VERDI 2025

Reasoning with Happens-Before Relations about Concurrent Programs in the Theta Framework

Short toolpaper

Csanád Telbisz, Levente Bajczi, Dániel Szekeres, András Vörös, István Majzik

December 7, 2025

MŰEGYETEM 1782

Critical Systems Research Group

Agenda

Introduction

Context: what is the problem?
What do we do? Why do we care?

Agenda

Introduction

Context: what is the problem?
What do we do? Why do we care?

Verification approaches (existing & novel)
Optimization techniques (novel)

Techniques

Agenda

Introduction

Context: what is the problem?
What do we do? Why do we care?

Verification approaches (existing & novel)
Optimization techniques (novel)

Techniques

Evaluation

Experiment and data analysis
Evaluation of impact

Agenda

Introduction

Context: what is the problem?
What do we do? Why do we care?

Verification approaches (existing & novel)
Optimization techniques (novel)

Techniques

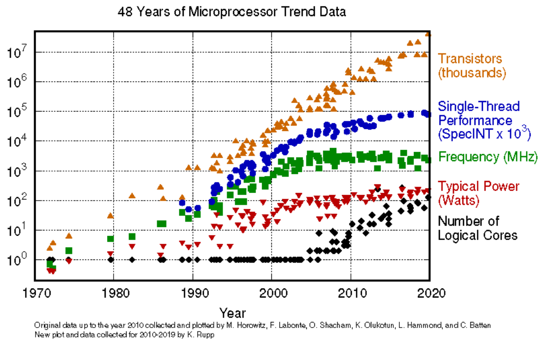
Evaluation

Experiment and data analysis
Evaluation of impact

Theta as a Verifier
Theta as a Framework

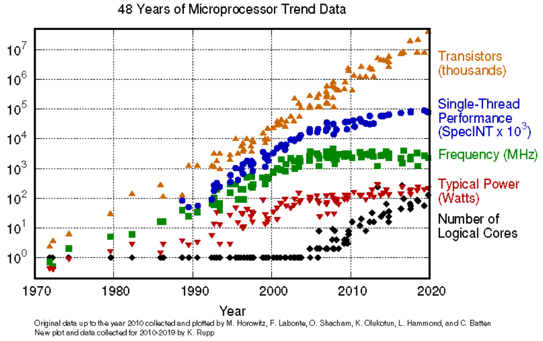
Availability

Context


48 Years of Microprocessor Trend Data
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
Year
1970
1980
1990
2000
2010
2020
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Context

- Safety critical systems need **multicore** CPUs
- Further performance: **relaxed** memory models

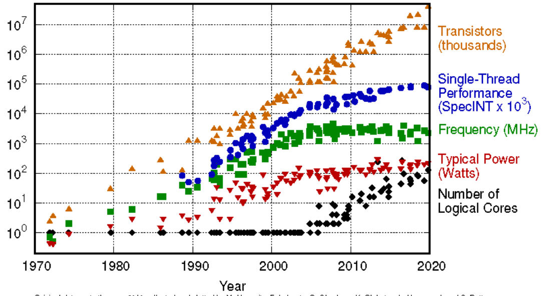


Context

- Safety critical systems need **multicore** CPUs
- Further performance: **relaxed** memory models

We need new formal techniques!

48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Introduction

- Emerging method for verifying concurrency: **Happens-before relation**
  - Decouples data- and control-flow
  - Enables semi-modular verification

Introduction

- Emerging method for verifying concurrency: **Happens-before relation**
  - Decouples data- and control-flow
  - Enables semi-modular verification

$$\xrightarrow{po}$$

$$\begin{tikzcd}
& \mathbf{x} := 0 \ \mathbf{y} := 0 \arrow[dl] \arrow[dr] & \\
\mathbf{x} := 1 \arrow[d] & & i := \mathbf{y} \arrow[d] \\
\mathbf{y} := 1 & & j := \mathbf{x}
\end{tikzcd}$$

