Verifying Relational Properties using Trace Logic

Bernhard Gleiss, Gilles Barthe, Renate Eilers,
Pamina Georgiou, Laura Kovacs, Matteo Maffei

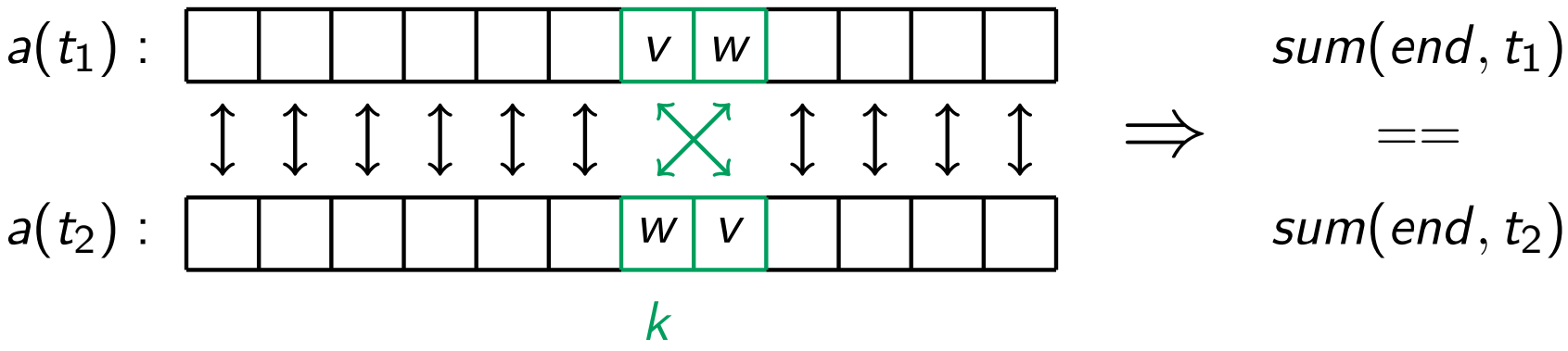
October 25, 2019

Motivating example

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

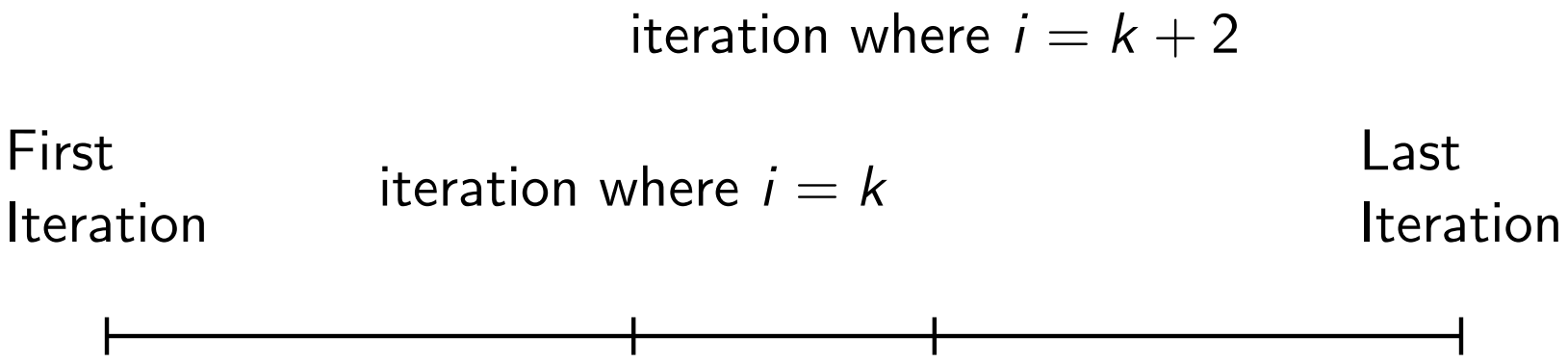
Motivating example

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

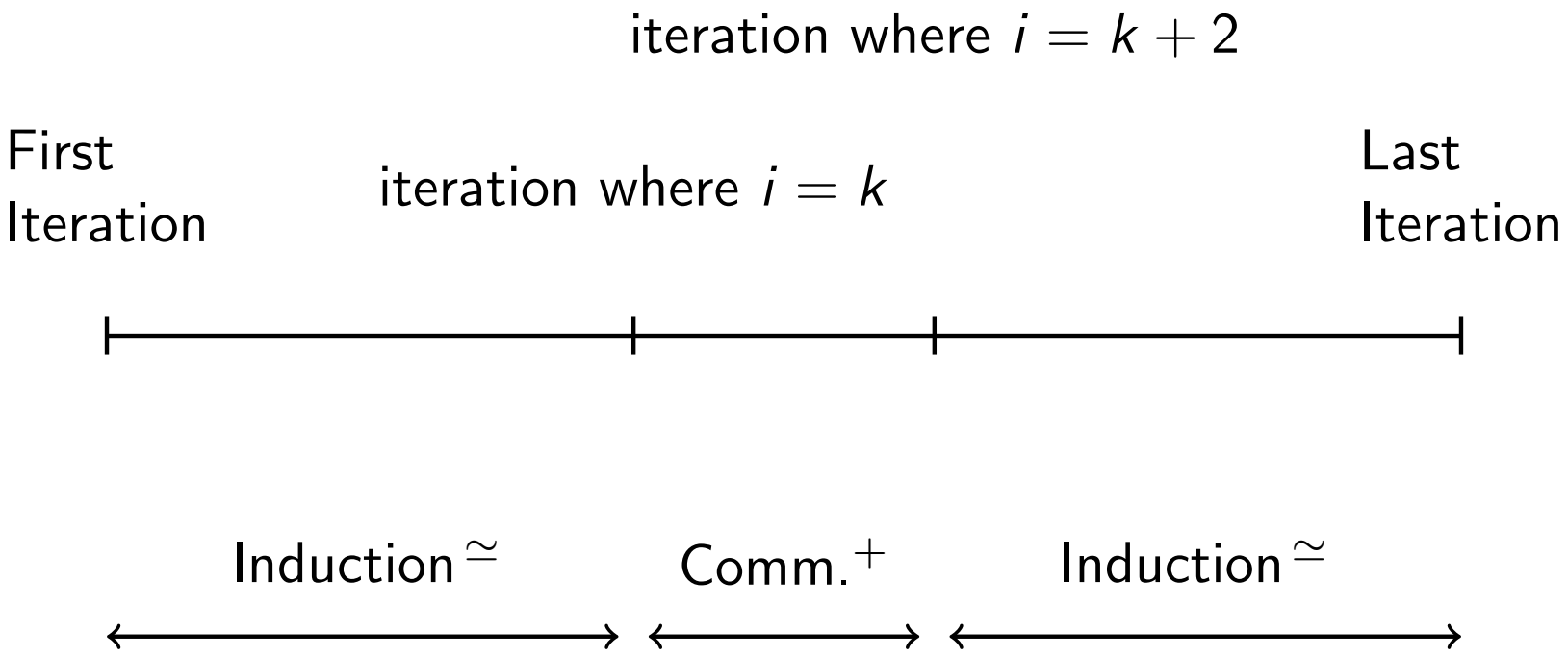


Motivating example - Human Proof

Motivating example - Human Proof



Motivating example - Human Proof



Focus

- (Software) programs containing loops and arrays
- Proving Relational Safety Properties
- Proving Correctness (instead of finding Counterexamples)

Remaining Talk - Outline

- Part 1: Language and Semantics - Trace Logic
- Part 2: Verification Approach - Vampire and Trace Lemmas
- Part 3: Extension - Relational Properties

Part 1: Language and Semantics - Trace Logic

Trace Logic

- full first-order logic over UFDTLIA
- explicit notion of time: able to refer to each timepoint of the execution uniquely, while preserving control flow structure
- can formulate induction directly in the language
- can denote parts of a loop and reason about those parts separately

