Verifying Array Manipulating Programs by Tiling

Divyesh Unadkat, IIT B & TCS

guided by
Prof. Supratik Chakraborty, IIT B
Prof. Ashutosh Gupta, TIFR

7 December, 2017

2nd SAT+SMT School
Infosys Campus, Mysore

Motivating Example

```
void foo(int A[], int N) {
  for (int i = 0; i < N; i++) {
    if(!(i==0 || i==N-1)) {
      if (A[i] < THRESH) {
        A[i+1] = A[i] + 1;
        A[i] = A[i-1];
      }
    } else {
      A[i] = THRESH;
    }
  }
  assert(for i in 0..N-1, A[i]>=THRESH);
}
```

Motivating Example

```
void foo(int A[], int N) {
  for (int i = 0; i < N; i++) {
    if(!(i==0 || i==N-1)) {
      if (A[i] < 5) {
        A[i+1] = A[i] + 1;
        A[i] = A[i-1];
      }
    } else {
      A[i] = 5;
    }
  }
  assert(for k in 0..N-1, A[k]>=5);
}
```

Motivating Example

```
void foo(int A[], int N) {
  for (int i = 0; i < N; i++) {
    if(!(i==0 || i==N-1)) {
      if (A[i] < 5) {
        A[i+1] = A[i] + 1;
        A[i] = A[i-1];
      }
    } else {
      A[i] = 5;
    }
  }
  assert(for k in 0..N-1, A[k]>=5);
}
```

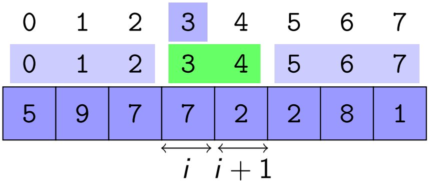
Initial array

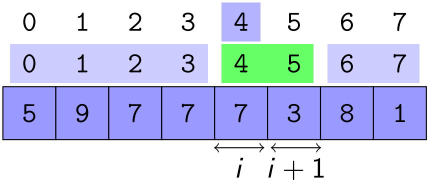
0	1	2	3	4	5	6	7	Loop Counter
0	1	2	3	4	5	6	7	Indices
5	9	7	1	9	2	8	1	Cell Contents

$\neg\forall k . a[k] \geq 5$

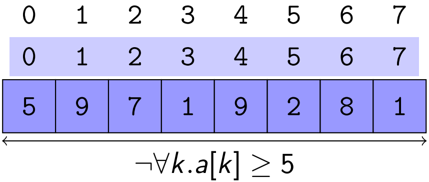
Motivating Example

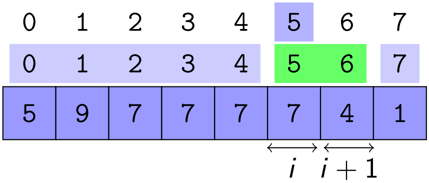
```
void foo(int A[], int N) {
  for (int i = 0; i < N; i++) {
    if(!(i==0 || i==N-1)) {
      if (A[i] < 5) {
        A[i+1] = A[i] + 1;
        A[i] = A[i-1];
      }
    } else {
      A[i] = 5;
    }
  }
  assert(for k in 0..N-1, A[k]>=5);
}
```





Initial array

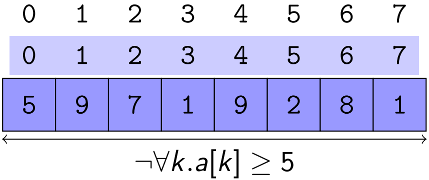


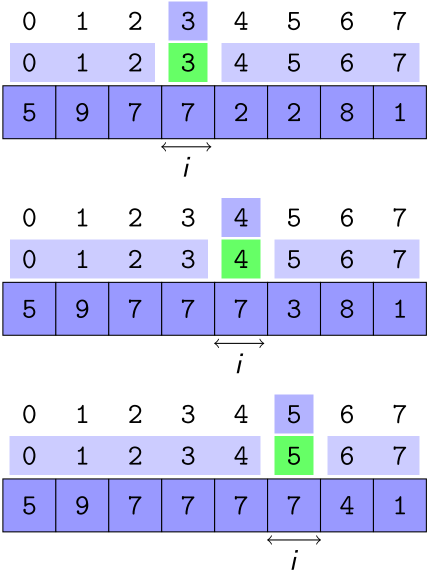


Motivating Example

```
void foo(int A[], int N) {
  for (int i = 0; i < N; i++) {
    if(!(i==0 || i==N-1)) {
      if (A[i] < 5) {
        A[i+1] = A[i] + 1;
        A[i] = A[i-1];
      }
    } else {
      A[i] = 5;
    }
  }
  assert(for k in 0..N-1, A[k]>=5);
}
```

