Differential Modular Software Verification

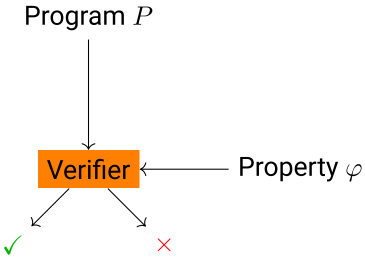
TECHNISCHE UNIVERSITÄT DARMSTADT

CPA'19

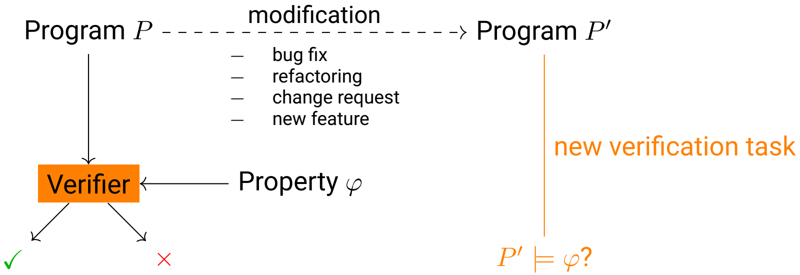
2019-10-01

Marie-Christine Jakobs

The Modification Challenge

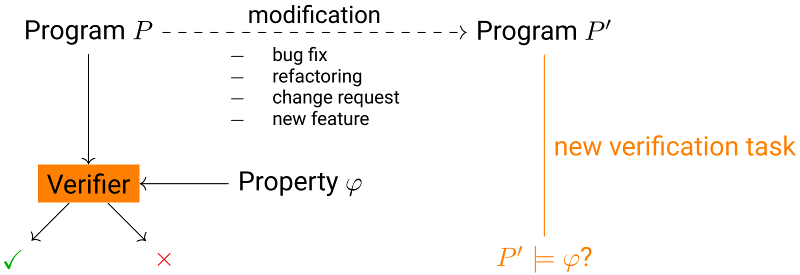


The Modification Challenge



The Modification Challenge

TECHNISCHE UNIVERSITÄT DARMSTADT



Challenge

- Programs change frequently
- Reverification after each change required
- Reverification must keep up with change rate

$\Longrightarrow$ Verifying modified programs from scratch infeasible

Insights about Program Modification

TECHNISCHE UNIVERSITÄT DARMSTADT

Observation: Program modifications typically affect few program executions

Insights about Program Modification

Observation: Program modifications typically affect few program executions

Original 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;
```

Modified 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;
```

Insights about Program Modification

TECHNISCHE UNIVERSITÄT DARMSTADT

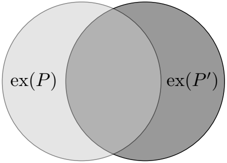
Observation: Program modifications typically affect few program executions

Original 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;
```

Modified 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;
```



Insights about Program Modification

TECHNISCHE UNIVERSITÄT DARMSTADT

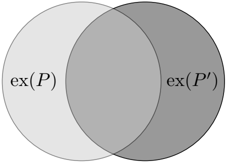
Observation: Program modifications typically affect few program executions

Original 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;
```

Modified 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;
```

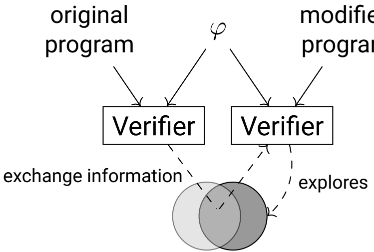


$\Longrightarrow$ verifying unchanged program executions from scratch wastes resources

