KoAT: An Automatic Complexity Analysis Tool for Integer Programs

Workshop on Termination 2023

Nils Lommen, Eleanore Meyer, and Jürgen Giesl
Motivation

**Goal:** Infer (upper) runtime bounds for “real-world” programs

```plaintext
while (x_3 > 0) do
    \[
    \begin{bmatrix}
    x_1 \\
    x_2
    \end{bmatrix} \leftarrow \begin{bmatrix}
    x_4 \\
    x_5
    \end{bmatrix}
    \]
    while (x_1^2 < x_2) do
        \[
        \begin{bmatrix}
        x_1 \\
        x_2
        \end{bmatrix} \leftarrow \begin{bmatrix}
        2 \cdot x_1 \\
        3 \cdot x_2
        \end{bmatrix}
        \]
    end
    \[
    x_3 \leftarrow x_3 - 1
    \]
end
```

▶ Does this program terminate?
▶ How often do we execute the inner loop?

• Solution: Use KoAT!

KoAT is an open-source complexity analysis tool for Integer Transition Systems.
Motivation

Goal: Infer (upper) runtime bounds for “real-world” programs

Does this program terminate?

\[
\text{while } (x_3 > 0) \text{ do}
\]
\[
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} x_4 \\ x_5 \end{bmatrix}
\]
\[
\text{while } (x_1^2 < x_2) \text{ do}
\]
\[
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix}
\]
\[
\text{end}
\]
\[
\begin{bmatrix} x_3 \end{bmatrix} \leftarrow [x_3 - 1]
\]
\[
\text{end}
\]
Motivation

Goal: Infer (upper) runtime bounds for “real-world” programs

while ($x_3 > 0$) do
    $[x_1 \ x_2] \leftarrow [[x_4 \ x_2^2] \ x_5]$
    while ($x_2^2 < x_2$) do
        $[x_1 \ x_2] \leftarrow [[2 \cdot x_1 \ 3 \cdot x_2]$
    end
    $[x_3] \leftarrow [x_3 - 1]$
end

▶ Does this program terminate?
▶ How often do we execute the inner loop?

Solution: Use KoAT

• Open-source complexity analysis tool for Integer Transition Systems
Motivation

**Goal:** Infer (upper) runtime bounds for “real-world” programs

```
while (x_3 > 0) do
    \[
    \begin{bmatrix}
        x_1 \\
        x_2
    \end{bmatrix}
    \leftarrow
    \begin{bmatrix}
        x_4 \\
        x^2_5
    \end{bmatrix}
    \]

while (x^2_1 < x_2) do
    \[
    \begin{bmatrix}
        x_1 \\
        x_2
    \end{bmatrix}
    \leftarrow
    \begin{bmatrix}
        2 \cdot x_1 \\
        3 \cdot x_2
    \end{bmatrix}
    \]
end

\[
\begin{bmatrix}
    x_3
\end{bmatrix}
\leftarrow
\begin{bmatrix}
    x_3 - 1
\end{bmatrix}
\]
end
```

- Does this program terminate?
- How often do we execute the inner loop?
  - Solution: Use KoAT!
Motivation

Goal: Infer (upper) runtime bounds for “real-world” programs

Does this program terminate?
How often do we execute the inner loop?
- Solution: Use KoAT!
- Open-source complexity analysis tool for Integer Transition Systems
Overview

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twloops
    [IJCAR '22]
  - $M\Phi$ RFs
    [RH '22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms
    [FroCoS '23]
  - changed accumulated size bounds
    [TOPLAS '16]

- local control flow-refinement by iRankFinder
  [RH '22].
Overview

- KoAT uses
  - a modular approach to compute time bounds combining
Overview

**KoAT uses**

- a modular approach to compute time bounds combining
  - a procedure to handle twin-loops
    [IJCAR '22]
Overview

- **KoAT uses**
  - a modular approach to compute time bounds combining
    - a procedure to handle twn-loops
    - MΦRFs [RH '22]
  - changed accumulated size bounds [TOPLAS '16]
Overview

*KoAT uses*

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops
    - MΦRFs [RH ‘22]
    - [IJCAR ‘22]
  - changed accumulated size bounds
    - [TOPLAS ‘16]

- a modular approach to compute size bounds combining
Overview

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops
    [IJCAR ’22]
  - MϕRFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms
    [FroCoS ’23]
Overview