Initial array





Tiling

- $\mathsf{Tile} : \mathsf{LoopCounter} \times \mathsf{Indices} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

Tiling

- Tile : LoopCounter $\times$ Indices $\rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

- $P_1(i, j) := i \leq j \leq i + 1$
- $P_2(i, j) := j == i$

Tiling

- Tile : LoopCounter $\times$ Indices $\rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

- $P_1(i, j) := i \leq j \leq i + 1$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[5] \not\geq 5$

- $P_2(i, j) := j == i$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[4] \geq 5$

Tiling

- Tile : LoopCounter $\times$ Indices $\rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

- $P_1(i,j) := i \leq j \leq i+1$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[5] \not\geq 5$

- $P_2(i,j) := j == i$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[4] \geq 5$

- Truth of the assertion wrt tile changes in the next iteration

- Truth of the assertion wrt tile doesn't change in the future

Tiling

- Tile : LoopCounter $\times$ Indices $\rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

- $P_1(i, j) := i \leq j \leq i + 1$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[5] \not\geq 5$

- Truth of the assertion wrt tile changes in the next iteration
- May miss update to some indices

- $P_2(i, j) := j == i$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[4] \geq 5$

- Truth of the assertion wrt tile doesn't change in the future
- Doesn't miss updates to any index

Tiling

- Tile : LoopCounter $\times$ Indices $\rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ for loop L

- $P_1(i, j) := i \leq j \leq i + 1$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[5] \not\geq 5$

- Truth of the assertion wrt tile changes in the next iteration
- May miss update to some indices

- $P_2(i, j) := j == i$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	7	3	8	1

$a[4] \geq 5$

- Truth of the assertion wrt tile doesn't change in the future
- Doesn't miss updates to any index

Finding the *right* tile is a challenge!

Proving Assertions using Tiles

If following conditions hold on the tile, we have proven the property

T1: Covers range

T2: Sliced post-condition holds inductively

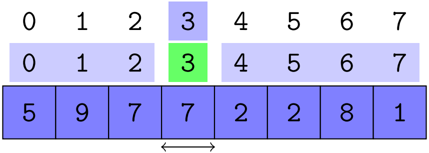
T3: Non-interference across tiles

T1: Covers Range

Indices of interest must be covered by some *tile*

T1: Covers Range

Indices of interest must be covered by some *tile*



$$\mathsf{Tile}(i, j) := (j == i)$$

- $\mathsf{Post} \triangleq \forall i(\Phi(i) \implies \Psi(\mathsf{A}, i))$

T1: Covers Range

Indices of interest must be covered by some *tile*

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	2	2	8	1

$$\mathrm{Tile}(i, j) := (j == i)$$

- $\mathrm{Post} \triangleq \forall i(\Phi(i) \implies \Psi(A, i))$
- $\eta_1 \;\equiv\; \forall j\,(\Phi(j) \implies \exists i\,(\mathrm{Tile}(i, j)))$, $\eta_2 \;\equiv\; \forall i, j\,(\mathrm{Tile}(i, j) \implies \Phi(j))$

T1: Covers Range

Indices of interest must be covered by some *tile*

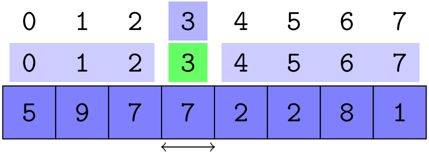
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
5	9	7	7	2	2	8	1

$$\text{Tile}(i, j) := (j == i)$$

- $\text{Post} \triangleq \forall i(\Phi(i) \implies \Psi(\text{A}, i))$
- $\eta_1 \equiv \forall j\,(\Phi(j) \implies \exists i\,(\text{Tile}(i,j)))$, $\eta_2 \equiv \forall i, j\,(\text{Tile}(i,j) \implies \Phi(j))$
- Validity of $\eta_1 \wedge \eta_2$ ensures T1