Introduction

- Emerging method for verifying concurrency: **Happens-before relation**
  - Decouples data- and control-flow
  - Enables semi-modular verification

$$\xrightarrow{po} \quad \xrightarrow{rf}$$

$$\begin{tikzcd}
 & \mathbf{x} := 0 \ \mathbf{y} := 0 \arrow[dl] \arrow[dr] \arrow[ddr, bend left, red] & \\
\mathbf{x} := 1 \arrow[d] & & i := \mathbf{y} \arrow[d] \\
\mathbf{y} := 1 \arrow[urr, red] & & j := \mathbf{x}
\end{tikzcd}$$

Introduction

- Emerging method for verifying concurrency: **Happens-before relation**
  - Decouples data- and control-flow
  - Enables semi-modular verification

$\xrightarrow{po}$ $\xrightarrow{rf}$ $\xrightarrow{co}$

$$\begin{tikzcd}
\mathbf{x} := 0\ \mathbf{y} := 0 \arrow[d] \arrow[d, bend right] \arrow[dd, bend right=60] \arrow[dr] \arrow[ddr, bend left=60] & \\
\mathbf{x} := 1 \arrow[d] & i := \mathbf{y} \arrow[d] \\
\mathbf{y} := 1 \arrow[ur] & j := \mathbf{x}
\end{tikzcd}$$

Introduction

- Emerging method for verifying concurrency: **Happens-before relation**
  - Decouples data- and control-flow
  - Enables semi-modular verification

$$\xrightarrow{po} \quad {\color{red}\xrightarrow{rf}} \quad {\color{orange}\xrightarrow{co}}$$

$$\begin{tikzcd}
 & \mathbf{x} := 0 \ \mathbf{y} := 0 \arrow[dl] \arrow[dl, orange, bend right=10] \arrow[dr] \arrow[ddl, orange, bend right=50] \arrow[ddr, red, bend left=50] & \\
\mathbf{x} := 1 \arrow[dd] & & i := \mathbf{y} \arrow[dd] \\
 & & \\
\mathbf{y} := 1 \arrow[uurr, red] & & j := \mathbf{x}
\end{tikzcd}$$

- Causality and memory semantics: happens-before (*hb*-) order
- **Verification idea**: encode threads *and* happens-before constraints
- **Error property**: reachable thread state, and $\exists$ consistent *hb*-order

Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

initially: $\mathtt{x}_0 = \mathtt{y}_0 = 0$

Thread t_1 — Thread t_2

```
if(y₁==1) x₁=1;   y₃=x₃;
else x₂=1-y₂;     x₄=1-y₄;
```

finally: `assert(x₅==1 || y₅==1)`

Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

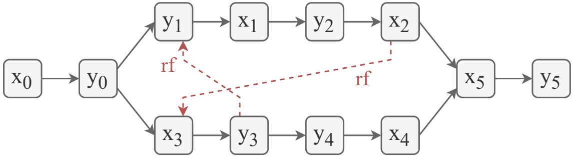
Is any assertion violated?

initially: $x_0 = y_0 = 0$

Thread t_1 | Thread t_2

```
if(y_1==1) x_1=1;     y_3=x_3;
else  x_2=1-y_2;      x_4=1-y_4;
```

finally: `assert(x_5==1 || y_5==1)`



Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

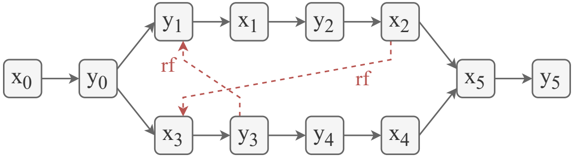
initially: $x_0 = y_0 = 0$

Thread t_1 | Thread t_2

```
if(y₁==1) x₁=1;   y₃=x₃;
else x₂=1-y₂;     x₄=1-y₄;
```

finally: `assert(x₅==1 || y₅==1)`

Unroll



Verification Approach (Overview)

