Difference Verification with Conditions

Thomas Lemberger

Joint work with Dirk Beyer and Marie-Christine Jakobs

LMU Munich and TU Darmstadt

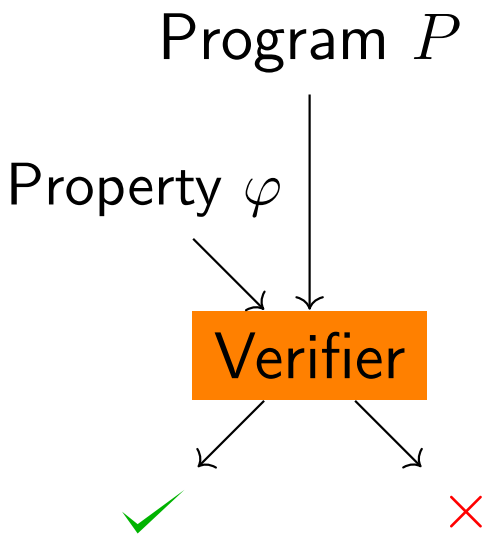


Preprint: https://www.sosy-lab.org/research/pub/2020-SEFM.Difference_Verification_with_Conditions.pdf

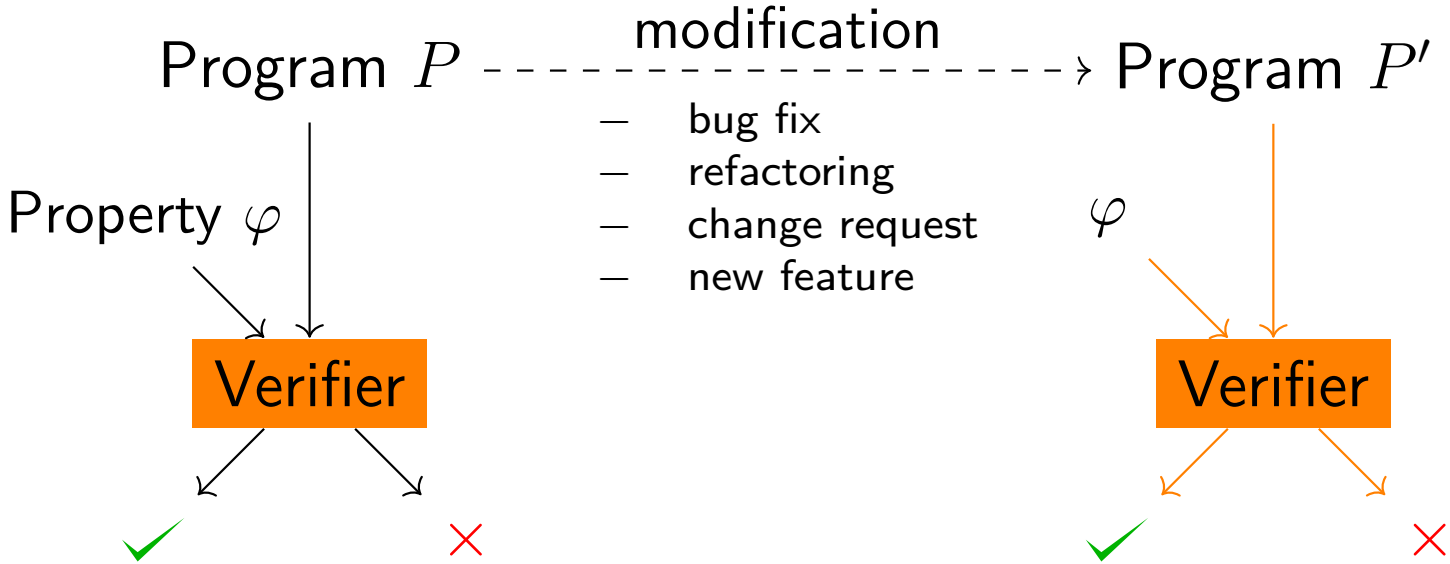
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