T1: Covers Range

Indices of interest must be covered by some *tile*



$\mathsf{Tile}(i,j) := (j == i)$

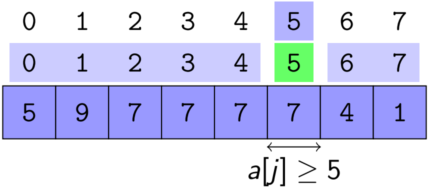
- Post $\triangleq \forall i(\Phi(i) \implies \Psi(\mathsf{A}, i))$
- $\eta_1 \equiv \forall j\,(\Phi(j) \implies \exists i\,(\mathsf{Tile}(i,j)))$, $\eta_2 \equiv \forall i,j\,(\mathsf{Tile}(i,j) \implies \Phi(j))$
- Validity of $\eta_1 \wedge \eta_2$ ensures T1
- Involves a quantifier alternation; can be handled by SMT solvers

T2: Sliced Post-condition holds Inductively

Post-condition wrt indices in the i^{th} tile holds inductively

T2: Sliced Post-condition holds Inductively

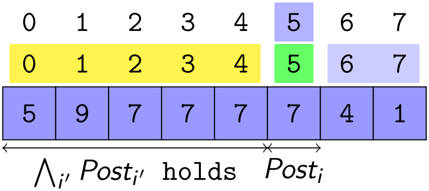
Post-condition wrt indices in the i^{th} tile holds inductively



- Post $\triangleq \forall i (\Phi(i) \implies \Psi(\mathsf{A}, i))$
- Sliced post-condition for the i^{th} tile
  $\text{Post}_i \triangleq \forall j\, (\text{Tile}(i,j) \wedge \Phi(j) \implies \Psi(\mathsf{A}, j))$

T2: Sliced Post-condition holds Inductively

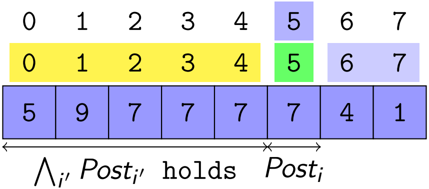
Post-condition wrt indices in the i^{th} tile holds inductively



- Post $\triangleq \forall i(\Phi(i) \implies \Psi(A, i))$
- Sliced post-condition for the i^{th} tile
  $\text{Post}_i \triangleq \forall j\,(\text{Tile}(i, j) \wedge \Phi(j) \implies \Psi(A, j))$

T2: Sliced Post-condition holds Inductively

Post-condition wrt indices in the i^{th} tile holds inductively



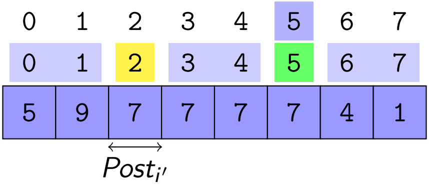
- Post $\triangleq \forall i(\Phi(i) \implies \Psi(A, i))$
- Sliced post-condition for the i^{th} tile
  $\mathsf{Post}_i \triangleq \forall j\,(\mathsf{Tile}(i, j) \wedge \Phi(j) \implies \Psi(A, j))$
- $\{\mathsf{Inv} \wedge \bigwedge_{i':0\leq i'<i} \mathsf{Post}_{i'}\}\ \mathrm{L_{body}}\ \{\mathsf{Inv} \wedge \mathsf{Post}_i\}$ must be valid

T3: Non-interference across Tiles

No iteration $i > i'$ interferes with the truth of $Post_{i'}$, once established

T3: Non-interference across Tiles

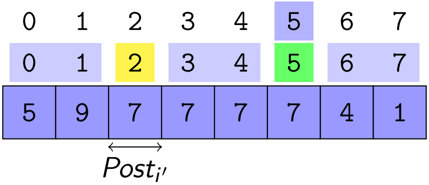
No iteration $i > i'$ interferes with the truth of $Post_{i'}$, once established



- $\mathsf{Post} \triangleq \forall i(\Phi(i) \implies \Psi(\mathsf{A}, i))$
- Sliced post-condition for the i'^{th} tile
  $\mathsf{Post}_{i'} \triangleq \forall j' \, (\mathsf{Tile}(i', j') \wedge \Phi(j') \implies \Psi(\mathsf{A}, j'))$

