Awaiting for Godot: Stateless Model Checking that Avoids Executions where Nothing Happens

Bengt Jonsson, Magnus Lång, Kostis Sagonas

21st October 2022

Uppsala University, Sweden

Contributions

Contributions

- *Partial Loop Purity Elimination*:
  Eliminate *pure* loop iterations by inserting `await` statements

Contributions

- *Partial Loop Purity Elimination*:
  Eliminate *pure* loop iterations by inserting `await` statements
- Optimal-DPOR-Await:
  Efficient SMC algorithm supporting `await`s and independent fetch-and-adds

Contributions

- *Partial Loop Purity Elimination*:
  Eliminate *pure* loop iterations by inserting `await` statements
- Optimal-DPOR-Await:
  Efficient SMC algorithm supporting `await`s and independent fetch-and-adds
- Implementation in Nidhugg

Stateless Model Checking

Exhaustively test terminating concurrent program with fixed input

Stateless Model Checking

Exhaustively test terminating concurrent program with fixed input

- How: Systematically explore all thread schedulings

Stateless Model Checking

Exhaustively test terminating concurrent program with fixed input

- How: Systematically explore all thread schedulings
- Problem: Combinatorial explosion

Stateless Model Checking

Exhaustively test terminating concurrent program with fixed input

- How: Systematically explore all thread schedulings
- Problem: Combinatorial explosion
- Solution: *Dynamic Partial-Order Reduction (DPOR)* [Flanagan & Godefroid 2005] explores only the orderings of dependent events

Stateless Model Checking

Exhaustively test terminating concurrent program with fixed input

- How: Systematically explore all thread schedulings
- Problem: Combinatorial explosion
- Solution: *Dynamic Partial-Order Reduction (DPOR)* [Flanagan & Godefroid 2005] explores only the orderings of dependent events
- Optimal DPOR [Abdulla et al. 2014, 2017] never visits two equivalent executions

Termination in Stateless Model Checking

- Problem: The programs must terminate

Example: Spinloops

p	q

`y := 42;`
`x := 1`
	`do  a := x`
	`while(a ≠ 1);`
	`b := y`

Example: Spinloops

p	q

`y := 42;`
`x := 1`
	`do  a := x`
	`while(a ≠ 1);`
	`b := y`

p : `y := 42`

p : `x := 1`

q : `do  a := x`

q : `while(a ≠ 1)`

q : `b := y`

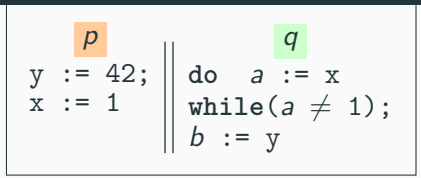
Example: Spinloops

p	q
`y := 42;`	`do  a := x`
`x := 1`	`while(a ≠ 1);`
	`b := y`

p : `y := 42`
p : `x := 1`
q : `do  a := x`
q : `while(a ≠ 1)`
q : `b := y`

p : `y := 42`
q : `do  a := x`
q : `while(a ≠ 1)`
p : `x := 1`
q : `do  a := x`
q : `while(a ≠ 1)`
q : `b := y`

Example: Spinloops



p : y := 42
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

Example: Spinloops

p	q
`y := 42;`	`do  a := x`
`x := 1`	`while(a ≠ 1);`
	`b := y`

p: `y := 42`
p: `x := 1`
q: `do  a := x`
q: `while(a ≠ 1)`
q: `b := y`

p: `y := 42`
q: `do  a := x`
q: `while(a ≠ 1)`
p: `x := 1`
q: `do  a := x`
q: `while(a ≠ 1)`
q: `b := y`

p: `y := 42`
q: `do  a := x`
q: `while(a ≠ 1)`
q: `do  a := x`
q: `while(a ≠ 1)`
p: `x := 1`
q: `do  a := x`
q: `while(a ≠ 1)`
q: `b := y`

...