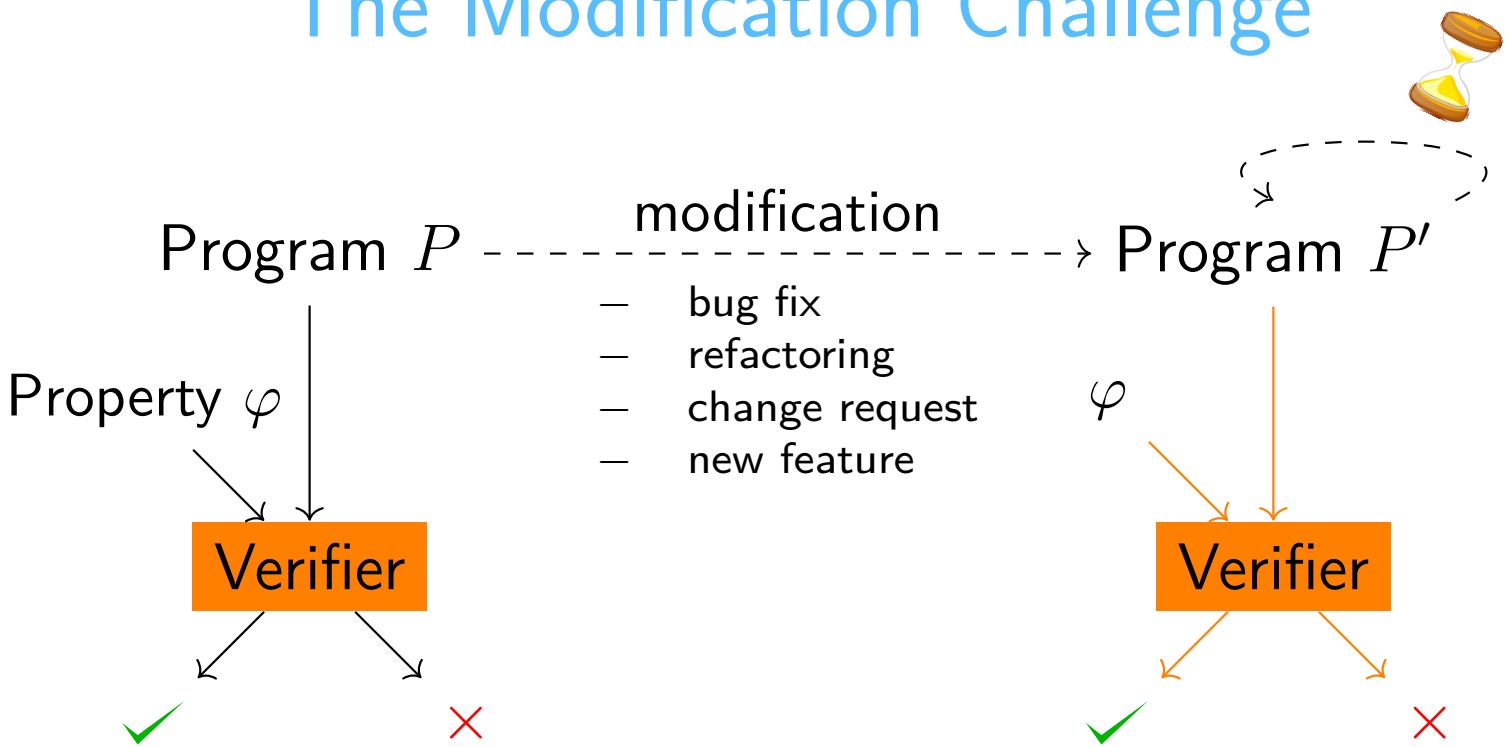
The Modification Challenge



The Modification Challenge



The Modification Challenge



Challenge:

- Programs change frequently
- Reverification necessary after each change
- Reverification must keep up with changes