**KoAT uses**

- a modular approach to compute time bounds combining
  - a procedure to handle twin-loops
    [IJCAR '22]
  - MΦRFs [RH '22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms
    [FroCoS '23]
  - changed accumulated size bounds
    [TOPLAS '16]
Overview

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twin-loops [IJCAR ’22]
  - MΦRFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS ’23]
  - changed accumulated size bounds [TOPLAS ’16]

- local control flow-refinement by iRankFinder [RH ’22].
Overview

KoAT uses

- **a modular approach** to compute time bounds combining
  - a procedure to handle twm-loops
    
    [IJCAR '22]
  
  - $\Phi$RFs [RH '22]

- **a modular approach** to compute size bounds combining
  
  - a procedure using closed-forms
    
    [FroCoS '23]
  
  - changed accumulated size bounds
    
    [TOPLAS '16]

- local control flow-refinement by iRankFinder [RH '22].
General Architecture

Translation to ITS

C Integer

while ($x_3 > 0$) do
\[
\begin{bmatrix}
    x_1 \\
    x_2
\end{bmatrix}
\leftarrow
\begin{bmatrix}
    x_4 \\
    x_5
\end{bmatrix}
\]
while ($x_2^2 < x_2$) do
\[
\begin{bmatrix}
    x_1 \\
    x_2
\end{bmatrix}
\leftarrow
\begin{bmatrix}
    2 \cdot x_1 \\
    3 \cdot x_2
\end{bmatrix}
\]
end
\[
\begin{bmatrix}
    x_3
\end{bmatrix}
\leftarrow
\begin{bmatrix}
    x_3 - 1
\end{bmatrix}
\]
end
General Architecture

- Translation to ITS

\[
\text{C Integer} \xrightarrow{\text{clang}} \text{LLVM}
\]

while \((x_3 > 0)\) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
x_4 \\
x_5^2
\end{bmatrix}
\]

while \((x_2^2 < x_2)\) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
2 \cdot x_1 \\
3 \cdot x_2
\end{bmatrix}
\]
end
\[
[x_3] \leftarrow [x_3 - 1]
\]
General Architecture

Translation to ITS

while \( (x_3 > 0) \) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
x_4 \\
x_5^2
\end{bmatrix}
\]
while \( (x_1^2 < x_2) \) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
2 \cdot x_1 \\
3 \cdot x_2
\end{bmatrix}
\]
end
\[
\begin{bmatrix}
x_3
\end{bmatrix} \leftarrow \begin{bmatrix}
x_3 - 1
\end{bmatrix}
\]
end
while \((x_3 > 0)\) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
x_4 \\
x_5
\end{bmatrix}
\]
while \((x_2 < x_2)\) do
\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} \leftarrow \begin{bmatrix}
2 \cdot x_1 \\
3 \cdot x_2
\end{bmatrix}
\]
end
\[
[x_3] \leftarrow [x_3 - 1]
\] end
General Architecture

Translation to ITS

\[
\text{while } (x_3 > 0) \text{ do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} x_4 \\ x_5 \end{bmatrix} \\
\text{while } (x_1^2 < x_2) \text{ do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix} \\
\text{end} \\
\begin{bmatrix} x_3 \end{bmatrix} \leftarrow \begin{bmatrix} x_3 - 1 \end{bmatrix} \\
\text{end}
\]
General Architecture

Translation to ITS

C Integer \xrightarrow{\text{clang}} \text{LLVM} \xrightarrow{\text{llvm2kittel}} \text{ITS} \xrightarrow{\text{KoAT}} \text{Upper Bound} \xrightarrow{\text{AProve}} \text{Termination}

\begin{align*}
\text{while} \ (x_3 > 0) \ & \text{do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & \leftarrow \begin{bmatrix} x_4 \\ x_5 \end{bmatrix} \\
\text{while} \ (x_2^2 < x_2) \ & \text{do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & \leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix} \\
\text{end} \\
\begin{bmatrix} x_3 \end{bmatrix} & \leftarrow \begin{bmatrix} x_3 - 1 \end{bmatrix} \\
\text{end}
\end{align*}
General Architecture