Given a program with…

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

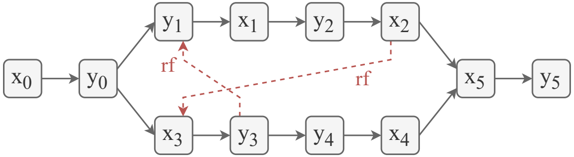
initially: $x_0 = y_0 = 0$

Thread t_1 | Thread t_2

```
if(y_1==1) x_1=1;    y_3=x_3;
else x_2=1-y_2;      x_4=1-y_4;
```

finally: `assert(x_5==1 || y_5==1)`

Unroll → CSSA



Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

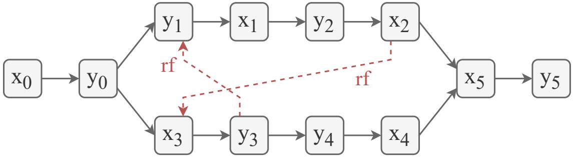
Is any assertion violated?

initially: $x_0 = y_0 = 0$

Thread t_1 Thread t_2

```
if(y₁==1) x₁=1;    y₃=x₃;
else x₂=1-y₂;      x₄=1-y₄;
```

finally: `assert(x₅==1 || y₅==1)`



Unroll → CSSA → SMT

Verification Approach (Overview)

Given a program with...

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

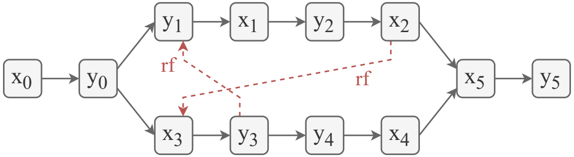
Is any assertion violated?

initially: x_0 = y_0 = 0

Thread t_1 | Thread t_2

```
if(y_1==1) x_1=1;   y_3=x_3;
else x_2=1-y_2;     x_4=1-y_4;
```

finally: `assert(x_5==1 || y_5==1)`



Unroll → CSSA → SMT → Solve

Verification Approach (Overview)

Given a program with...

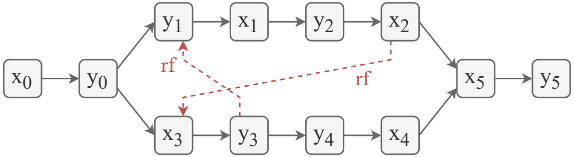
POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

initially: $x_0 = y_0 = 0$

Thread t_1	Thread t_2
`if(`y_1`==1)` x_1`=1;`	y_3`=`x_3`;`
`else` x_2`=1-`y_2`;`	x_4`=1-`y_4`;`

finally: `assert(`x_5`==1 || `y_5`==1)`



Unroll → CSSA → SMT → Solve → Verdict

Verification Approach (Overview)

Given a program with…

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

initially: $x_0 = y_0 = 0$

Thread t_1 | Thread t_2

```
if(y1==1) x1=1;    y3=x3;
else x2=1-y2;      x4=1-y4;
```

finally: assert(x_5 == 1 || y_5 == 1)

$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[dddl, phantom] \arrow[ddlll, dashed, "rf"'] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[uul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

Unroll → CSSA → SMT → Solve → Verdict

Optimizations

Verification Approach (Overview)

Given a program with…

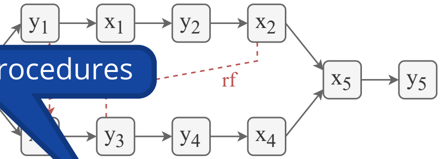
POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

Is any assertion violated?

initially: $x_0 = y_0 = 0$

Thread t_1	Thread t_2
`if(`y_1`==1) `x_1`=1;`	y_3`=`x_3`;`
`else `x_2`=1-`y_2`;`	x_4`=1-`y_4`;`

finally: `assert(`x_5`==1 || `y_5`==1);`



Decision procedures

Optimizations

$$\begin{tikzcd}\text{Unroll} \arrow[r] & \text{CSSA} \arrow[r] & \text{SMT} \arrow[r] & \text{Solve} \arrow[r] & \text{Verdict}\end{tikzcd}$$

Verification Approach (Overview)