$\Rightarrow$ Verification of every modified program **infeasible!**

Program Modifications

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program Modifications

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

Program Modifications

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

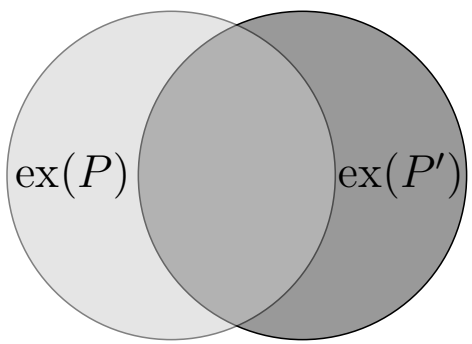
Program Modifications

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```



Program Modifications

Program P

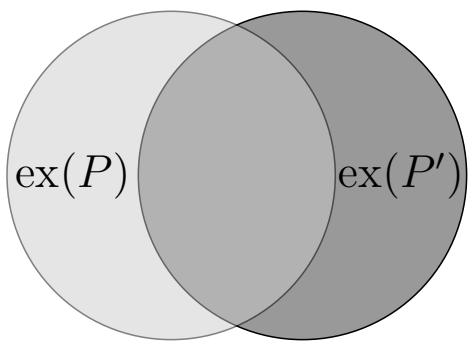
```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

Assumption:

Program modifications affect only few program executions



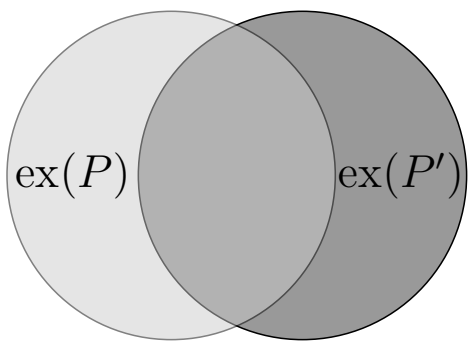
Program Modifications

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```



Assumption:

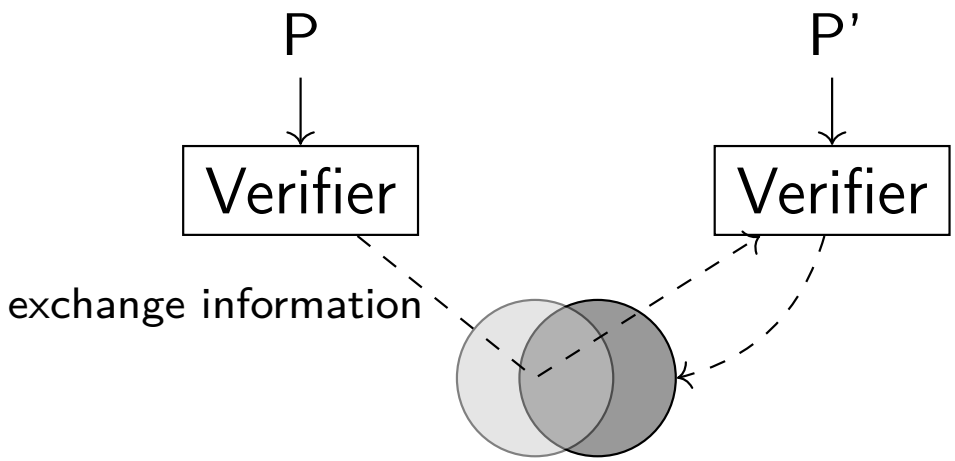
Program modifications affect only few program executions

$\Rightarrow$ verification of unchanged program executions wastes resources

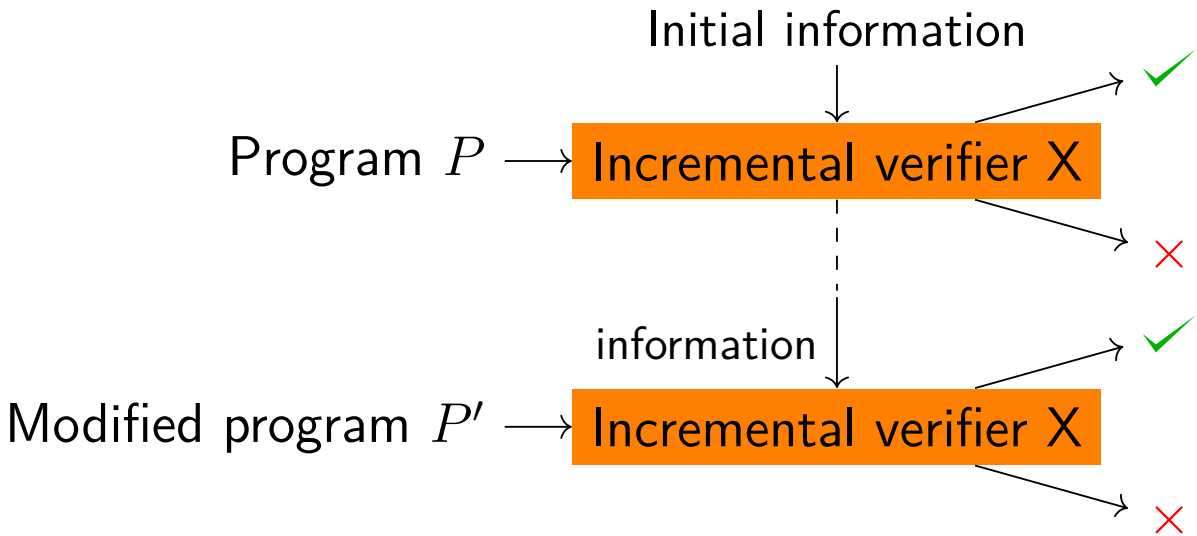
$\Rightarrow$ **better:** reuse information obtained in previous verification run(s)

Principle of Incremental Verification

General idea: Use information about previous verification run
$\Rightarrow$ speed up verification of unchanged parts



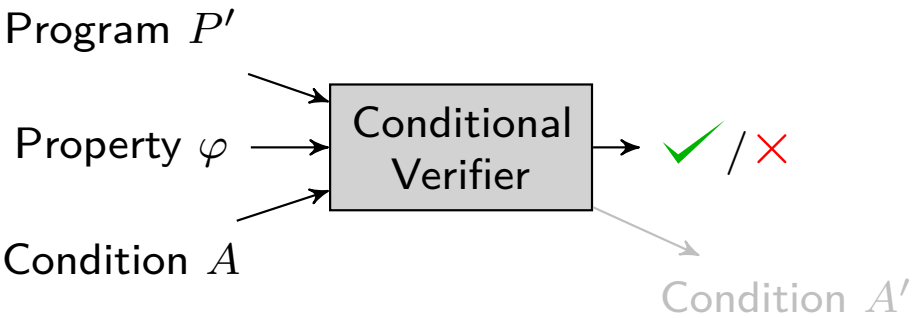
Weaknesses of Existing Approaches



Often...

- require initial full verification (initial information)
- require same verifier in each step
- tailored to specific verification approach

Conditional Model Checking[1]

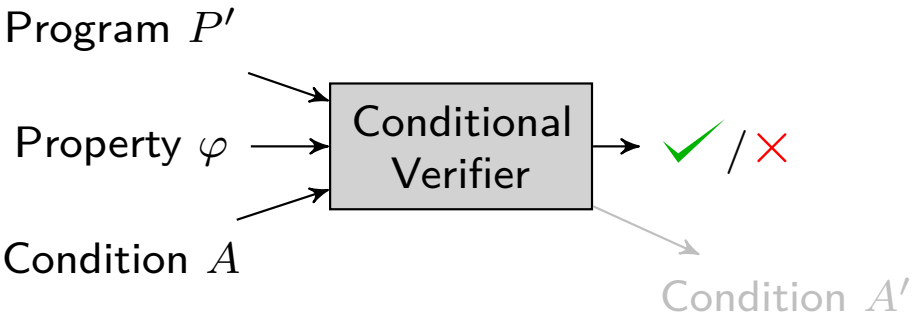


- Condition: Automaton
- Accepting runs describe program-state space to consider

- *Source-code guards* restrict syntactic paths
- *State-space guards* restrict possible program states
- *Here: only source-code guards*

[1] D. Beyer, T. A. Henzinger, M. E. Keremoglu, P. Wendler: Conditional Model Checking, 2012.

Conditional Model Checking[1]

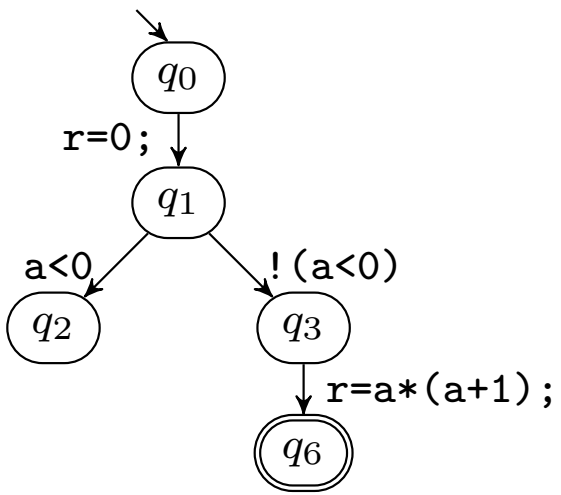


- Condition: Automaton
- Accepting runs describe program-state space to consider

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

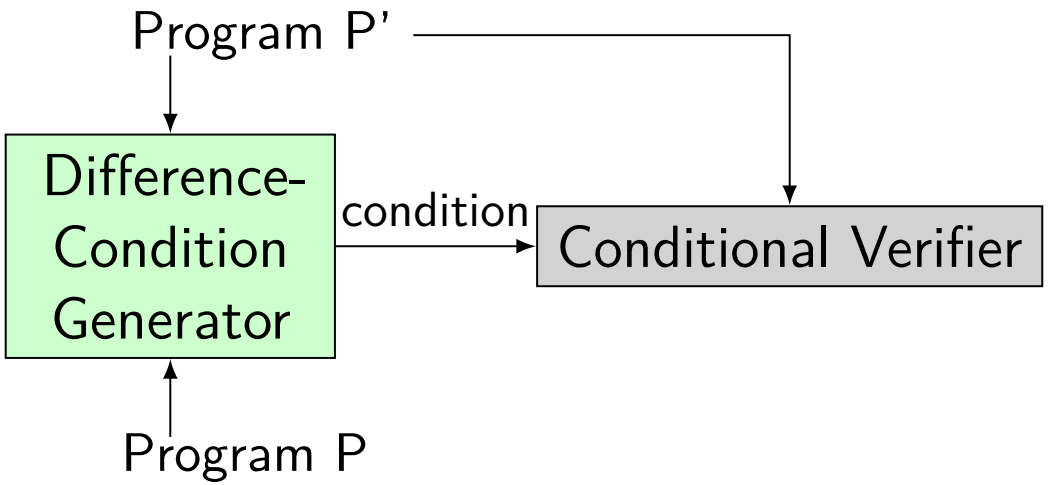
Example Condition



- *Source-code guards* restrict syntactic paths
- *State-space guards* restrict possible program states
- *Here: only source-code guards*

