_1 upon entry is x.

$$x \cdot (\log_{3/2}(x) + 1)$$

Lifting Bounds via Modularity

How do we get the global runtime of the nested program?

```
while (x > 0) do
  [x, y1, y2] ← [x − 1, x, 1]
  while (y1 > y2 ∧ y1 > 0) do
    [y1, y2] ← [2 · y1, 3 · y2]
  end
end
```

- Inner loop has local bound $\log_{3/2}(y_1) + 1$. How often is it executed in total?
- **Step 1 (Outer Runtime):** Outer loop executes at most x times by MΦRFs
- **Step 2 (Size Bounds):** What is the size of y_1 before entering the inner loop?
  - Update is $y_1 \leftarrow x$.
  - Size bound for y_1 upon entry is x.
- **Step 3 (Composition):** Substitute size bound into local runtime bound and multiply by outer runtime:

$$\mathcal{RB}_{\text{global}} \leq x \cdot (\log_{3/2}(x) + 1)$$

Live Demo

```
(GOAL COMPLEXITY)
(STARTTERM (FUNCTIONSYMBOLS l0))
(VAR x1 y1 y2)
(RULES
  l0(x1,y1,y2) -> l1(x1,y1,y2)
  l1(x1,y1,y2) -> l2(x1-1,x1,1) :|: x1 > 0
  l2(x1,y1,y2) -> l2(x1,2*y1,3*y2) :|: y1 > y2 && y1 > 0
  l2(x1,y1,y2) -> l1(x1-1,x1,1)
)
```

https://koat.verify.rwth-aachen.de

Evaluation

Complexity Analysis (Complexity_ITS, 838 Benchmarks, < 647 Terminating)

	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$< \omega$	$\mathrm{AVG}^{+}(\mathrm{s})$
KoAT	132	9	261	113	548	2.81
KoAT-2016	132	0	216	104	475	0.67
CoFloCo	125	0	230	93	457	1.50

Evaluation

Complexity Analysis (Complexity_ITS, 838 Benchmarks, < 647 Terminating)

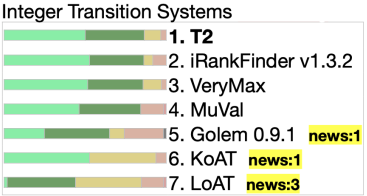
	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$< \omega$	$\mathrm{AVG}^{+}(\mathrm{s})$	Succ. Rate
KoAT	132	9	261	113	548	2.81	84 %
KoAT-2016	132	0	216	104	475	0.67	73 %
CoFloCo	125	0	230	93	457	1.50	70 %

Evaluation

Complexity Analysis (Complexity_ITS, 838 Benchmarks, < 647 Terminating)

	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$< \omega$	$\mathrm{AVG}^{+}(\mathrm{s})$	Succ. Rate
KoAT	132	9	261	113	548	2.81	84 %
KoAT-2016	132	0	216	104	475	0.67	73 %
CoFloCo	125	0	230	93	457	1.50	70 %

Termination Analysis (Termination_ITS, 1222 Benchmarks)



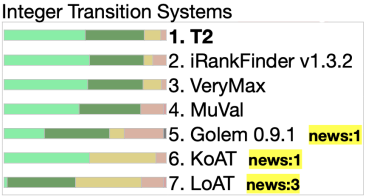
Evaluation

Complexity Analysis (Complexity_ITS, 838 Benchmarks, < 647 Terminating)

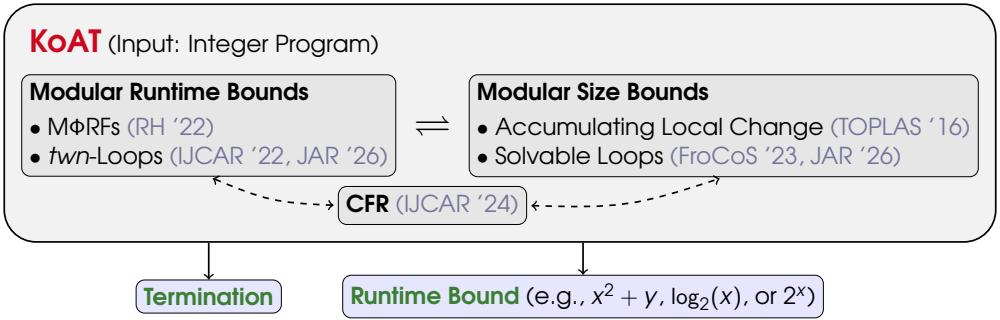
	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$< \omega$	$\mathrm{AVG}^+(\mathrm{s})$	Succ. Rate
KoAT	132	9	261	113	548	2.81	84 %
KoAT-2016	132	0	216	104	475	0.67	73 %
CoFloCo	125	0	230	93	457	1.50	70 %