Given a program with…

POSIX/C11 threads
Nondeterministic inputs
Thread-local assertions

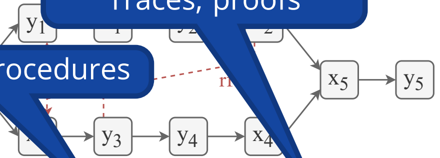
Is any assertion violated?

initially: x_0 = y_0 = 0

Thread t_1 Thread t_2

```
if(y₁==1) x₁=1;   y₃=x₃;
else x₂=1-y₂;
```

finally: assert(x_5=



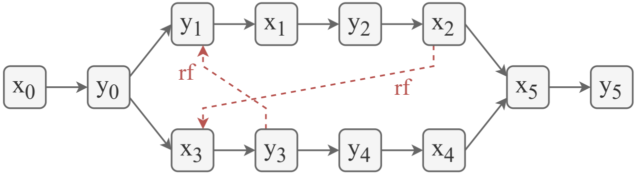
Traces, proofs

Decision procedures

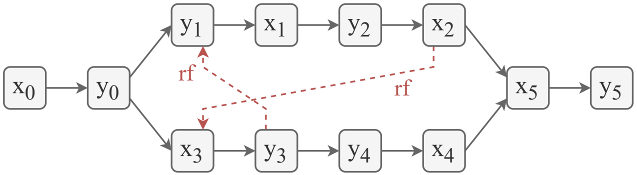
Optimizations

$$\begin{tikzcd} \text{Unroll} \arrow[r] & \text{CSSA} \arrow[r] & \text{SMT} \arrow[r] & \text{Solve} \arrow[r] & \text{Verdict} \end{tikzcd}$$

Techniques - Integer Difference Logic

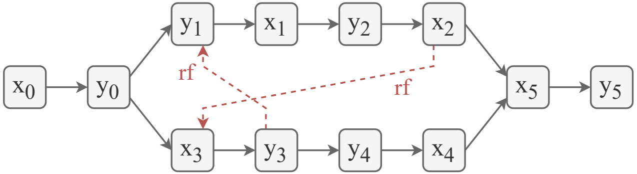


Techniques - Integer Difference Logic



- Each event has a *clock* integer variable
- A happens-before order implies an integer order

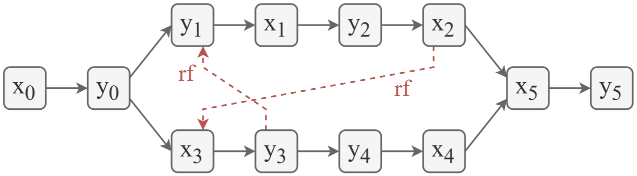
Techniques - Integer Difference Logic



- Each event has a *clock* integer variable
- A happens-before order implies an integer order

Advantages Easy encoding, any solver can be used

Techniques - Integer Difference Logic

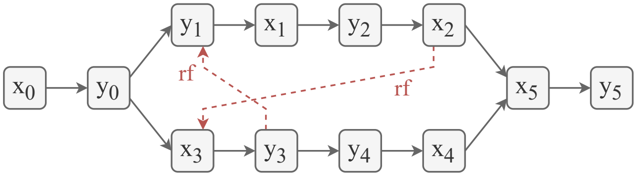


- Each event has a *clock* integer variable
- A happens-before order implies an integer order

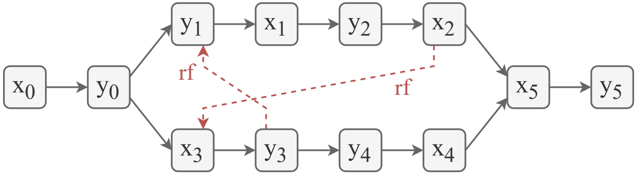
Advantages Easy encoding, any solver can be used

Disadvantages Overly specific, thus suboptimal

Techniques - Refinement Step-by-Step


$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[dllll, dashed, "rf"'] & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[ull, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] &
\end{tikzcd}$$