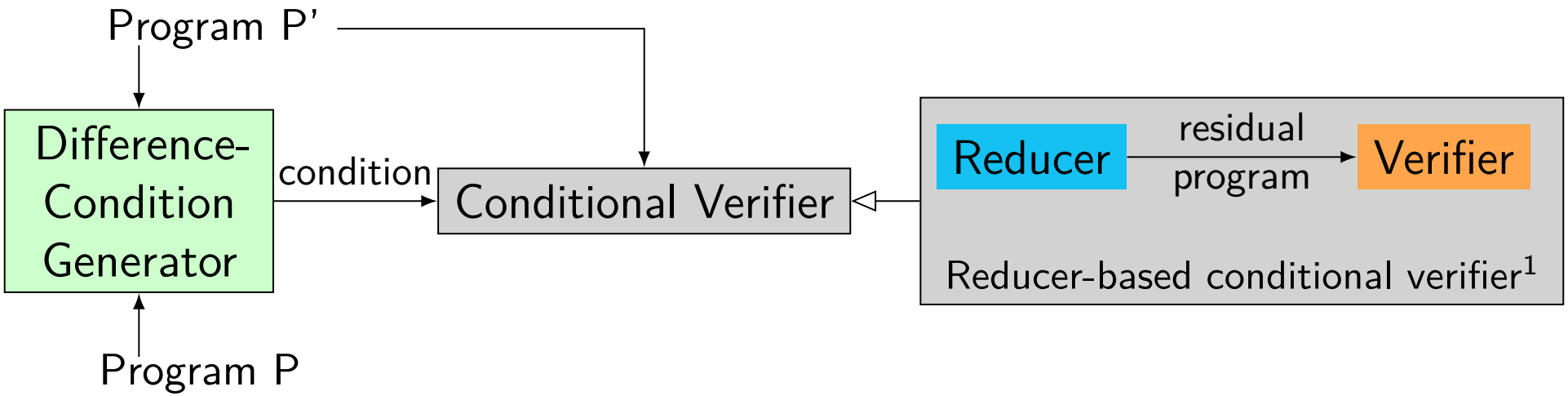
[1] D. Beyer, T. A. Henzinger, M. E. Keremoglu, P. Wendler: Conditional Model Checking, 2012.

Difference Verification with Conditions



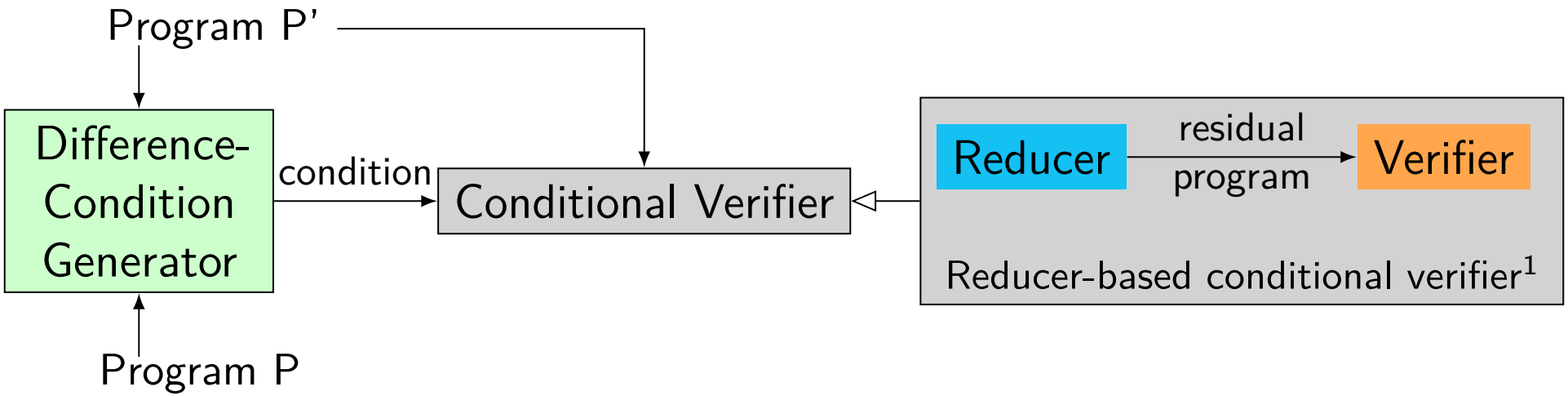
[1] D. Beyer, M.-C. Jakobs, T. Lemberger, H. Wehrheim: Reducer-Based Construction of Conditional Verifiers

Difference Verification with Conditions



[1] D. Beyer, M.-C. Jakobs, T. Lemberger, H. Wehrheim: Reducer-Based Construction of Conditional Verifiers

Difference Verification with Conditions



Soundness Criterion:

The generated condition does not cover any modified execution, i.e., it does not cover executions from $ex(P') \setminus ex(P)$.

[1] D. Beyer, M.-C. Jakobs, T. Lemberger, H. Wehrheim: Reducer-Based Construction of Conditional Verifiers

Difference Verification with Conditions

- No initial verification necessary
- Applicable to arbitrary verifiers
  due to reducer-based conditional model checking
- Unchanged program parts are as correct as before
- Only program parts affected by change are verified
- Verifier can be newly selected for each verification run

Example implementation for difference-condition generator

DiffCond: Syntactic Difference-Condition Generator

DiffCond: Syntactic Difference-Condition Generator

1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

DiffCond: Syntactic Difference-Condition Generator

1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

DiffCond: Syntactic Difference-Condition Generator

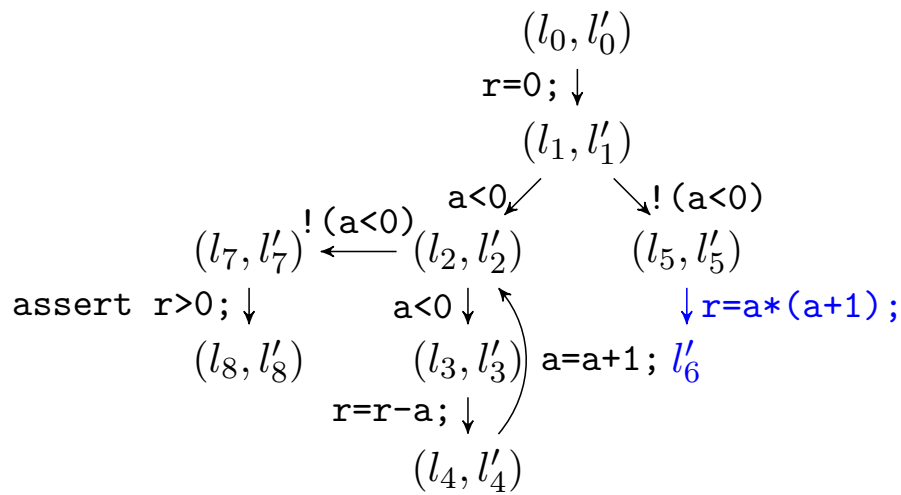
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```



DiffCond: Syntactic Difference-Condition Generator

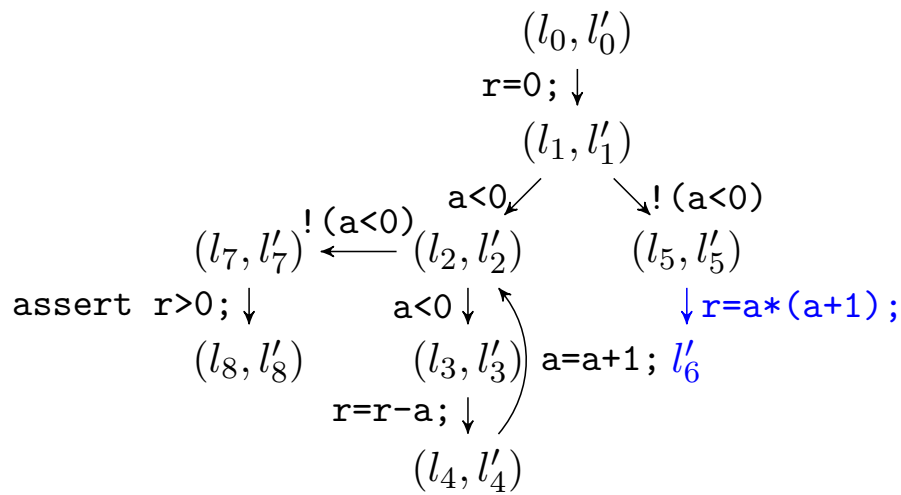
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```



2. Generate condition from parallel composition (standard procedure)

DiffCond: Syntactic Difference-Condition Generator

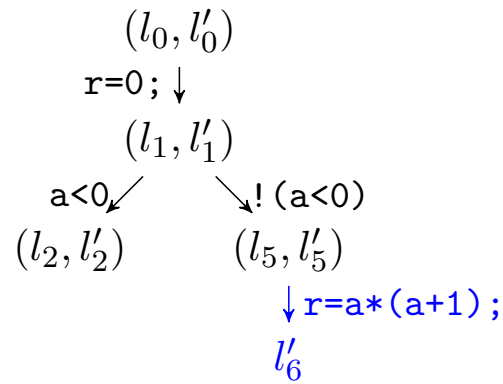
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```



2. Generate condition from parallel composition (standard procedure)

DiffCond: Syntactic Difference-Condition Generator

1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Program P

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a+a+1;
6   r=r/2;
7 assert r>0;
```