Translation to ITS

C Integer \xrightarrow{\text{clang}} \text{LLVM} \xrightarrow{\text{llvm2kittel}} \text{ITS} \xrightarrow{\text{KoAT}} \text{Upper Bound}

\begin{align*}
\text{while } (x_3 > 0) \text{ do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &\leftarrow \begin{bmatrix} x_4 \\ x_5 \end{bmatrix} \\
\text{while } (x_2^2 < x_2) \text{ do} \\
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &\leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix} \\
\text{end} \\
\begin{bmatrix} x_3 \end{bmatrix} &\leftarrow \begin{bmatrix} x_3 - 1 \end{bmatrix} \\
\text{end}
\end{align*}
## General Architecture

### Translation to ITS

![Translation Diagram]

```
while (x_3 > 0) do
    \[ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} x_4 \\ x_5^2 \end{bmatrix} \]
    while (x_1^2 < x_2) do
        \[ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix} \]
    end
    \[ x_3 \leftarrow x_3 - 1 \]
end
```
General Architecture

► Translation to ITS

C Integer $\xrightarrow{\text{clang}}$ LLVM $\xrightarrow{\text{llvm2kittel}}$ ITS $\xrightarrow{\text{KoAT}}$ AProVE $\xrightarrow{\text{Termination}}$

while ($x_3 > 0$) do

$$
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} x_4 \\ x_5^2 \end{bmatrix}
$$

while ($x_1^2 < x_2$) do

$$
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leftarrow \begin{bmatrix} 2 \cdot x_1 \\ 3 \cdot x_2 \end{bmatrix}
$$

end

$$
\begin{bmatrix} x_3 \end{bmatrix} \leftarrow [x_3 - 1]
$$
end

$t_0 : \tau = x_3 > 0$

$\eta(x_1) = x_4$

$\eta(x_2) = x_5^2$

$t_1: \eta(x_1) = x_4$

$t_2: \eta(x_2) = x_5^2$

$t_3: \tau = x_1^2 < x_2$

$\eta(x_1) = 2 \cdot x_1$

$\eta(x_2) = 3 \cdot x_2$
General Architecture

- Preprocess Programs:
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more
  - Time bounds: How many executions?
  - Size bounds: What is the value of a variable after evaluating transition?
  - Analyze SCCs one after another
  - Propagate information from "previous" SCCs to "later" SCCs via size bounds

[TOPLAS '16]
General Architecture

- Preprocess Programs:
  - Invariants by Apron
General Architecture

► Preprocess Programs:
  • Invariants by Apron
  • Remove unsatisfiable Transitions
General Architecture

- Preprocess Programs:
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations

TOPLAS '16
General Architecture

- Preprocess Programs:
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ...
General Architecture

- **Preprocess Programs:**
  - Invariants by *Apron*
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more …

- **Time bounds:** How many executions?
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more …

- **Time bounds:** How many executions?

- **Size bounds:** What is the value of a variable after evaluating transition?
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ...

- **Time bounds:** How many executions?
- **Size bounds:** What is the value of a variable after evaluating transition?
General Architecture

- **Preprocess Programs:**
  - Invariants by *Apron*
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more . . .

- **Time bounds:** How many executions?

- **Size bounds:** What is the value of a variable after evaluating transition?
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ...

- **Time bounds:** How many executions?
- **Size bounds:** What is the value of a variable after evaluating transition?
General Architecture

- Preprocess Programs:
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more …

- **Time** bounds: How many executions?
- **Size** bounds: What is the value of a variable after evaluating transition?

![Diagram of a control flow graph with labels and arrows indicating control flow and data dependencies.](image-url)
General Architecture

- Preprocess Programs:
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more …

- Time bounds: How many executions?
- Size bounds: What is the value of a variable after evaluating transition?
- Analyze SCCs one after another
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ...

- **Time bounds:** How many executions?
- **Size bounds:** What is the value of a variable after evaluating transition?
- Analyze SCCs one after another
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ...

- **Time bounds:** How many executions?
- **Size bounds:** What is the value of a variable after evaluating transition?

- Analyze SCCs one after another
General Architecture

- **Preprocess Programs:**
  - Invariants by *Apron*
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more ... (and so on)

- **Time bounds:** How many executions?
- **Size bounds:** What is the value of a variable after evaluating transition?
- Analyze SCCs one after another
- Propagate information from “previous” SCCs to “later” SCCs via *size bounds* ([TOPLAS ’16])
General Architecture

- **Preprocess Programs:**
  - Invariants by Apron
  - Remove unsatisfiable Transitions
  - Remove unreachable Locations
  - many more …

- **Time bounds:** How many executions?