Techniques - Refinement Step-by-Step



- We track *happens-before* orders externally to the solver
- Any model is validated, and potentially *refined*
- The SMT solver does not propagate, we do

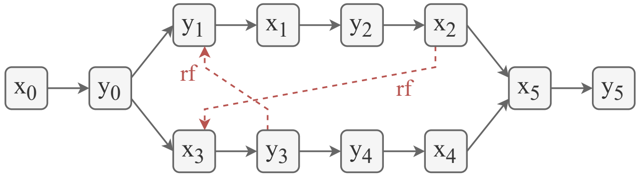
Techniques - Refinement Step-by-Step

$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[dddlll, phantom] \arrow[ddlll, dashed, "rf"] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[uul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

- We track *happens-before* orders externally to the solver
- Any model is validated, and potentially *refined*
- The SMT solver does not propagate, we do

Advantages Any solver can be used

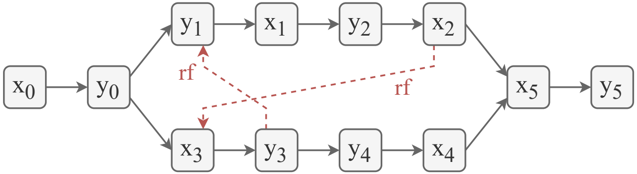
Techniques - Refinement Step-by-Step


$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[dlll, dashed, "rf"] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[ul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

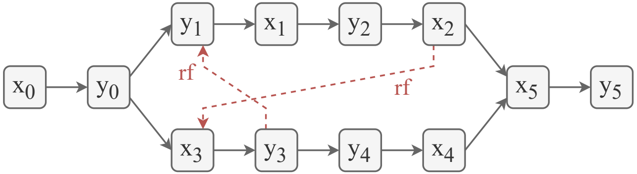
- We track *happens-before* orders externally to the solver
- Any model is validated, and potentially *refined*
- The SMT solver does not propagate, we do

Advantages	Any solver can be used
Disadvantages	Number of SMT queries increase

Techniques - User Propagator



Techniques - User Propagator



- We track *happens-before* orders internally in the solver
- Derived relations and conflicts are propagated
- A model is always consistent with *hb* acyclicity rules

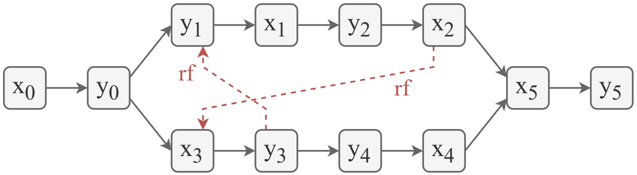
Techniques - User Propagator

$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[ddlll, dashed, "rf"] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[uul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

- We track *happens-before* orders internally in the solver
- Derived relations and conflicts are propagated
- A model is always consistent with *hb* acyclicity rules

Advantages Native solution

Techniques - User Propagator



- We track *happens-before* orders internally in the solver
- Derived relations and conflicts are propagated
- A model is always consistent with *hb* acyclicity rules

Advantages	Native solution
Disadvantages	Only certain SMT solvers are usable

Optimization: Automatic Conflict Avoidance

$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[ddlll, dashed, "rf"] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[uul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

Optimization: Automatic Conflict Avoidance

$$\begin{tikzcd}
 & & y_1 \arrow[r] & x_1 \arrow[r] & y_2 \arrow[r] & x_2 \arrow[dr] \arrow[ddlll, dashed, "rf"] & & \\
x_0 \arrow[r] & y_0 \arrow[ur] \arrow[dr] & & & & & x_5 \arrow[r] & y_5 \\
 & & x_3 \arrow[r] & y_3 \arrow[r] \arrow[uul, dashed, "rf"] & y_4 \arrow[r] & x_4 \arrow[ur] & &
\end{tikzcd}$$

- **Idea**: we enumerate some potential cycles explicitly
- This is sound (we don't overconstrain the state space)
- This helps the solvers (based on empirical results)

Experimental Evaluation

Research Question 1	How do the decision procedures compare?
Research Question 2	How much does the optimization help?
Research Question 3	How does Theta compare to other tools?
Research Question 4	Does solver choice influence performance?