Program P'

```
0 r=0;
1 if(a<0)
2   while(a<0)
3     r=r-a;
4     a=a+1;
  else
5   r=a*(a+1);
6   r=r/2;
7 assert r>0;
```

Theorem (proven):

DiffCond fulfills soundness criterion

2. Generate condition from parallel composition (standard procedure)

Theoretical Limitations

- Modifications propagate
  - Changes in global variables
  - Looping programs

$\Rightarrow$ Works best on loosely coupled systems

Program P^{∞}

```
0 while (1)
1   r=0;
2   if(a<0)
3     while(a<0)
4       r=r-a;
5       a=a+1;
    else
6     r=a+a+1;
7     r=r/2;
8   assert r>0;
```

Evaluation

Evaluation

1. Full verification vs. diff. verification (native cond. verifier)
2. Full verification vs. diff. verification (reducer-based cond. verifier)
3. Comparison with existing incr. technique: *Precision reuse*
   - Difference in performance?
   - Combination of both techniques?

Evaluation Setting

Environment

- Similar to Software-Verification Competition SV-COMP
- 15 min. time limit, 15 GB memory per verification run

Evaluation Setting

Environment

- Similar to Software-Verification Competition SV-COMP
- 15 min. time limit, 15 GB memory per verification run

Verifiers

- Predicate-based, native conditional model checker
- CPA-Seq
- Ultimate Automizer

Evaluation Setting

Evaluation Tasks

- sv-benchmarks strongly coupled
- Combinations of two sv-benchmarks tasks, each

```
int main() {
  if(__VERIFIER_nondet_int())
    main1();
  else
    main2();
}
```

Evaluation Setting

Evaluation Tasks

- sv-benchmarks strongly coupled
- Combinations of two sv-benchmarks tasks, each

```
int main() {
  if(__VERIFIER_nondet_int())
    main1();
  else
    main2();
}
```

- 5 different combination-pairs

Evaluation Setting

Evaluation Tasks

- sv-benchmarks strongly coupled
- Combinations of two sv-benchmarks tasks, each

```
int main() {
  if(__VERIFIER_nondet_int())
    main1();
  else
    main2();
}
```

- 5 different combination-pairs

- Program Modification
  - Replace one of the two tasks by other version of same task
  - Fix an unsafe program, introduce a bug, or stay safe

Evaluation Setting

Evaluation Tasks

- sv-benchmarks strongly coupled
- Combinations of two sv-benchmarks tasks, each

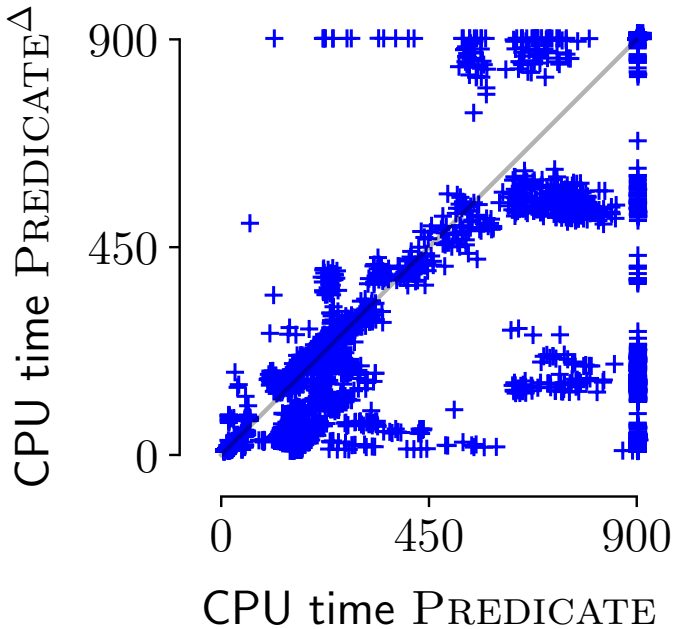
```
int main() {
  if(__VERIFIER_nondet_int())
    main1();
  else
    main2();
}
```

- 5 different combination-pairs

- Program Modification
  - Replace one of the two tasks by other version of same task
  - Fix an unsafe program, introduce a bug, or stay safe