- **Size bounds:** What is the value of a variable after evaluating transition?

- Analyze SCCs one after another

- Propagate information from “previous” SCCs to “later” SCCs via *size bounds* [TOPLAS ’16]
General Architecture – Analyzing an SCC

Alternatingly compute **time** and **size** bounds
Alternatingly compute time and size bounds

- Compute initial global size bounds
General Architecture – Analyzing an SCC

- Alternatingly compute **time** and **size** bounds
  - Compute initial **global size** bounds
  - Compute as many finite **global time** bounds as possible
Alternatingly compute **time** and **size** bounds
- Compute initial **global size** bounds
- Compute as many finite **global time** bounds as possible
- Improve **global size** bounds

Analysis results are passed on via **global size** bounds of connecting transitions
General Architecture – Analyzing an SCC

- Alternatingly compute time and size bounds
  - Compute initial global size bounds
  - Compute as many finite global time bounds as possible
  - Improve global size bounds
  - Improve global time bounds
General Architecture – Analyzing an SCC

- Alternatingly compute time and size bounds
  - Compute initial global size bounds
  - Compute as many finite global time bounds as possible
  - Improve global size bounds
  - Improve global time bounds
  - Improve global size bounds

...
General Architecture – Analyzing an SCC

► Alternatingly compute time and size bounds
  • Compute initial global size bounds
  • Compute as many finite global time bounds as possible
  • Improve global size bounds
  • Improve global time bounds
  • Improve global size bounds
  • ...

Possibly apply sub-SCC CFR with $i$RankFinder

Analysis results are passed on via global size bounds of connecting transitions
General Architecture – Analyzing an SCC

- Alternatingly compute time and size bounds
  - Compute initial global size bounds
  - Compute as many finite global time bounds as possible
  - Improve global size bounds
  - Improve global time bounds
  - Improve global size bounds
  - ...

- Possibly apply sub-SCC CFR with iRankFinder
General Architecture – Analyzing an SCC

- Alternatingly compute time and size bounds
  - Compute initial global size bounds
  - Compute as many finite global time bounds as possible
  - Improve global size bounds
  - Improve global time bounds
  - Improve global size bounds
  - ...

- Possibly apply sub-SCC CFR with iRankFinder

- Analysis results are passed on via global size bounds of connecting transitions
Overview

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle tw-loop [IJCAR ’22]
  - $M\Phi$RFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS ’23]
  - changed accumulated size bounds [TOPLAS ’16]

- local control flow-refinement by iRankFinder [RH ’22].
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]

In order to bound $t < w.r.t. subprogram $T'$, $t <$ decreases the value of the ranking function. If $T \subseteq \{ t < \}$ does not increase the value of the ranking function, we can lift the local time bound for the subprogram to a time bound for the complete program.

\[ \ell_0 \leq \ell_1 \leq \ell_2 \]

For $t_0$: \[ \tau = x^3 > 0 \]

For $t_1$: \[ \eta(x_1) = x_4 \]

For $t_2$: \[ \eta(x_2) = 3 \cdot x_2 \]

For $t_3$: \[ \tau = x_2^1 < x_2 \]

For $t_4$: \[ \eta(x_1) = 2 \cdot x_1 \]

For $t_5$: \[ \eta(x_2) = 3 \cdot x_2 \]

$t < t_2$ and $T = \{ t_1, t_2, t_3 \}$

The ranking function $x_3$ yields a well-founded order on $\mathbb{N}$. Consider the entry transition $t_0$.
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t < w.r.t.$ subprogram $T'$
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t_<$ w.r.t. subprogram $\mathcal{T}'$

\begin{align*}
\ell_0 & \xrightarrow{t_0} \ell_1 \\
\ell_1 & \xrightarrow{t_1} \ell_2 \\
\ell_2 & \xrightarrow{t_3} \ell_2
\end{align*}

\begin{align*}
t_1 : \tau &= x_3 > 0 \\
\eta(x_1) &= x_4 \\
\eta(x_2) &= x_5^2 \\
t_2 : \eta(x_3) &= x_3 - 1 \\
t_3 : \tau &= x_1^2 < x_2 \\
\eta(x_1) &= 2 \cdot x_1 \\
\eta(x_2) &= 3 \cdot x_2
\end{align*}
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t_\prec$ w.r.t. subprogram $\mathcal{T}'$