T3: Non-interference across Tiles

No iteration $i > i'$ interferes with the truth of $Post_{i'}$, once established



- $\mathsf{Post} \triangleq \forall i(\Phi(i) \implies \Psi(\mathsf{A}, i))$
- Sliced post-condition for the i'^{th} tile
  $\mathsf{Post}_{i'} \triangleq \forall j' \, (\mathsf{Tile}(i', j') \wedge \Phi(j') \implies \Psi(\mathsf{A}, j'))$
- $\{\mathsf{Inv} \wedge (0 \leq i' < i) \wedge \mathsf{Post}_{i'}\} \mathsf{L}_{\mathrm{body}} \{\mathsf{Post}_{i'}\}$ must be valid

Inductive Compositional Reasoning

- Inductive Reasoning
  - T2 Sliced post-condition holds for each iteration
- Compositional Reasoning
  - T3 Truth of sliced post-condition once established is not altered subsequently
  - T1 Tiles cover the entire range of array indices of interest

Sequentially Composed Loops

```
void copynswap(int N)
{
 int i, tmp;
 int a[], b[], acopy[];

 for (i = 0; i < N; i++) {
   acopy[i] = a[i];
 }

 for (i = 0; i < N; i++) {
   tmp = a[i];
   a[i] = b[i];
   b[i] = tmp;
 }

 for (i = 0; i < N; i++) {
   assert(b[i] == acopy[i]);
 }
}
```

Sequentially Composed Loops

```
void copynswap(int N)
{
 int i, tmp;
 int a[], b[], acopy[];

 for (i = 0; i < N; i++) {
   acopy[i] = a[i];
 }

 for (i = 0; i < N; i++) {
   tmp = a[i];
   a[i] = b[i];
   b[i] = tmp;
 }

 for (i = 0; i < N; i++) {
   assert(b[i] == acopy[i]);
 }
}
```

Mid-conditions

- Invariants between sequentially composed loops
- Hard to generate precise invariants
- Identify *candidate* mid-conditions using annotation assistants

Sequentially Composed Loops

```
void copynswap(int N)
{
 int i, tmp;
 int a[], b[], acopy[];

 for (i = 0; i < N; i++) {
   acopy[i] = a[i];
 }

 for (i = 0; i < N; i++) {
   tmp = a[i];
   a[i] = b[i];
   b[i] = tmp;
 }

 for (i = 0; i < N; i++) {
   assert(b[i] == acopy[i]);
 }
}
```

Mid-conditions

- Invariants between sequentially composed loops
- Hard to generate precise invariants
- Identify *candidate* mid-conditions using annotation assistants

Candidate mid-conditions

- $\forall i(a[i] = acopy[i])$
- $\forall i(a[i] \neq b[i])$

Sequentially Composed Loops

```
void copynswap(int N)
{
 int i, tmp;
 int a[], b[], acopy[];

 for (i = 0; i < N; i++) {
   acopy[i] = a[i];
 }

 for (i = 0; i < N; i++) {
   tmp = a[i];
   a[i] = b[i];
   b[i] = tmp;
 }

 for (i = 0; i < N; i++) {
   assert(b[i] == acopy[i]);
 }
}
```

Mid-conditions

- Invariants between sequentially composed loops
- Hard to generate precise invariants
- Identify *candidate* mid-conditions using annotation assistants
- *Prove* them using Tiling

Candidate mid-conditions

- $\forall i(a[i] = acopy[i])$
- $\forall i(a[i] \neq b[i])$

Proved mid-conditions

- $\forall i(a[i] = acopy[i])$

Tiler Tool Diagram

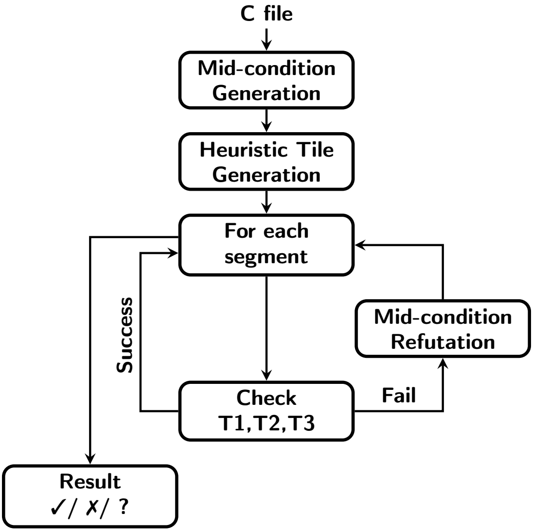


Figure : Tiler Tool Diagram

Tiler Benchmarking

- 60 benchmarks from industry and academia
- Performance compared with tools
  - SMACK+Corral - Bounded model checker
  - Booster - Acceleration based verification for arrays
  - Vaphor - Distinguished cell abstraction for arrays
- Memory limit - 1GB
- Time limit - 900s