Termination in Stateless Model Checking

- Problem: The programs must terminate

Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding

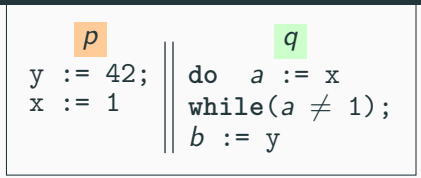
Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding
  - Sacrifices exhaustiveness

Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding
  - Sacrifices exhaustiveness
  - Poor performance

Example: Spinloops



p : y := 42
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

...

Example: Spinloops

p	q
y := 42; x := 1	do a := x while($a \neq 1$); b := y

p : y := 42
p : x := 1
q : do a := x
q : while($a \neq 1$)
q : b := y

p : y := 42
q : do a := x
q : while($a \neq 1$)
p : x := 1
q : do a := x
q : while($a \neq 1$)
q : b := y

p : y := 42
q : do a := x
q : while($a \neq 1$)
q : do a := x
q : while($a \neq 1$)
p : x := 1
q : do a := x
q : while($a \neq 1$)
q : b := y

...

Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding
  - Sacrifices exhaustiveness
  - Poor performance

Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding
  - Sacrifices exhaustiveness
  - Poor performance
- Observation: Many programs are non-terminating due to *pure* loop iterations

Termination in Stateless Model Checking

- Problem: The programs must terminate
- Common approach: Loop Bounding
  - Sacrifices exhaustiveness
  - Poor performance
- Observation: Many programs are non-terminating due to *pure* loop iterations
- Idea: Avoid exploring executions containing pure loop iterations

Example: Spinloops

p	q
y := 42; x := 1	do a := x while($a \neq 1$); b := y

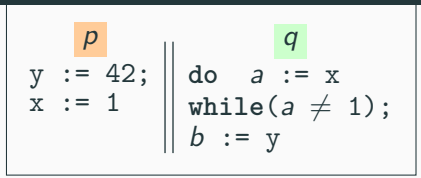
p: y := 42
p: x := 1
q: do a := x
q: while($a \neq 1$)
q: b := y

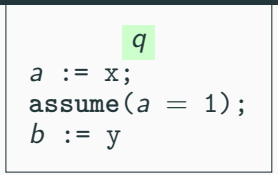
p: y := 42
q: do a := x
q: while($a \neq 1$)
p: x := 1
q: do a := x
q: while($a \neq 1$)
q: b := y

p: y := 42
q: do a := x
q: while($a \neq 1$)
q: do a := x
q: while($a \neq 1$)
p: x := 1
q: do a := x
q: while($a \neq 1$)
q: b := y

...

Example: Spinloops





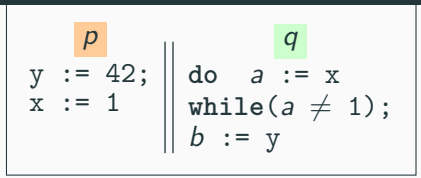
p : y := 42
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

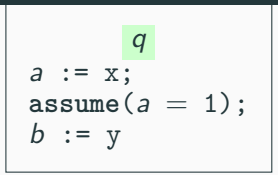
p : y := 42
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

...

Example: Spinloops





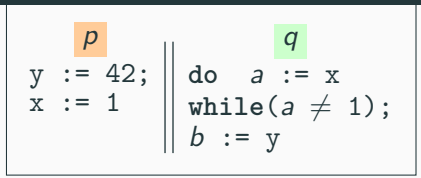
p : y := 42
p : x := 1
q : a := x
q : assume(a = 1)
q : b := y

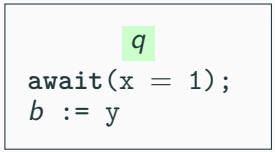
p : y := 42
q : a := x
q : assume(a = 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

...