\begin{itemize}
  \item $t_1: \tau = x_3 > 0$
  \item $\eta(x_1) = x_4$
  \item $\eta(x_2) = x_5^2$
  \item $t_2: \eta(x_3) = x_3 - 1$
  \item $t_3: \tau = x_1^2 < x_2$
  \item $\eta(x_1) = 2 \cdot x_1$
  \item $\eta(x_2) = 3 \cdot x_2$
  \item $t_\prec = t_2$ and $\mathcal{T} = \{t_1, t_2, t_3\}$
\end{itemize}
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t_<$ w.r.t. subprogram $\mathcal{T}'$
  - $t_<$ decreases value of ranking function

$t_2 = t_2$ and $\mathcal{T} = \{t_1, t_2, t_3\}$
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound \( t_\prec \) w.r.t. subprogram \( T' \)
  - \( t_\prec \) decreases value of ranking function
  - \( T_\subseteq \setminus \{ t_\prec \} \) does not increase value of ranking function

\[
\begin{align*}
\ell_0 & \xrightarrow{t_0} \ell_1 & t_1 : \tau = x_3 > 0 \\
\ell_1 & \xrightarrow{t_2} \ell_2 & t_2 : \eta(x_3) = x_3 - 1 \\
\ell_2 & \xrightarrow{t_3} \ell_1 & t_3 : \tau = x_1^2 < x_2 \\
& & \eta(x_1) = 2 \cdot x_1 \\
& & \eta(x_2) = 3 \cdot x_2
\end{align*}
\]

\( t_\prec = t_2 \) and \( T = \{ t_1, t_2, t_3 \} \)
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t_<$ w.r.t. subprogram $T'$
  - $t_<$ decreases value of ranking function
  - $T_\subseteq \{ t_< \}$ does not increase value of ranking function

- $t_\leq t_2$ and $T = \{ t_1, t_2, t_3 \}$
- ranking function $x_3$ yields well-founded order on $\mathbb{N}$
Analyzing Programs: Time Bounds

► Triangular Weakly Non-Linear Loops [IJCAR ’22]
► Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound $t_<$ w.r.t. subprogram $T'$
  - $t_<$ decreases value of ranking function
  - $T_\subseteq \{t_<\}$ does not increase value of ranking function
  - Lift local time bound for subprogram to time bound for complete program

$t_<$ = $t_2$ and $\mathcal{T} = \{t_1, t_2, t_3\}$

- ranking function $x_3$ yields well-founded order on $\mathbb{N}$

\[
\begin{align*}
\ell_0 & \xrightarrow{t_0} \ell_1 \xrightarrow{t_1} \ell_2 \\
\ell_0 & \xrightarrow{t_2} \ell_1 \\
\ell_1 & \xrightarrow{t_3} \ell_2
\end{align*}
\]

- $t_1: \tau = x_3 > 0$
- $\eta(x_1) = x_4$
- $\eta(x_2) = x_5^2$
- $t_2: \eta(x_3) = x_3 - 1$
- $t_3: \tau = x_1^2 < x_2$
- $\eta(x_1) = 2 \cdot x_1$
- $\eta(x_2) = 3 \cdot x_2$
Analyzing Programs: Time Bounds

- Triangular Weakly Non-Linear Loops [IJCAR ’22]
- Multiphase-Linear Ranking Functions [Ben-Amram, Genaim] & [RH ’22]
  - Try to bound \( t_\prec \) w.r.t. subprogram \( \mathcal{T}' \)
  - \( t_\prec \) decreases value of ranking function
  - \( \mathcal{T}_\subseteq \setminus \{ t_\prec \} \) does not increase value of ranking function
  - Lift local time bound for subprogram to time bound for complete program

\[
\begin{align*}
\ell_0 & \quad t_0 \quad \ell_1 \\
\ell_1 & \quad t_1 : \tau = x_3 > 0 \\
\ell_1 & \quad \eta(x_1) = x_4 \\
\ell_1 & \quad \eta(x_2) = x_5^2 \\
\ell_2 & \quad t_2 : \eta(x_3) = x_3 - 1 \\
\ell_2 & \quad t_3 : \tau = x_1^2 < x_2 \\
\ell_2 & \quad \eta(x_1) = 2 \cdot x_1 \\
\ell_2 & \quad \eta(x_2) = 3 \cdot x_2 \\
\end{align*}
\]