Timepoints

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

l_4 $l_8(0)$ $l_6(s(0))$ $l_6(n_6)$ $l_6(it)$

Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$sum(l_8(0))$ $\quad$ $i(l_6(n_6))$ $\quad$ $a(l_2, pos)$

Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$sum(l_8(0))$ $\quad$ $i(l_6(n_6))$ $\quad$ $a(pos)$

Semantics in Trace Logic

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$$i(l_6(0)) \simeq 0$$

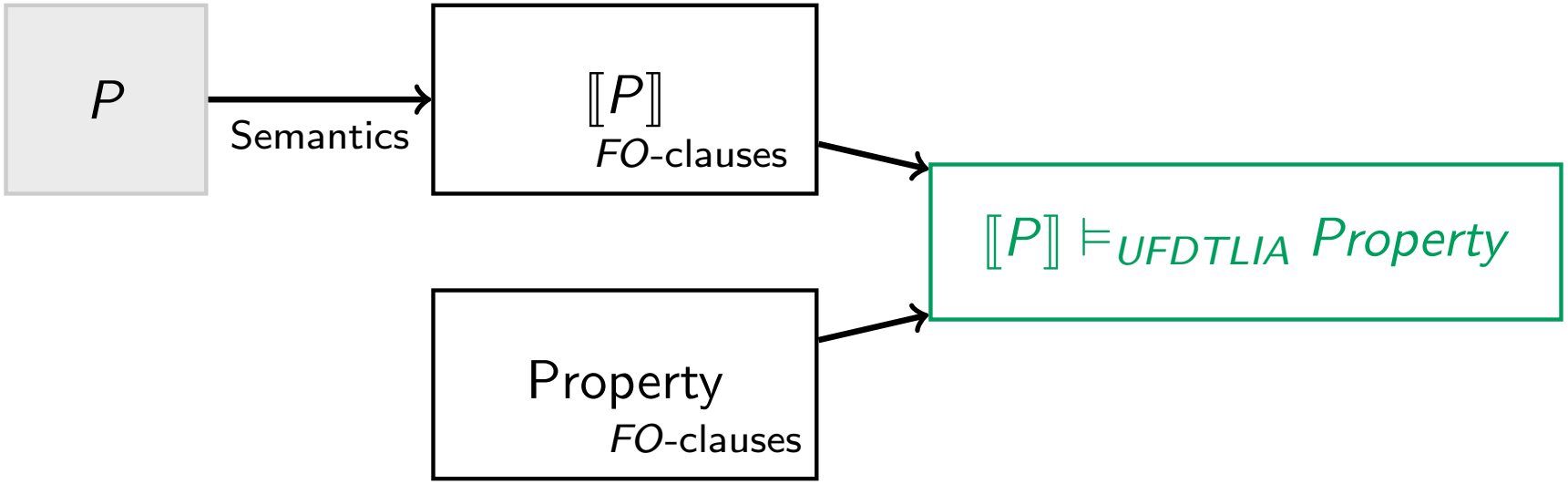
Semantics in Trace Logic

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

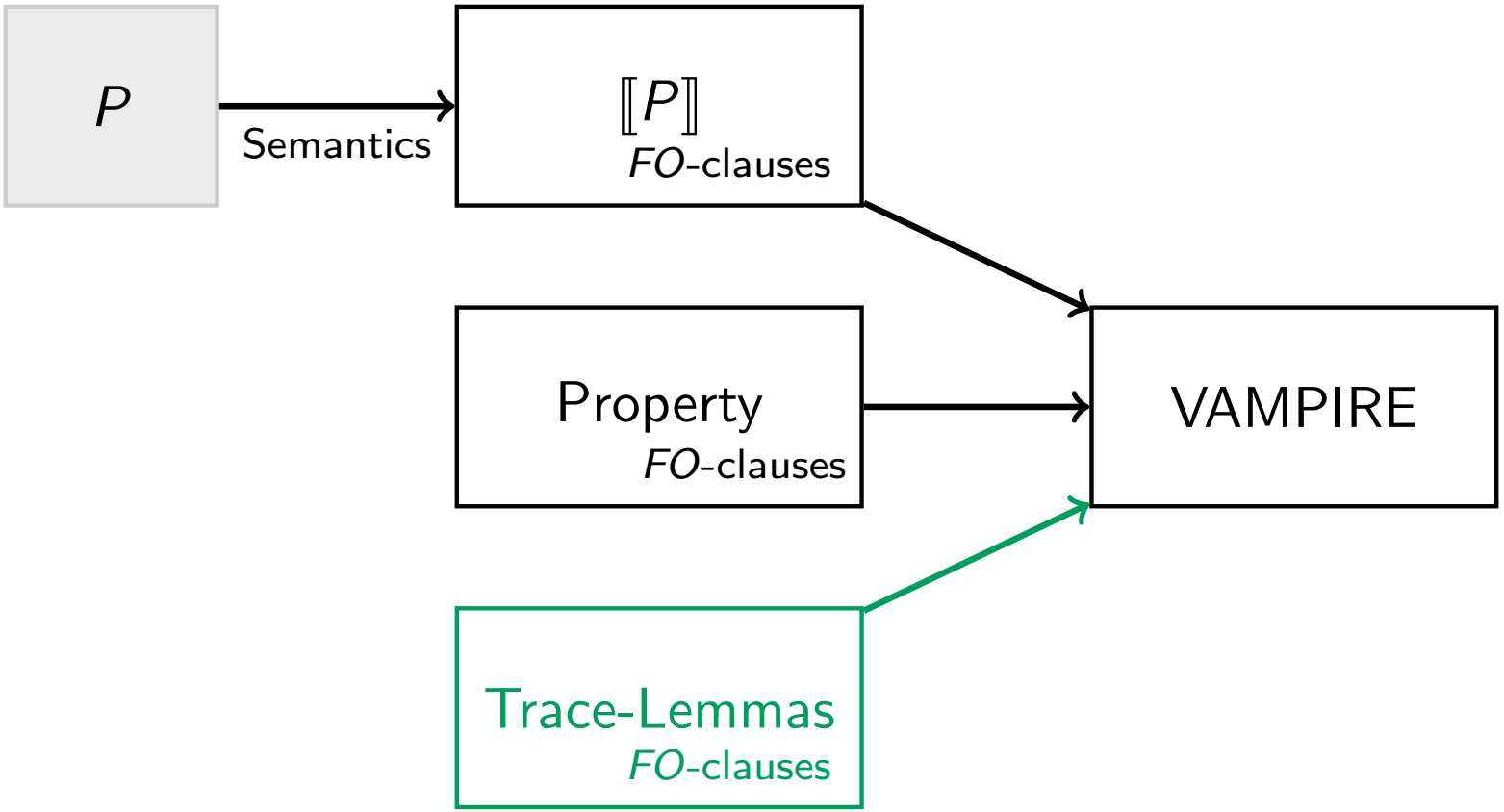
$$\forall it^{\mathbb{N}}.\ \big(it < n_6 \rightarrow i(l_6(s(it))) \simeq i(l_8(it)) + 1\big)$$

Part 2: Verification Approach - Vampire and Trace Lemmas

Workflow - Rapid



Workflow - Rapid



Trace Lemmas

- provide necessary inductive reasoning
- valid formulas, derivable from instances of the induction axiom scheme
- can't be automatically generated by state-of-the-art tools
- manually identified set of useful Trace Lemmas

Trace Lemmas - Example 1

"For an arbitrary interval: if the value of v stays the same in each step, then the value of v at the end is the same as the value of v at the beginning"

$$
\begin{aligned}
&\forall it_L^{\mathbb{N}}, \forall it_R^{\mathbb{N}}.(\\
&\qquad \forall it^{\mathbb{N}}.\big(it_L \leq it < it_R \rightarrow v(l_6(it)) \simeq v(l_6(s(it)))\big) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \\
&\qquad\qquad\qquad\qquad \forall it^{\mathbb{N}}.v(l_6(it_L)) \simeq v(l_6(it_R)) \\
&)
\end{aligned}
$$