Example: Spinloops





p : y := 42
p : x := 1
q : a := x
q : assume(a = 1)
q : b := y

p : y := 42
q : a := x
q : assume(a = 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1) ...
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

Example: Spinloops

```
p                    q
y := 42;  ||  do  a := x
x := 1    ||  while(a ≠ 1);
          ||  b := y
```

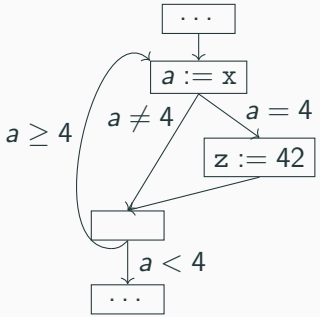
```
q
await(x = 1);
b := y
```

p : y := 42
p : x := 1
q : await(x = 1)
q : b := y

p : y := 42
q : a := x
q : assume(a = 1)
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

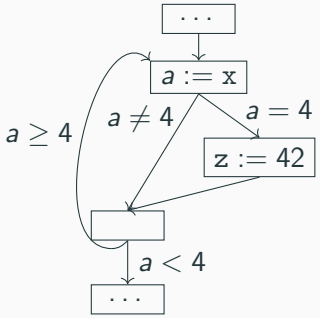
p : y := 42
q : do a := x
q : while(a ≠ 1)
q : do a := x
q : while(a ≠ 1) ...
p : x := 1
q : do a := x
q : while(a ≠ 1)
q : b := y

Partial Loop Purity Analysis



In SSA form

Partial Loop Purity Analysis

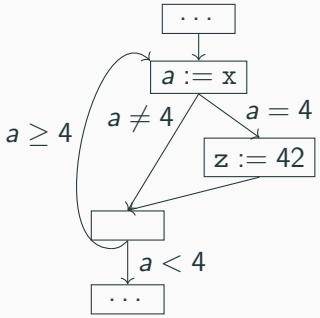


In SSA form

A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

Partial Loop Purity Analysis



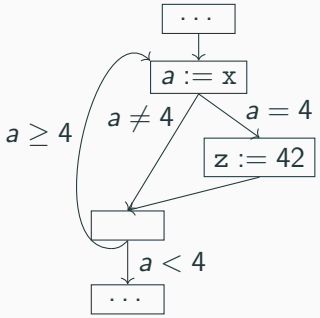
In SSA form

A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop

Partial Loop Purity Analysis



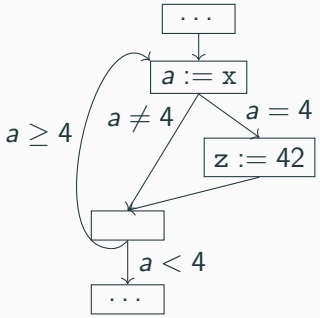
In SSA form

A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?