- \( t_\prec = t_2 \) and \( \mathcal{T} = \{ t_1, t_2, t_3 \} \)
- ranking function \( x_3 \) yields well-founded order on \( \mathbb{N} \)
- consider entry transition \( t_0 \)
Overview

- **KoAT uses**
  - A modular approach to compute time bounds combining
    - A procedure to handle twin-loops
      - MΦRFs [RH '22]
    - Size bounds combining
      - A procedure using closed-forms
        - Changed accumulated size bounds [TOPLAS '16]
      - Local control flow-refinement by iRankFinder [RH '22].
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop

```plaintext
while (x₁ > 0) do
    \[
    \begin{bmatrix} x₁ \\ x₂ \end{bmatrix} \leftarrow \begin{bmatrix} x₁ - 1 \\ x₂ + x₁² \end{bmatrix}
    \]
end
```
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop

```
while (x_1 > 0) do
  [x_1] ← [x_1 - 1]
  [x_2] ← [x_2 + x_1^2]
end
```

- Closed form:

$$\text{cl}_{x_2}^n = x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 - x_1 \cdot n - \frac{n^2}{2} + \frac{n^2}{3} \right)$$
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression

```plaintext
while (x_1 > 0) do
    \[
    \begin{bmatrix}
        x_1 \\
        x_2
    \end{bmatrix} \leftarrow \begin{bmatrix}
        x_1 - 1 \\
        x_2 + x_1^2
    \end{bmatrix}
    \]
end
```

- Closed form:
  \[
  \text{cl}_x^n x_2 = x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 - x_1 \cdot n - \frac{n}{2} + \frac{n^2}{3} \right)
  \]
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression

```plaintext
while (x₁ > 0) do
  [x₁] ← [x₁ - 1]
  [x₂] ← [x₂ + x₁^2]
end
```

- Closed form:
  \[ cl_{x₂}^n = x₂ + n \cdot \left( \frac{1}{6} + x₁ + x₁^2 - x₁ \cdot n - \frac{n^2}{2} + \frac{n^2}{3} \right) \]

- Over-approximation:
  \[ x₂ + n \cdot \left( \frac{1}{6} + x₁ + x₁^2 + x₁ \cdot n + \frac{n^2}{2} + \frac{n^2}{3} \right) \]
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute \( n \) by runtime bound

```
while (x_1 > 0) do
    \[
    \begin{bmatrix}
    x_1 \\
    x_2
    \end{bmatrix}
    \leftarrow
    \begin{bmatrix}
    x_1 - 1 \\
    x_2 + x_1^2
    \end{bmatrix}
    \]
end
```

- Closed form:
  \[
  cl_{x_2}^n = x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 - x_1 \cdot n - \frac{n^2}{2} + \frac{n^2}{3} \right)
  \]

- Over-approximation:
  \[
  x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 + x_1 \cdot n + \frac{n}{2} + \frac{n^2}{3} \right)
  \]
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute $n$ by runtime bound

```plaintext
while ($x_1 > 0$) do
  $[x_1] \leftarrow [x_1 - 1]$
  $[x_2] \leftarrow [x_2 + x_1^2]$
end
```

- Closed form:
  $$\text{cl}_{x_2}^n = x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 - x_1 \cdot n - \frac{n^2}{2} + \frac{n^3}{3} \right)$$

- Over-approximation:
  $$x_2 + n \cdot \left( \frac{1}{6} + x_1 + x_1^2 + x_1 \cdot n + \frac{n^2}{2} + \frac{n^3}{3} \right)$$

- Size bound:
  $$x_2 + x_1 \cdot \left( \frac{1}{6} + x_1 + x_1^2 + x_1 \cdot x_1 + \frac{x_1}{2} + \frac{x_1^2}{3} \right)$$
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute $n$ by runtime bound
- Changed accumulated size bounds [TOPLAS ’16]
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute $n$ by runtime bound
- Changed accumulated size bounds [TOPLAS ’16]
  - Construct variable dependence graph
Analyzing Programs: Size Bounds