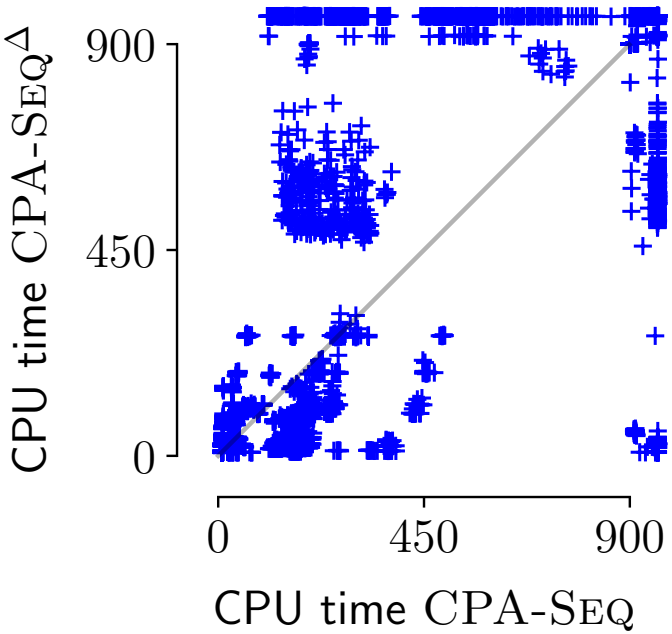
$\Rightarrow$ 10 426 verification tasks

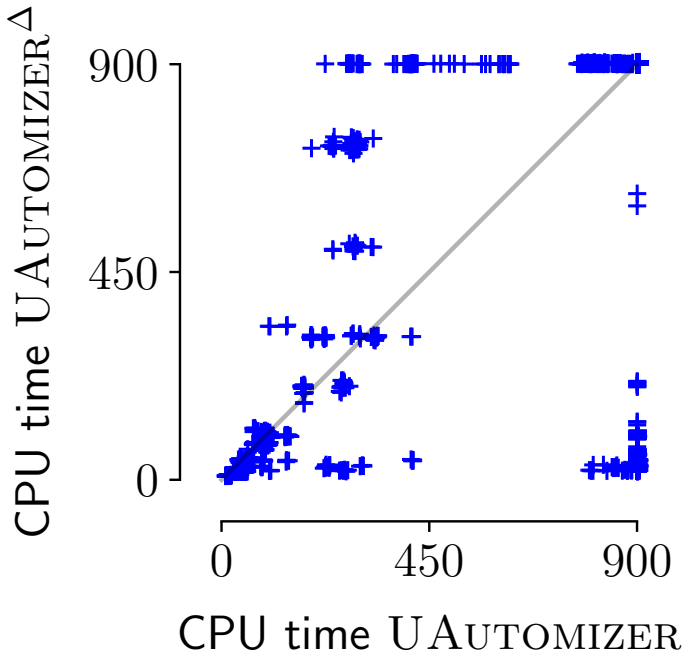
1. Full vs. Diff. Verification (Native)



- CPU time (in s) per task Δ: Diff. verification
- Typically, difference verification with conditions similar or better

2. Full vs. Diff. Verification (Reducer-Based)

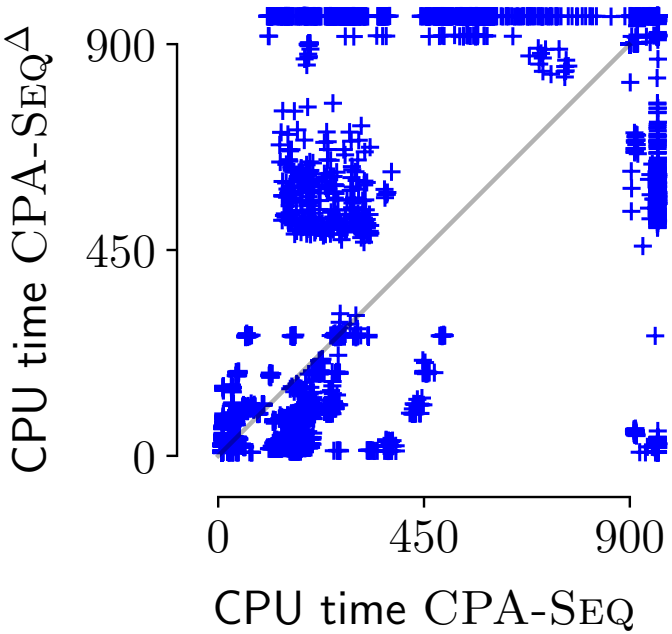


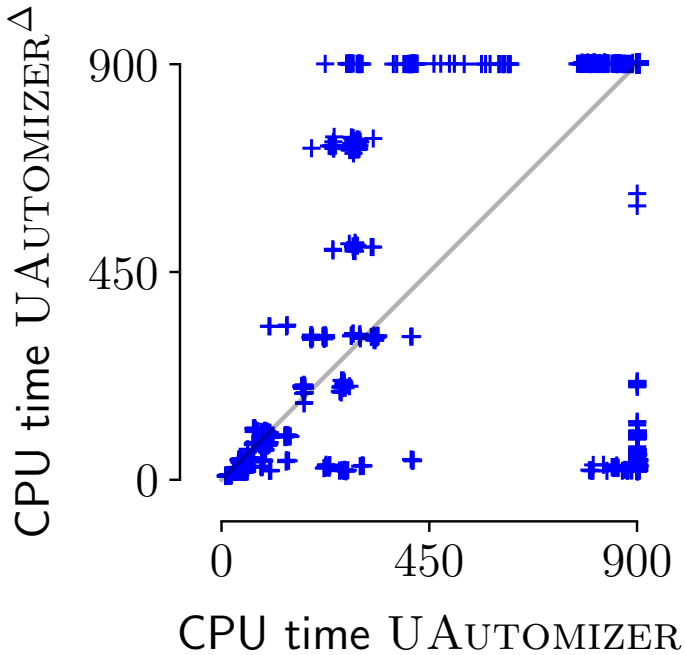


- CPU time (in s) per task

Δ: Diff. verification

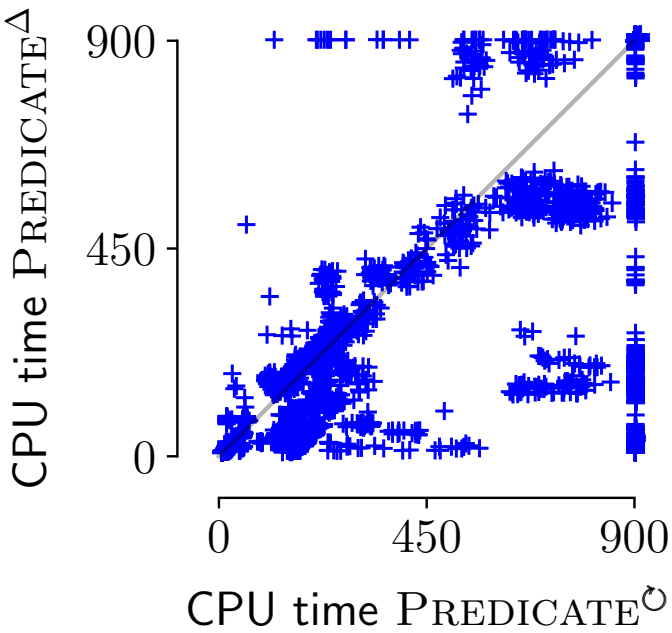
2. Full vs. Diff. Verification (Reducer-Based)