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

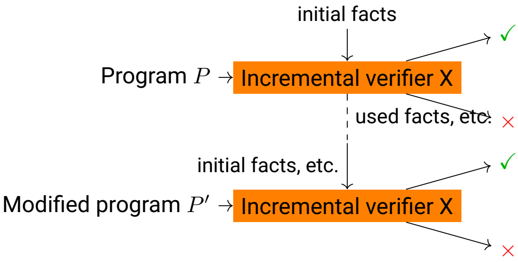
Principle of Incremental Verification

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



Weaknesses of Existing Approaches

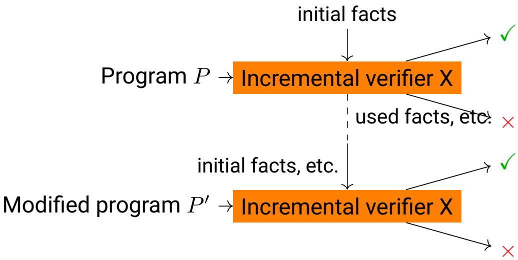
TECHNISCHE
UNIVERSITÄT
DARMSTADT



Often,

- Require initial full verification
- Use same verifier in each step
- Tailored to specific verification approach

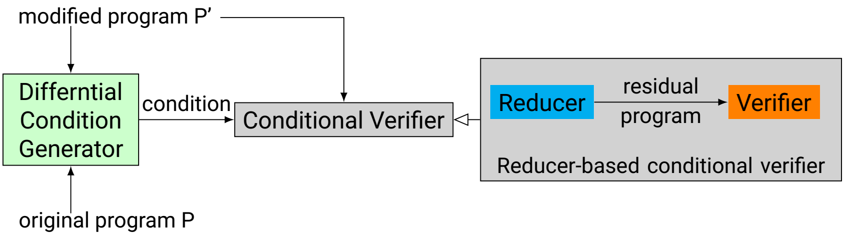
Weaknesses of Existing Approaches



Often,
- Require initial full verification
- Use same verifier in each step
- Tailored to specific verification approach

Equivalence checking
- Does not verify property
- Modified program only as correct as original
- Can be expensive

Differential Modular Software Verification
The Basic Idea



Soundness Criterion

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

Properties of Modular Differential Software Verification

TECHNISCHE UNIVERSITÄT DARMSTADT

- Unchanged program parts are as correct as before
- Only verify modified program parts
- Applicable to arbitrary verifiers
  due to reducer-based conditional model checking
- Verifier can be newly selected for each modified program

Syntactic Differential Condition Generator

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Syntactic Differential Condition Generator

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Original 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;
```

Modified 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;
```

Syntactic Differential Condition Generator

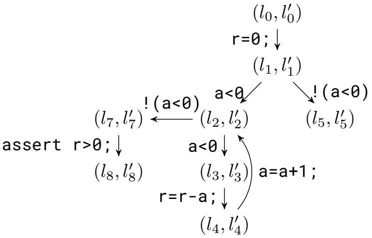
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Original 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;
```

Modified 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;
```



Syntactic Differential Condition Generator

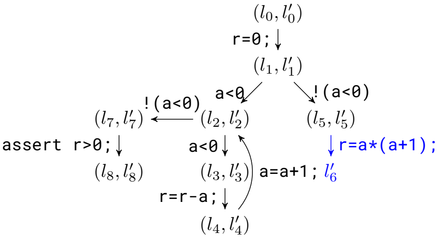
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Original 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;
```

Modified 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;
```



Syntactic Differential Condition Generator

TECHNISCHE UNIVERSITÄT DARMSTADT

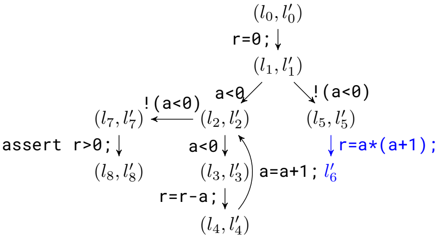
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Original 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;
```

Modified 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 ARG of parallel composition (standard procedure)

Syntactic Differential Condition Generator

TECHNISCHE UNIVERSITÄT DARMSTADT

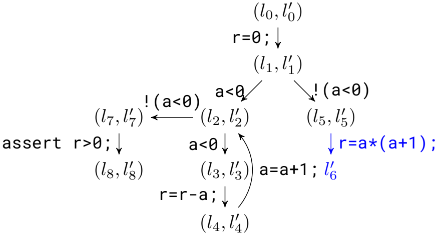
1. Detect differences
   - Compute the parallel composition of original and modified program
   - Stop if modified program deviates

Original 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;
```