Tiler in Action

Benchmark	#L	**Tiler**	S+C	Booster	Vaphor
cpynrev.c	2	✓3.8	†	✓3.1	✓5.4
cpynswp.c	2	✓4.2	†	✓12.4	✓1.38
cpynswp2.c	3	✓10.2	†	✓198	✓7.2*
maxinarr.c	1	✓0.51	†	✓0.01	✓0.11
mininarr.c	1	✓0.53	†	✓0.02	✓0.13
poly1.c	1	TO	†	✓15.7	TO
poly2.c	2	? 6.44	†	? 19.5	TO
tcpy.c	1	? 0.65	†	TO	✓25.1
rew.c	1	✓0.48	†	✓0.01	TO
skipped.c	1	✓1.24	†	TO	TO
rewrev.c	1	✓0.39	†	TO	TO
pr4.c	1	✓0.68	†	TO	TO
pr5.c	1	✓1.32	†	TO	TO
pnr4.c	1	✓0.86	†	TO	TO
pnr5.c	1	✓1.98	†	TO	TO
mbpr4.c	4	✓12.75	†	TO	TO
mbpr5.c	5	✓18.08	†	TO	TO
nr4.c	1-1	✓2.43*	†	TO	TO
nr5.c	1-1	✓2.90*	†	TO	TO
copy9u.c	9	✗0.16	✗4.48	✗0.44	✗30.8
skippedu.c	1	✗0.81	✗2.94	✗0.02	TO

Tiler in Action

Benchmark	#L	**Tiler**	S+C	Booster	Vaphor
cpynrev.c	2	✓3.8	†	✓3.1	✓5.4
cpynswp.c	2	✓4.2	†	✓12.4	✓1.38
cpynswp2.c	3	✓10.2	†	✓198	✓7.2*
maxinarr.c	1	✓0.51	†	✓0.01	✓0.11
mininarr.c	1	✓0.53	†	✓0.02	✓0.13
poly1.c	1	TO	†	✓15.7	TO
poly2.c	2	? 6.44	†	? 19.5	TO
tcpy.c	1	? 0.65	†	TO	✓25.1
rew.c	1	✓0.48	†	✓0.01	TO
skipped.c	1	✓1.24	†	TO	TO
rewrev.c	1	✓0.39	†	TO	TO
pr4.c	1	✓0.68	†	TO	TO
pr5.c	1	✓1.32	†	TO	TO
pnr4.c	1	✓0.86	†	TO	TO
pnr5.c	1	✓1.98	†	TO	TO
mbpr4.c	4	✓12.75	†	TO	TO
mbpr5.c	5	✓18.08	†	TO	TO
nr4.c	1-1	✓2.43*	†	TO	TO
nr5.c	1-1	✓2.90*	†	TO	TO
copy9u.c	9	✗0.16	✗4.48	✗0.44	✗30.8
skippedu.c	1	✗0.81	✗2.94	✗0.02	TO

Tiler in Action

Benchmark	#L	**Tiler**	S+C	Booster	Vaphor
cpynrev.c	2	✓3.8	†	✓3.1	✓5.4
cpynswp.c	2	✓4.2	†	✓12.4	✓1.38
cpynswp2.c	3	✓10.2	†	✓198	✓7.2*
maxinarr.c	1	✓0.51	†	✓0.01	✓0.11
mininarr.c	1	✓0.53	†	✓0.02	✓0.13
poly1.c	1	TO	†	✓15.7	TO
poly2.c	2	? 6.44	†	? 19.5	TO
tcpy.c	1	? 0.65	†	TO	✓25.1
rew.c	1	✓0.48	†	✓0.01	TO
skipped.c	1	✓1.24	†	TO	TO
rewrev.c	1	✓0.39	†	TO	TO
pr4.c	1	✓0.68	†	TO	TO
pr5.c	1	✓1.32	†	TO	TO
pnr4.c	1	✓0.86	†	TO	TO
pnr5.c	1	✓1.98	†	TO	TO
mbpr4.c	4	✓12.75	†	TO	TO
mbpr5.c	5	✓18.08	†	TO	TO
nr4.c	1-1	✓2.43*	†	TO	TO
nr5.c	1-1	✓2.90*	†	TO	TO
copy9u.c	9	✗0.16	✗4.48	✗0.44	✗30.8
skippedu.c	1	✗0.81	✗2.94	✗0.02	TO