Trace Lemmas - Example 2

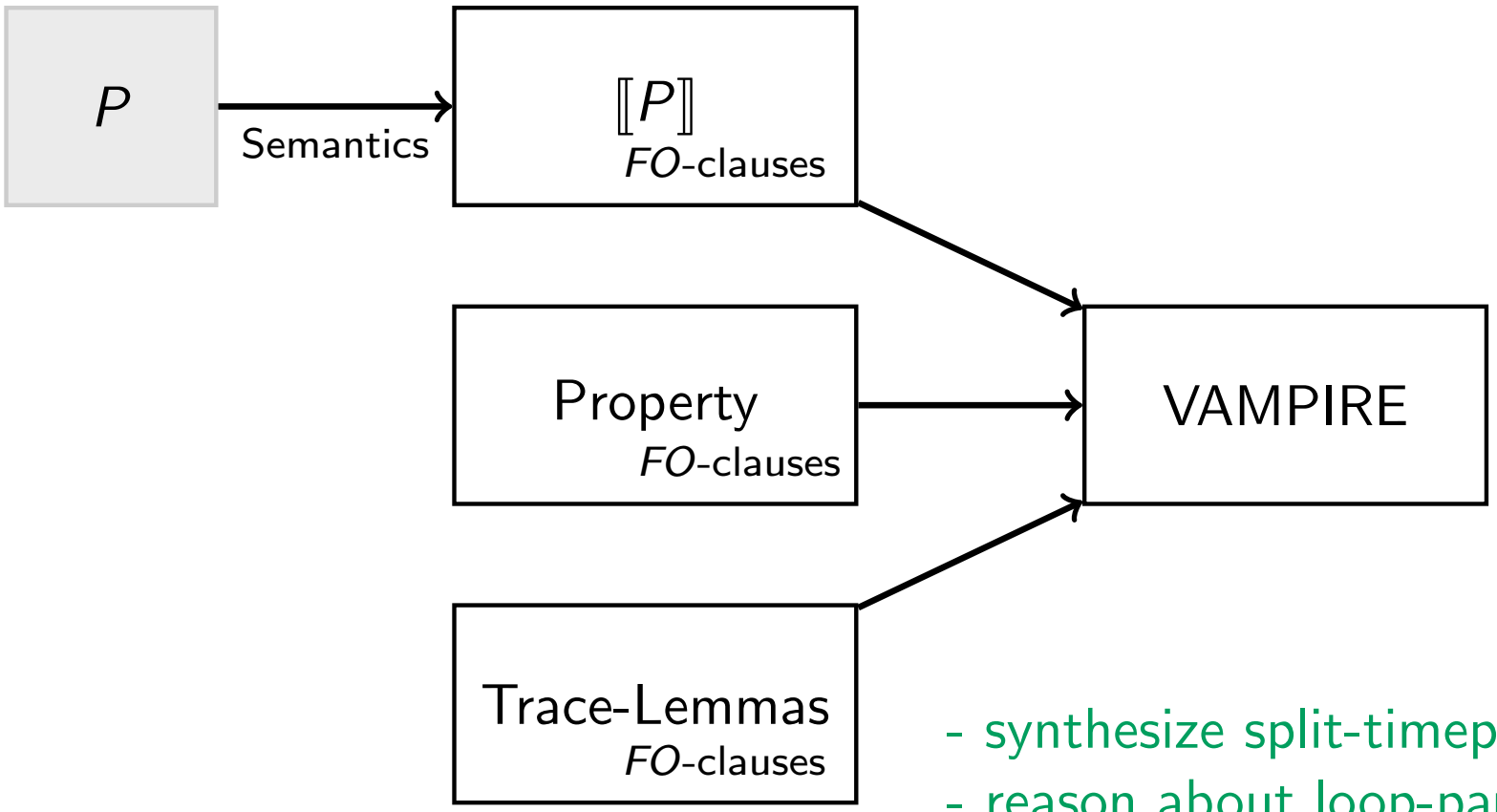
Intermediate Value Theorem: "If $i \leq v$ at the beginning and $i > v$ at the end and if i is incremented by 1 in each iteration, then there exists an iteration, where $i = v$."