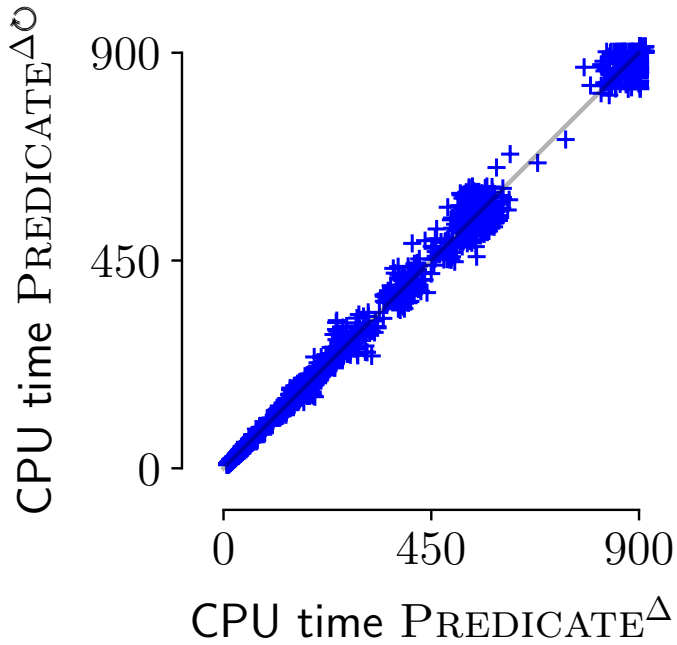




- CPU time (in s) per task Δ: Diff. verification
- Diff. verification can be better
- Full verification often better
  (diff. verification suffers from residual programs)

3. Comparison with Precision Reuse[3]

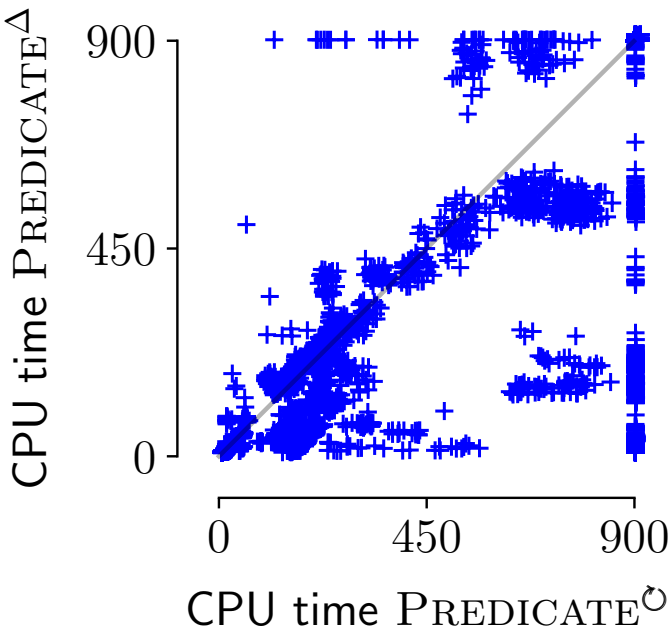


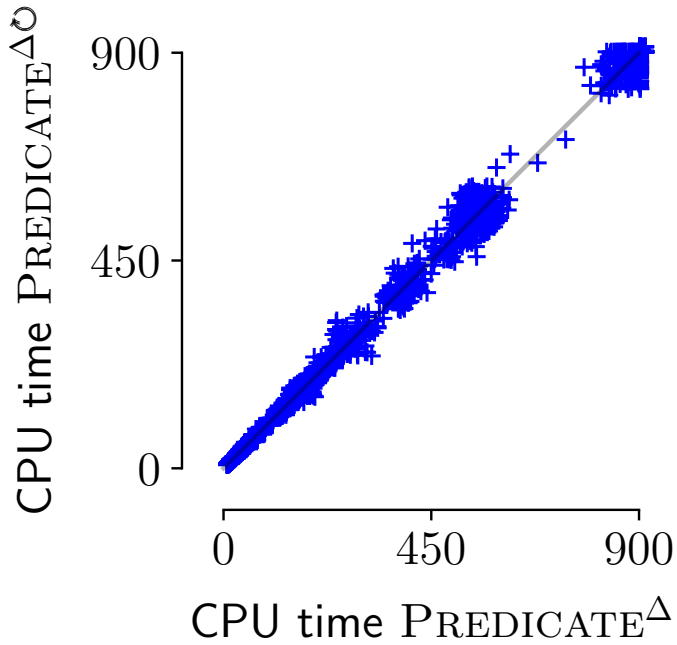


- ↻: Precision reuse Δ: Diff. verification Δ ↻: Combo

[3] D. Beyer, S. Löwe, E. Novikov, A. Stahlbauer, P. Wendler: Precision Reuse for Efficient Regression Verification, 2013.