Termination Analysis (Termination_ITS, 1222 Benchmarks)

- KoAT + LoAT can solve 1157 benchmarks

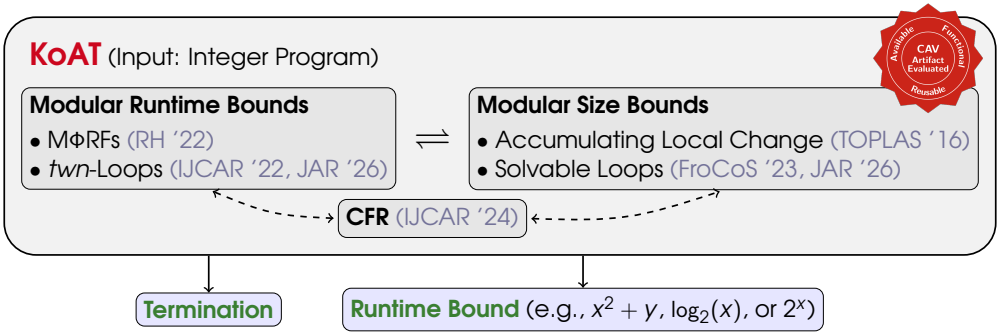


Conclusion



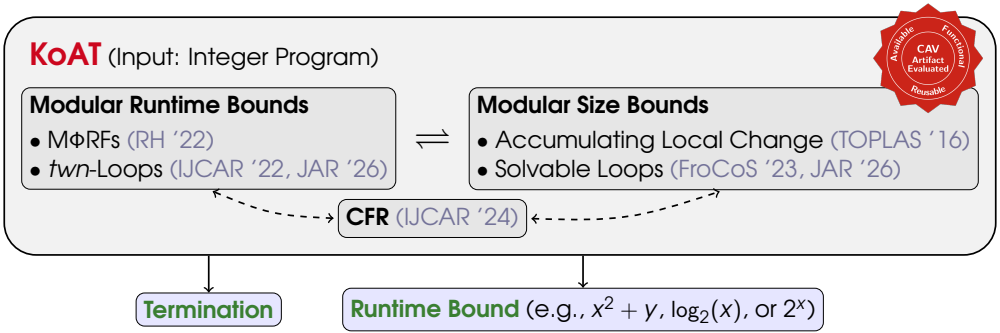
- Function calls (ESOP '26)
- Probabilistic programs (TACAS '21, IJCAR '24)

Conclusion



- Function calls (ESOP '26)
- Probabilistic programs (TACAS '21, IJCAR '24)

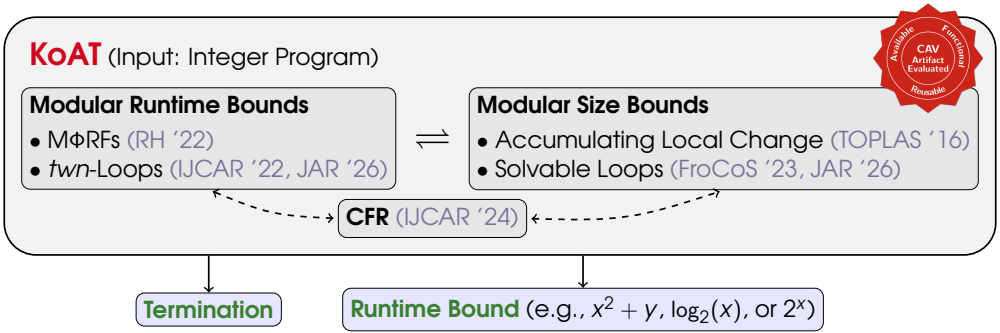
Conclusion



- Function calls (ESOP '26)
- Probabilistic programs (TACAS '21, IJCAR '24)

`https://koat.verify.rwth-aachen.de`

Conclusion



- Function calls (ESOP '26)
- Probabilistic programs (TACAS '21, IJCAR '24)

`https://koat.verify.rwth-aachen.de`

Thank You!