- **Size bounds by closed forms** [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute \( n \) by runtime bound

- **Changed accumulated size bounds** [TOPLAS ’16]
  - Construct variable dependence graph
Analyzing Programs: Size Bounds

- Size bounds by closed forms [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute $n$ by runtime bound

- Changed accumulated size bounds [TOPLAS ’16]
  - Construct variable dependence graph
  - Connecting transitions: Directly apply updates
Analyzing Programs: Size Bounds

- **Size bounds by closed forms** [FroCoS ’23]
  - Compute closed form for loop
  - Over-approximate closed form to non-negative, weakly monotonic increasing expression
  - Substitute \( n \) by runtime bound
- **Changed accumulated size bounds** [TOPLAS ’16]
  - Construct variable dependence graph
  - Connecting transitions: Directly apply updates
  - SCC transitions: Capture repetitions by using runtime bounds
Overview

**KoAT uses**

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops \[IJCAR '22\]
  - MΦRFs \[RH '22\]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms \[FroCoS '23\]
  - changed accumulated size bounds \[TOPLAS '16\]

- local control flow-refinement by iRankFinder \[RH '22\].
Problem: complex, nested loops

while (x > 0) do
  if (y > 0) then
    y ← y − x
  else
    x ← x − 1
  end
end
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of *two* phases:

```plaintext
while (x > 0) do
  if (y > 0) then
    y ← y − x
  else
    x ← x − 1
  end
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of two phases:
  1. *then-case* is repeated until $y \leq 0$

```plaintext
while (x > 0) do
  if (y > 0) then
    y ← y − x
  else
    x ← x − 1
end
```

while (x > 0 ∧ y > 0) do
  y ← y − x
end

while (x > 0 ∧ y ≤ 0) do
  x ← x − 1
end
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of two phases:
  1. **then-case** is repeated until $y \leq 0$
  2. **else-case** is repeated until $x \leq 0$

```c
while (x > 0) do
    if (y > 0) then
        y ← y − x
    else
        x ← x − 1
    end
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of *two* phases:
  1. *then-*case is repeated until $y \leq 0$
  2. *else-*case is repeated until $x \leq 0$

  ⇒ No run, where *second* phase is executed before *first* phase

```
while (x > 0) do
  if (y > 0) then
    y ← y − x
  else
    x ← x − 1
  end
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops

  Loop consists of *two* phases:

  1. **then-case** is repeated until \( y \leq 0 \)
  2. **else-case** is repeated until \( x \leq 0 \)

  \( \Rightarrow \) No run, where *second* phase is executed before *first* phase

Control-Flow Refinement by Partial Evaluation (CFR) [Doménech et al. ’19]

```plaintext
while (x > 0) do
    if (y > 0) then
        y ← y − x
    else
        x ← x − 1
end
```

WST 2023
Nils Lommen, Eleanore Meyer, and Jürgen Giesl
RWTH Aachen University – LuFGi2
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops

- Loop consists of *two* phases:
  1. *then-case* is repeated until $y \leq 0$
  2. *else-case* is repeated until $x \leq 0$

  $\Rightarrow$ No run, where *second* phase is executed before *first* phase

Control-Flow Refinement by Partial Evaluation (CFR) [Doménech et al. ’19]

- sort out certain program paths

```python
while (x > 0) do
    if (y > 0) then
        y ← y − x
    else
        x ← x − 1
    end
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of *two* phases:
  1. **then-case** is repeated until \( y \leq 0 \)
  2. **else-case** is repeated until \( x \leq 0 \)

  \[\Rightarrow\] No run, where **second** phase is executed before **first** phase

**Control-Flow Refinement by Partial Evaluation (CFR)** [Doménech et al. ’19]
- sort out certain program paths

```plaintext
while (x > 0) do
    if (y > 0) then
        y ← y − x
    else
        x ← x − 1
    end
end
```

```plaintext
while (x > 0 ∧ y > 0) do
    y ← y − x
end
while (x > 0 ∧ y ≤ 0) do
    x ← x − 1
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of *two* phases:
  1. *then*-case is repeated until \( y \leq 0 \)
  2. *else*-case is repeated until \( x \leq 0 \)

\[ \Rightarrow \] No run, where *second* phase is executed before *first* phase

Control-Flow Refinement by Partial Evaluation (CFR) [Doménech et al. ’19]

- sort out certain program paths
- \( \Rightarrow \) integrate CFR into our modular approach