Tiler in Action

Benchmark	#L	**Tiler**	S+C	Booster	Vaphor
cpynrev.c	2	✓3.8	†	✓3.1	✓5.4
cpynswp.c	2	✓4.2	†	✓12.4	✓1.38
cpynswp2.c	3	✓10.2	†	✓198	✓7.2*
maxinarr.c	1	✓0.51	†	✓0.01	✓0.11
mininarr.c	1	✓0.53	†	✓0.02	✓0.13
poly1.c	1	TO	†	✓15.7	TO
poly2.c	2	? 6.44	†	? 19.5	TO
tcpy.c	1	? 0.65	†	TO	✓25.1
rew.c	1	✓0.48	†	✓0.01	TO
skipped.c	1	✓1.24	†	TO	TO
rewrev.c	1	✓0.39	†	TO	TO
pr4.c	1	✓0.68	†	TO	TO
pr5.c	1	✓1.32	†	TO	TO
pnr4.c	1	✓0.86	†	TO	TO
pnr5.c	1	✓1.98	†	TO	TO
mbpr4.c	4	✓12.75	†	TO	TO
mbpr5.c	5	✓18.08	†	TO	TO
nr4.c	1-1	✓2.43*	†	TO	TO
nr5.c	1-1	✓2.90*	†	TO	TO
copy9u.c	9	✗0.16	✗4.48	✗0.44	✗30.8
skippedu.c	1	✗0.81	✗2.94	✗0.02	TO

Tiler in Action

Benchmark	#L	**Tiler**	S+C	Booster	Vaphor
cpynrev.c	2	✓3.8	†	✓3.1	✓5.4
cpynswp.c	2	✓4.2	†	✓12.4	✓1.38
cpynswp2.c	3	✓10.2	†	✓198	✓7.2*
maxinarr.c	1	✓0.51	†	✓0.01	✓0.11
mininarr.c	1	✓0.53	†	✓0.02	✓0.13
poly1.c	1	TO	†	✓15.7	TO
poly2.c	2	? 6.44	†	? 19.5	TO
tcpy.c	1	? 0.65	†	TO	✓25.1
rew.c	1	✓0.48	†	✓0.01	TO
skipped.c	1	✓1.24	†	TO	TO
rewrev.c	1	✓0.39	†	TO	TO
pr4.c	1	✓0.68	†	TO	TO
pr5.c	1	✓1.32	†	TO	TO
pnr4.c	1	✓0.86	†	TO	TO
pnr5.c	1	✓1.98	†	TO	TO
mbpr4.c	4	✓12.75	†	TO	TO
mbpr5.c	5	✓18.08	†	TO	TO
nr4.c	1-1	✓2.43*	†	TO	TO
nr5.c	1-1	✓2.90*	†	TO	TO
copy9u.c	9	✗0.16	✗4.48	✗0.44	✗30.8
skippedu.c	1	✗0.81	✗2.94	✗0.02	TO

Tiles in Benchmarks

- **Reverse** the contents of the array
  - $\text{Tile}(i,j) := j == N - i - 1$
- A **bunch** of indices updated in a loop
  - $\text{Tile}(i,j) := 2 * i - 2 \leq j < 2 * i$
  - $\text{Tile}(i,j) := 3 * i - 3 \leq j < 3 * i$
  - $\text{Tile}(i,j) := 4 * i - 4 \leq j < 4 * i$
- **Adjacent** indices to the counter
  - $\text{Tile}(i,j) := j == i - 1$
  - $\text{Tile}(i,j) := j == i + 1$
- Most **common** tile in array processing loops
  - $\text{Tile}(i,j) := j == i$

Conclusion and Future Work

- Presented a novel verification technique that
  - proves universally quantified assertions over arrays
  - decomposes reasoning about arrays using *tiles*
  - is property driven, compositional and efficient
- Future directions
  - Automated synthesis of *tiles*
  - Combining the strengths of Booster, Vaphor and Tiler
  - Integration of other candidate invariant generators like Houdini
  - Verify sorting by tiling

Thank you