Partial Loop Purity Analysis



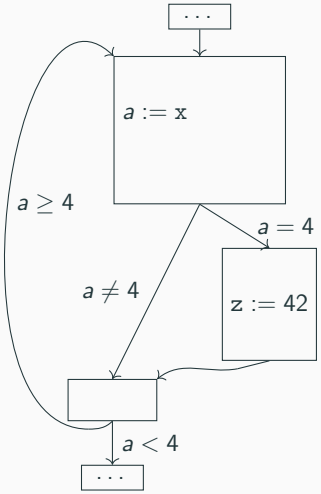
In SSA form

A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

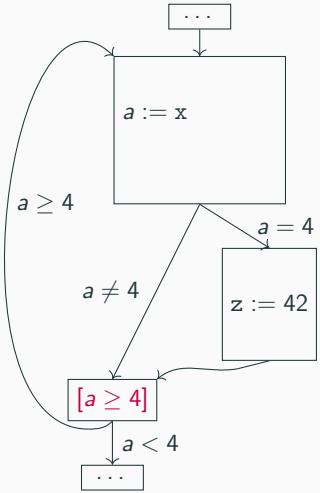


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

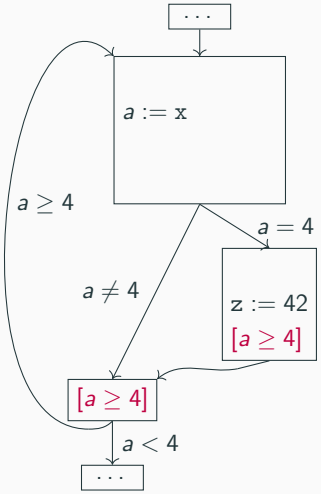


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

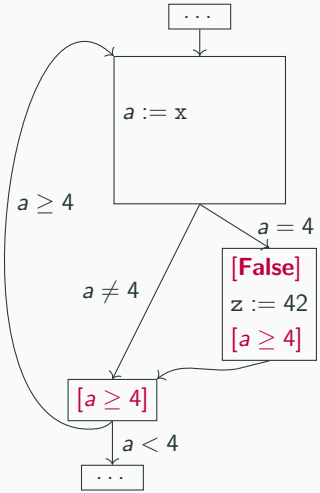


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

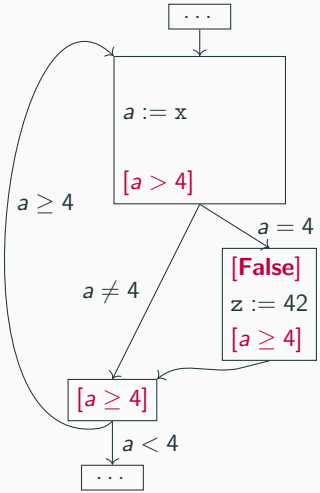


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

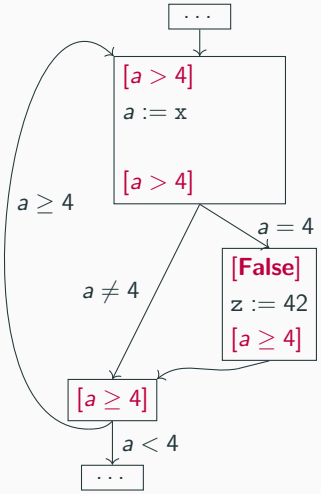


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Analysis

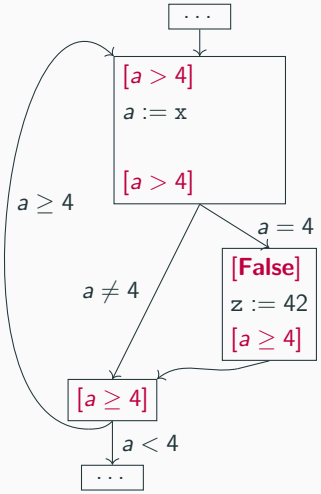


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)

Partial Loop Purity Elimination

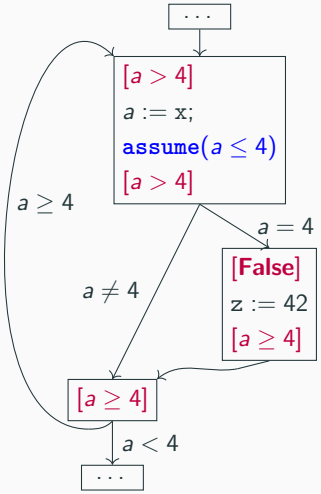


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)
- Insert `assumes` to prevent pure loop iterations

Partial Loop Purity Elimination

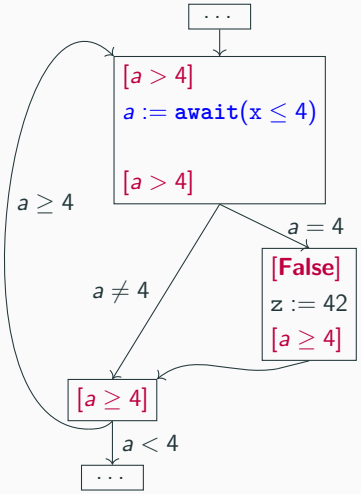


A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)
- Insert `assume`s to prevent pure loop iterations

Partial Loop Purity Elimination



A loop iteration is *pure* if it

1. does not have side-effects
2. does not exit the loop

- An iteration is pure when $a > 4$, the *purity condition* of this loop
- How do we compute it?
- Annotate each point with a *Forward Purity Condition* (for finishing a pure iteration)
- Insert `assumes` to prevent pure loop iterations