Results - RQ1

Research Question 1

How do the decision procedures compare?

Results - RQ1

Research Question 1 How do the decision procedures compare?

- SV-Benchmarks tasks: 725 concurrent C programs
- 15 CPU-minute timeout, 15GB RAM

Results - RQ1

Research Question 1 How do the decision procedures compare?

- SV-Benchmarks tasks: 725 concurrent C programs
- 15 CPU-minute timeout, 15GB RAM

Solved tasks

	IDL	RFN	PROP
Solved	398	409	410
Time (s)	26000	4150	5770

Results - RQ1

Research Question 1 How do the decision procedures compare?

- SV-Benchmarks tasks: 725 concurrent C programs
- 15 CPU-minute timeout, 15GB RAM

Solved tasks

	IDL	RFN	PROP
Solved	398	409	410
Time (s)	26000	4150	5770

- Integer difference logic is worse and slowest
- Step-by-step refinement is quickest and almost best
- User propagation is best but not quickest

Results - RQ2

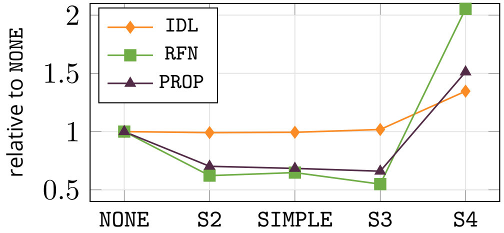
Research Question 2

How much does the optimization help?

Results - RQ2

Research Question 2

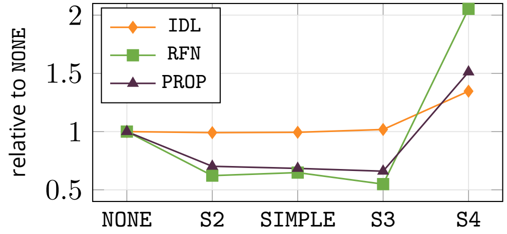
How much does the optimization help?



Results - RQ2

Research Question 2

How much does the optimization help?



- SN: Length of cycle hardcoded in SMT
- "Simple": 2, plus one derivation rule

Results - RQ3: Dartagnan

Research Question 3

How does Theta compare to other tools?

Results - RQ3: Dartagnan

Research Question 3 How does Theta compare to other tools?

	Theta (`complete`)			Dartagnan	
	`IDL`	`RFN`	`PROP`	`Eager`	`Lazy`
Solved/	398	409	410	456	457
filtered	398	409	410	434	433
Time (s)	26000	4150	5770	11200	6370

Results - RQ3: Deagle

Research Question 3 How does Theta compare to other tools?

	Theta (bounded)			Deagle
	IDL	RFN	PROP	
Solved/	538	554	553	623
filtered	538	554	553	577
Time (s)	35000	6020	8690	2020

- Bounded safety: unsound results

Results - RQ4

Research Question 4

Does solver choice influence performance?

Results - RQ4

Research Question 4 Does solver choice influence performance?

	IDL	RFN
Z3	538	544
cvc5	537	529
MathSAT	307	528
Princess	280	184

- Few new tasks solved

Theta

θtheta

`github.com/ftsrg/theta`

- Model checking framework originally designed for verification using abstraction and refinement (CEGAR)
- **Modular, extensible architecture**, supporting different verification engines (CEGAR, BMC, etc.).
- Emphasis on **reusability** across domains (control-flow automata, transition systems, statecharts, CHCs, ...).
- Supports **reusability** through regular updates on maven
- Full verifier packages via `Docker`, and Zenodo releases

Summary

Introduction

Context: what is the problem?
What do we do? Why do we care?

Verification approaches (existing & novel)
Optimization techniques (novel)

Techniques

Evaluation

Experiment and data analysis
Evaluation of impact

Theta as a Verifier
Theta as a Framework

Availability