$$
\begin{aligned}
&\forall v^{\mathbb{I}}.(\\
&\quad (\\
&\qquad i(l_6(0)) \leq v \quad \wedge \\
&\qquad i(l_6(n_6)) > v \quad \wedge \\
&\qquad \forall it^{\mathbb{N}}.\big(it < n \rightarrow i(l_6(s(it))) \simeq i(l_6(it)) + 1\big) \\
&\quad) \\
&\quad \rightarrow \exists it'_{\mathbb{N}}.i(l_6(it')) \simeq v \\
&)
\end{aligned}
$$

Trace Lemmas

- can be instantiated to parts of the loop
- can feature existential quantification over iterations
- can feature quantifier alternations
- can not be synthesized automatically by state-of-the-art techniques

Workflow - Rapid



- synthesize split-timepoints
- reason about loop-parts sep.
- perform interleaved

Part 3: Extension to Relational Properties

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

n_6 $\quad$ $sum(l_8(0))$ $\quad$ $i(l_6(n_8))$

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$n_6(t_1)$ $\quad$ $sum(l_8(0), t_1)$ $\quad$ $i(l_6(n_8), t_2)$

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$$i(l_6(0)) \simeq 0$$

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$$\forall tr^{\mathbb{T}}.i(l_6(0), tr) \simeq 0$$

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$$\forall it^{\mathbb{N}}.\ \big(it < n_6 \rightarrow i(l_6(s(it))) \simeq i(l_8(it)) + 1\big)$$

Extension - Timepoints and Program Variables

```
1    func main() {
2        const Int[] a;
3
4        Int sum = 0;
5
6        for (Int i=0; i < a.length; ++i)
7        {
8            sum += a[i];
9        }
10   }
```

$$\forall tr^{\mathbb{T}}.\forall it^{\mathbb{N}}.\ \left(it < n_6(tr) \rightarrow i(l_6(s(it)), tr) \simeq i(l_8(it), tr) + 1\right)$$

Extension - Trace Lemmas

sum has the same value in both traces in iteration *it*:

$$Eq_{sum}(it) := sum(l_6(it), t1) \simeq sum(l_6(it), t_2).$$

Bounded induction with induction hypothesis $Eq_{sum}(it)$:

$$
\begin{array}{l}
\forall itL^{\mathbb{N}}, itR^{\mathbb{N}}. \\
(\\
\quad (\\
\qquad Eq_{sum}(itL) \quad \wedge \\
\qquad \forall it^{\mathbb{N}}.((itL \leq it < itR \wedge Eq_{sum}(it)) \rightarrow Eq_{sum}(s(it))) \\
\quad) \\
\quad \rightarrow Eq_{sum}(itR) \\
)
\end{array}
$$

Benchmarks

- 27 challenging benchmarks from security applications
- 2-safety properties: non-interference and sensitivity
- 60 second timeout