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements
- Efficient, even with independent fetch-and-adds (IFAA)

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements
- Efficient, even with independent fetch-and-adds (IFAA)

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements
- Efficient, even with independent fetch-and-adds (IFAA)

- Without IFAA: $n!$ executions

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements
- Efficient, even with independent fetch-and-adds (IFAA)

- Without IFAA: $n!$ executions
- With IFAA: 1 execution

Optimal-DPOR-Await

A new optimal DPOR algorithm

- Handles `await` statements
- Efficient, even with independent fetch-and-adds (IFAA)

And an implementation in NIDHUGG, with PLP and IFAA

- Without IFAA: $n!$ executions
- With IFAA: 1 execution

Evaluation

	GenMC	Nidhugg
Benchmark		

Evaluation

	GenMC	Nidhugg
Benchmark	Baseline	Baseline

Evaluation

	GenMC		Nidhugg
Benchmark	Baseline	Saver	Baseline

Evaluation

	GenMC		Nidhugg	
Benchmark	Baseline	Saver	Baseline	PLP

Evaluation

	GenMC		Nidhugg		
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564
mutex(2)	12+2	12+2	12+2	12+2	12	10
mutex(3)	9486+1236	6582+1188	9486+1236	6582+1188	6582+336	3618+312

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564
mutex(2)	12+2	12+2	12+2	12+2	12	10
mutex(3)	9486+1236	6582+1188	9486+1236	6582+1188	6582+336	3618+312
seqlock(3)	147+230	9+83	147+230	9+83	9+36	9+36
seqlock(4)	87980+105123	88+2805	87980+104583	88+2769	88+729	88+729

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564
mutex(2)	12+2	12+2	12+2	12+2	12	10
mutex(3)	9486+1236	6582+1188	9486+1236	6582+1188	6582+336	3618+312
seqlock(3)	147+230	9+83	147+230	9+83	9+36	9+36
seqlock(4)	87980+105123	88+2805	87980+104583	88+2769	88+729	88+729
mpmc-queue(3)	11206+11612	166+987	11206+8188	166+840	166+517	76+421
mpmc-queue(4)		39706+1277783		39706+1123234	39706+360426	5410+114208

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564
mutex(2)	12+2	12+2	12+2	12+2	12	10
mutex(3)	9486+1236	6582+1188	9486+1236	6582+1188	6582+336	3618+312
seqlock(3)	147+230	9+83	147+230	9+83	9+36	9+36
seqlock(4)	87980+105123	88+2805	87980+104583	88+2769	88+729	88+729
mpmc-queue(3)	11206+11612	166+987	11206+8188	166+840	166+517	76+421
mpmc-queue(4)		39706+1277783		39706+1123234	39706+360426	5410+114208
treiber-stack(3)	426	274+80	426	274+80	274+60	274+60
treiber-stack(4)	1546168+9216	250088+167916	1546168+9216	250088+167916	250088+90896	250088+90896

Evaluation

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
qspinlock(2)	6+2	6+2	6+2	6+2	6+2	6+2
qspinlock(3)	564	564	564	564	564	564
mutex(2)	12+2	12+2	12+2	12+2	12	10
mutex(3)	9486+1236	6582+1188	9486+1236	6582+1188	6582+336	3618+312
seqlock(3)	147+230	9+83	147+230	9+83	9+36	9+36
seqlock(4)	87980+105123	88+2805	87980+104583	88+2769	88+729	88+729
mpmc-queue(3)	11206+11612	166+987	11206+8188	166+840	166+517	76+421
mpmc-queue(4)	🐻	39706+1277783	🐻	39706+1123234	39706+360426	5410+114208
treiber-stack(3)	426	274+80	426	274+80	274+60	274+60
treiber-stack(4)	1546168+9216	250088+167916	1546168+9216	250088+167916	250088+90896	250088+90896
sortnet(4)	†	1+728	1+312	1+312	1	1
sortnet(5)	†	1+15231	1+4517	1+4517	1	1
sortnet(6)	†	1+163292	1+38285	1+38285	1	1