3. Comparison with Precision Reuse[3]

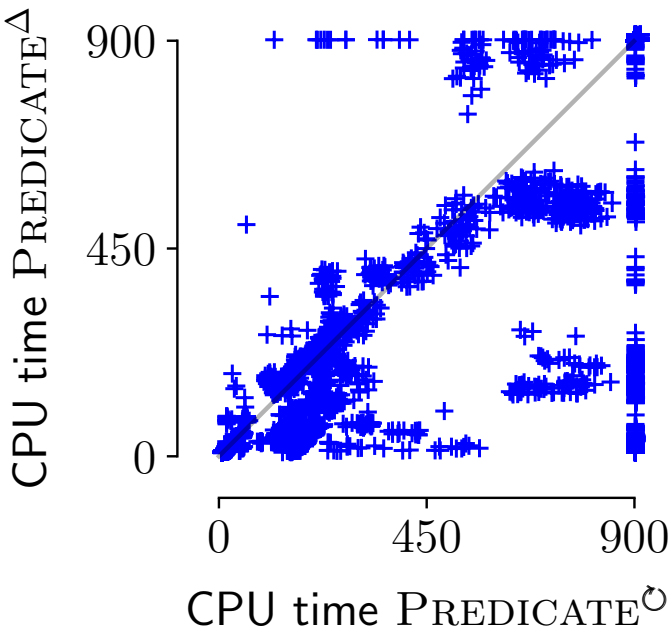


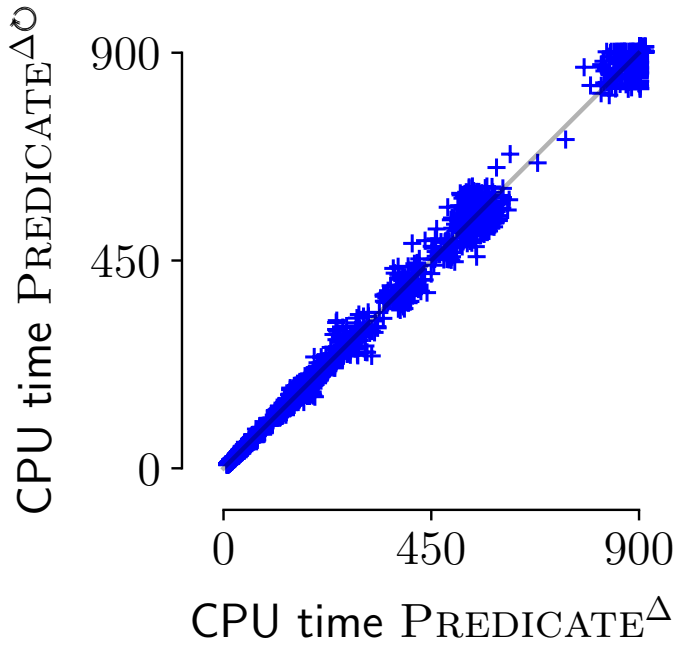


- ↻: Precision reuse Δ: Diff. verification Δ ↻: Combo
- Diff. verification often better than precision reuse

[3] D. Beyer, S. Löwe, E. Novikov, A. Stahlbauer, P. Wendler: Precision Reuse for Efficient Regression Verification, 2013.

3. Comparison with Precision Reuse[3]

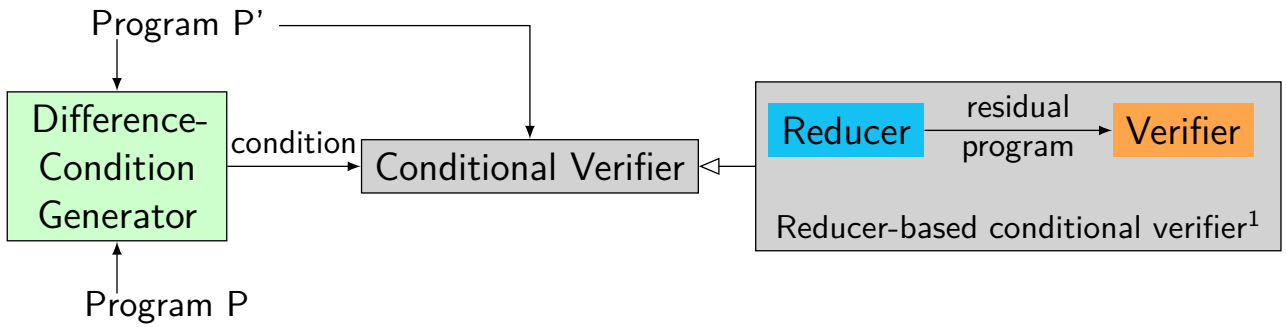


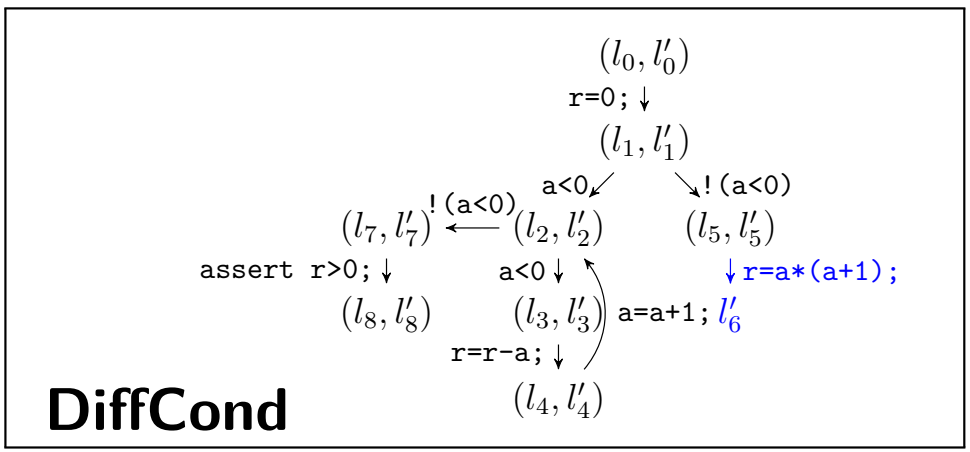


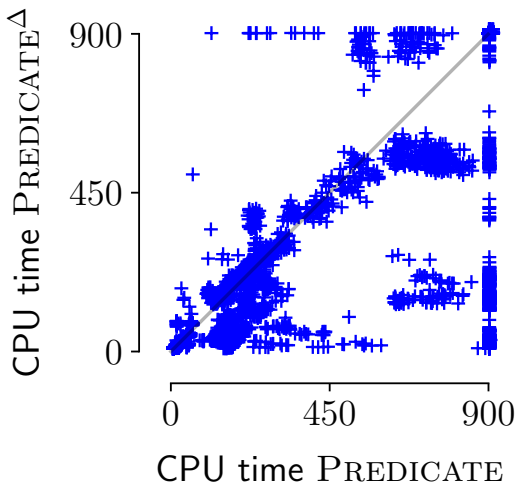
- ↻: Precision reuse Δ: Diff. verification Δ ↻: Combo
- Diff. verification often better than precision reuse
- Combination of the two approaches only seldomly beneficial (but can be: +29 tasks)

[3] D. Beyer, S. Löwe, E. Novikov, A. Stahlbauer, P. Wendler: Precision Reuse for Efficient Regression Verification, 2013.

Conclusion


Program P'
Difference-Condition Generator
condition
Conditional Verifier
Reducer
residual program
Verifier
Reducer-based conditional verifier[1]
Program P


(l_0, l'_0)
r=0;
(l_1, l'_1)
a<0
!(a<0)
(l_7, l'_7)
!(a<0)
(l_2, l'_2)
(l_5, l'_5)
assert r>0;
a<0
r=a*(a+1);
(l_8, l'_8)
(l_3, l'_3)
a=a+1; l'_6
r=r-a;
(l_4, l'_4)
DiffCond


CPU time PREDICATE$^\Delta$
900
450
0
0
450
900
CPU time PREDICATE