Modified 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 ARG of parallel composition (standard procedure)

Theorem (proven): Syntactical generator fulfills soundness criterion

Evaluation Setting

TECHNISCHE UNIVERSITÄT DARMSTADT

Environment

- Similar to SV-COMP
- But only granted 4 cores

Evaluation Setting

TECHNISCHE UNIVERSITÄT DARMSTADT

Environment

- Similar to SV-COMP
- But only granted 4 cores

Verifiers

- Predicate-based, native conditional model checker
- CPASeq
- Ultimate Automizer

Evaluation Tasks

- Combinations of two SV-COMP tasks

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

Evaluation Tasks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Combinations of two SV-COMP tasks

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

- (1) eca+token, (2) gcd+newton, (3) pals+eca, (4) sfifo+token, (5) square+soft, and (6) ssh (client+server)

Evaluation Tasks

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Combinations of two SV-COMP tasks

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

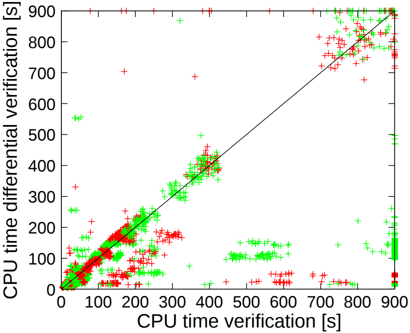
- (1) eca+token, (2) gcd+newton, (3) pals+eca, (4) sfifo+token, (5) square+soft, and (6) ssh (client+server)

- Program Modification
  - Replace one SV-COMP task by another of same category
  - Either fix an unsafe program or
  - Introduce a bug in a safe program

  $\Longrightarrow$ Full verification and our approach find the same violations

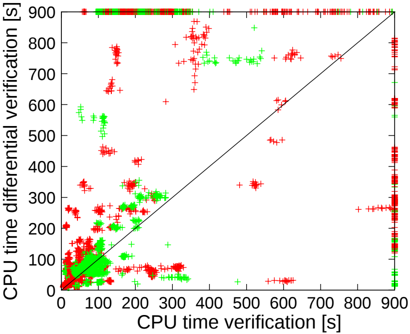
Full vs. Differential (native CMC)

TECHNISCHE UNIVERSITÄT DARMSTADT



Typically, modular differential software verification similar or better

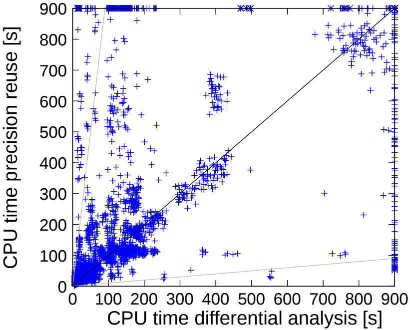
Full vs. Differential (reducer-based CMC)



- Modular differential software verification can be better
- Full verification often better
  (modular differential software verification suffers from residual programs)

Comparison with Precision Reuse

TECHNISCHE
UNIVERSITÄT
DARMSTADT

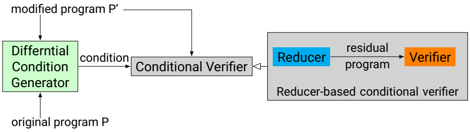


- Modular differential software verification often better than precision reuse
- Combination of two approaches often not beneficial

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

Proposed differential modular software verification



Identified weaknesses

- Evaluation only on artificial benchmarks
- Syntactic difference often too imprecise in practice
- Suffers from residual program structure