SafeStack

An attempt at a lock-free stack

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`
- Smallest test harness with the bug: `safestack-331`

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`
- Smallest test harness with the bug: `safestack-331`
- After 80+ days, GenMC (w & w/o Saver) and Nidhugg found nothing

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`
- Smallest test harness with the bug: `safestack-331`
- After 80+ days, GenMC (w & w/o Saver) and Nidhugg found nothing
- Nidhugg/...+IFAA:

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`
- Smallest test harness with the bug: `safestack-331`
- After 80+ days, GenMC (w & w/o Saver) and Nidhugg found nothing
- Nidhugg/...+IFAA:
  - finds it in 8 min, 2+2453474 executions

SafeStack

An attempt at a lock-free stack

- Posted on CHESS Mailing List in 2010
- No systematic technique can consistently find the bug
- Original test harness: `safestack-444`
- Smallest test harness with the bug: `safestack-331`
- After 80+ days, GenMC (w & w/o Saver) and Nidhugg found nothing
- Nidhugg/...+IFAA:
  - finds it in 8 min, 2+2453474 executions
  - exhausts the search space in 24 mins, 5772+8521721 executions

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25
safestack-32(2)	7189+296	7189+296	7189+296	1073+27	1073+12	463+12

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25
safestack-32(2)	7189+296	7189+296	7189+296	1073+27	1073+12	463+12
safestack-33(2)	121334+12652	121334+12652	121334+12652	6434+1636	6434+1584	2600+1160

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25
safestack-32(2)	7189+296	7189+296	7189+296	1073+27	1073+12	463+12
safestack-33(2)	121334+12652	121334+12652	121334+12652	6434+1636	6434+1584	2600+1160
safestack-211(3)	1267120+325932	995224+325932	1259280+324382	2690+1126	2690+928	962+686

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25
safestack-32(2)	7189+296	7189+296	7189+296	1073+27	1073+12	463+12
safestack-33(2)	121334+12652	121334+12652	121334+12652	6434+1636	6434+1584	2600+1160
safestack-211(3)	1267120+325932	995224+325932	1259280+324382	2690+1126	2690+928	962+686
safestack-311(3)	0+286818740	0+275399108		0+26536	0+24078	0+14960

Results on SafeStack

	GenMC		Nidhugg			
Benchmark	Baseline	Saver	Baseline	PLP	PLP+Await	...+IFAA
safestack-21(2)	119+6	119+6	119+6	34+2	34+1	19+1
safestack-31(2)	928+107	928+107	928+107	103+27	103+25	56+25
safestack-32(2)	7189+296	7189+296	7189+296	1073+27	1073+12	463+12
safestack-33(2)	121334+12652	121334+12652	121334+12652	6434+1636	6434+1584	2600+1160
safestack-211(3)	1267120+325932	995224+325932	1259280+324382	2690+1126	2690+928	962+686
safestack-311(3)	0+286818740	0+275399108		0+26536	0+24078	0+14960
safestack-321(3)				906529+388117	906529+331337	288057+216830

Conclusions

- Partial Pure Loop Analysis can discover pure loop iterations

Conclusions

- Partial Pure Loop Analysis can discover pure loop iterations
- Rewriting them with `await` statements eliminates them

Conclusions

- Partial Pure Loop Analysis can discover pure loop iterations
- Rewriting them with `await` statements eliminates them
- Optimal-DPOR-Await efficiently checks programs with `await`s

Conclusions

- Partial Pure Loop Analysis can discover pure loop iterations
- Rewriting them with **`await`** statements eliminates them
- Optimal-DPOR-Await efficiently checks programs with **`await`**s

Available today in Nidhugg: `https://github.com/nidhugg/nidhugg`

Conclusions

- Partial Pure Loop Analysis can discover pure loop iterations
- Rewriting them with **`await`** statements eliminates them
- Optimal-DPOR-Await efficiently checks programs with **`await`**s

Available today in Nidhugg: `https://github.com/nidhugg/nidhugg`

Thank you for listening!