Benchmarks	Vampire				CVC4	Z3
	S	S+A	F	F+A		
1-hw-equal-arrays	✓	✓	-	✓	✓	✓
2-hw-last-position-swapped	-	✓	-	-	✓	✓
3-hw-swap-and-two-arrays	-	✓	-	-	-	-
4-hw-swap-in-array-lemma	-	✓	-	-	-	-
4-hw-swap-in-array-full	-	✓	-	-	-	-
1-ni-assign-to-high	✓	✓	✓	✓	✓	✓
2-ni-branch-on-high-twice	✓	✓	✓	✓	✓	✓
3-ni-high-guard-equal-branches	✓	✓	✓	✓	✓	✓
4-ni-branch-on-high-twice-prop2	✓	✓	-	-	✓	✓
5-ni-temp-impl-flow	-	-	✓	✓	✓	✓
6-ni-branch-assign-equal-val	-	-	✓	✓	✓	✓
7-ni-explicit-flow	✓	✓	✓	✓	✓	✓
8-ni-explicit-flow-while	✓	✓	-	✓	✓	✓
9-ni-equal-output	✓	-	-	-	-	✓
10-ni-rsa-exponentiation	✓	✓	✓	✓	✓	-
1-sens-equal-sums	✓	✓	✓	✓	✓	✓
2-sens-equal-sums-two-arrays	✓	✓	✓	✓	-	-
3-sens-abs-diff-up-to-k	-	-	-	-	✓	✓
4-sens-abs-diff-up-to-k-two-arrays	-	-	-	-	-	-
5-sens-two-arrays-equal-k	✓	✓	✓	✓	-	-
6-sens-diff-up-to-explicit-k	✓	✓	✓	✓	-	-
7-sens-diff-up-to-explicit-k-sum	-	-	✓	✓	-	-
8-sens-explicit-swap	-	-	✓	✓	-	-
9-sens-explicit-swap-prop2	-		✓	✓	-	-
10-sens-equal-k	✓	✓	✓	✓	-	-
11-sens-equal-k-twice	✓	✓	✓	✓	-	-
12-sens-diff-up-to-forall-k	-	-	✓	✓	✓	-
Total VAMPIRE	15	18	17	19		
Unique VAMPIRE	1	4	0	0		
Total	25				14	13

Conclusion

- Trace Logic Language and Semantics
- Verification Approach - Vampire and Trace Lemmas
- Application to Relational Verification

Extra slides

Background Theory

- Full first-order logic with equality and uninterpreted functions
- Iterations - Datatype $(0, s, p, <)$ (no arithmetic!)
- Timepoints - Uninterpreted Sort
- Values of program variables - Integers

Motivating example - Property in Trace Logic

$$
\begin{aligned}
&Swapped_a(k) := \\
&\qquad\qquad 0 \leq k < k+1 < a.length \\
&\wedge \qquad \forall pos^{\mathbb{I}}.((pos \not\simeq k \wedge pos \not\simeq k{+}1) \rightarrow \\
&\qquad\qquad\quad a(pos, t_1) \simeq a(pos, t_2)) \\
&\wedge \qquad a(k, t_1) \simeq a(k{+}1, t_2) \\
&\wedge \qquad a(k, t_2) \simeq a(k{+}1, t_1)
\end{aligned}
$$

$$\forall k^{\mathbb{I}}.\big(Swapped_a(k) \rightarrow sum(end, t_1) \simeq sum(end, t_2)\big)$$

Rapid Tool

Available at:

https://github.com/gleiss/rapid

Theory reasoning - Blowup

new: 40588. less(-4,0) — less(-6,4)
new: 40589. less(-4,-1) — less(-4,4)
new: 40590. less(-4,-1) — less(-3,4)
new: 40591. less(-4,-1) — less(-2,4)
new: 40592. less(-4,-1) — 0 = 4 — less(0,4)
new: 40593. less(-4,-1) — less(4,0)
new: 40594. less(-4,2) — less(-2,4)
new: 40595. less(-4,2) — less(-1,4)
new: 40596. less(-4,2) — less(0,4)
new: 40597. less(-4,2) — less(4,3)
new: 40598. less(-4,2) — less(1,4)
new: 40599. less(-4,-2) — less(-4,4)
new: 40600. less(-4,-2) — less(-3,4)
new: 40601. less(-4,-3) — less(-4,4)
new: 40602. less(-4,3) — less(-3,4)
new: 40603. less(-4,3) — less(-1,4)
new: 40604. less(-4,3) — less(-2,4)
new: 40605. less(-4,3) — less(0,4)
new: 40606. less(-4,3) — less(1,4)
new: 40607. less(-4,3) — less(2,4)
new: 40608. less(-4,4) — less(-3,4)
new: 40609. less(-4,4) — less(-2,4)
new: 40610. less(-4,4) — less(2,4)
new: 40611. less(-4,4) — less(1,4)
new: 40612. less(-4,4) — less(0,4)
new: 40613. less(-4,4) — less(-1,4)
new: 40614. less(-4,5) — less(-1,4)
new: 40615. less(-4,5) — less(1,4)
new: 40616. less(-4,5) — less(0,4)