```latex
while (x > 0) do
  if (y > 0) then
    y ← y − x
  else
    x ← x − 1
  end
\end{verbatim} 
```

```
while (x > 0 ∧ y > 0) do
  y ← y − x
end
while (x > 0 ∧ y ≤ 0) do
  x ← x − 1
end
```
Integrating Control-Flow Refinement

- **Problem**: complex, nested loops
- Loop consists of *two* phases:
  1. *then*-case is repeated until $y \leq 0$
  2. *else*-case is repeated until $x \leq 0$

  $\Rightarrow$ No run, where second phase is executed before first phase

Control-Flow Refinement by Partial Evaluation (CFR) [Doménech et al. ’19]

- sort out certain program paths
  $\Rightarrow$ integrate CFR into our modular approach
- CFR *modular* for SCCs with “problematic” transitions on-demand [RH ’22]
(GOAL COMPLEXITY)
(STARTTERM (FUNCTIONSYMBOLS 10))
(VAR A B C D E)
(RULES
  l0(A,B,C,D,E) -> l1(A,B,C,D,E) :|: C > 0
  l1(A,B,C,D,E) -> l2(D,E^2,C,D,E) :|: C > 0
  l2(A,B,C,D,E) -> l2(2*A,3*B,C,D,E) :|: A^2 < B && A > 0
  l2(A,B,C,D,E) -> l1(A,B,C - 1,D,E)
)

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

- KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle tw-loops [IJCAR '22]
  - $M^\Phi$ RFs [RH '22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS '23]
  - changed accumulated size bounds [TOPLAS '16]

- local control flow-refinement by iRankFinder [RH '22].

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

Thank You!
Conclusion

- **KoAT uses**
  - a modular approach to compute time bounds combining

- **KoAT uses**
  - a modular approach to compute size bounds combining

- **KoAT uses**
  - local control flow-refinement by iRankFinder [RH '22].

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

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops
    [IJCAR '22]
Conclusion

- **KoAT uses**
  - a modular approach to compute time bounds combining
    - a procedure to handle twn-loops [IJCAR '22]
    - MΦRFs [RH '22]
  - local control flow-refinement by iRankFinder [RH '22].

Thank You!
Conclusion

**KoAT uses**

- a modular approach to compute time bounds combining
  - a procedure to handle twin-loops [IJCAR '22]
  - MΦRFs [RH '22]

- a modular approach to compute size bounds combining

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

Thank You!
Conclusion

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops [IJCAR '22]
  - $\Phi$RFs [RH '22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS '23]

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

Thank You!
Conclusion

- **KoAT uses**
  - a modular approach to compute time bounds combining
    - a procedure to handle twl-loops [IJCAR '22]
    - MΦRFs [RH '22]
  - a modular approach to compute size bounds combining
    - a procedure using closed-forms [FroCoS '23]
    - changed accumulated size bounds [TOPLAS '16]
Conclusion

- **KoAT uses**
  - a modular approach to compute time bounds combining
    - a procedure to handle twn-loops
    - MΦRFs [RH ’22]
    [IJCAR ’22]
  - a modular approach to compute size bounds combining
    - a procedure using closed-forms
    - changed accumulated size bounds
    [FroCoS ’23]
    [TOPLAS ’16]
  - local control flow-refinement by iRankFinder [RH ’22].
Conclusion

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twn-loops [IJCAR ’22]
  - MΦRFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS ’23]
  - changed accumulated size bounds [TOPLAS ’16]

- local control flow-refinement by iRankFinder [RH ’22].

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

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twin-loops [IJCAR ’22]
  - MΦRFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS ’23]
  - changed accumulated size bounds [TOPLAS ’16]

- local control flow-refinement by iRankFinder [RH ’22].

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

KoAT uses

- a modular approach to compute time bounds combining
  - a procedure to handle twl-loops [IJCAR ’22]
  - MΦRFs [RH ’22]

- a modular approach to compute size bounds combining
  - a procedure using closed-forms [FroCoS ’23]
  - changed accumulated size bounds [TOPLAS ’16]

- local control flow-refinement by iRankFinder [RH ’22].

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

Thank You!