Università
della
Svizzera
italiana

Faculty of Informatics

Incremental Proof-Based Verification of Compiler Optimizations

Grigory Fedyukovich

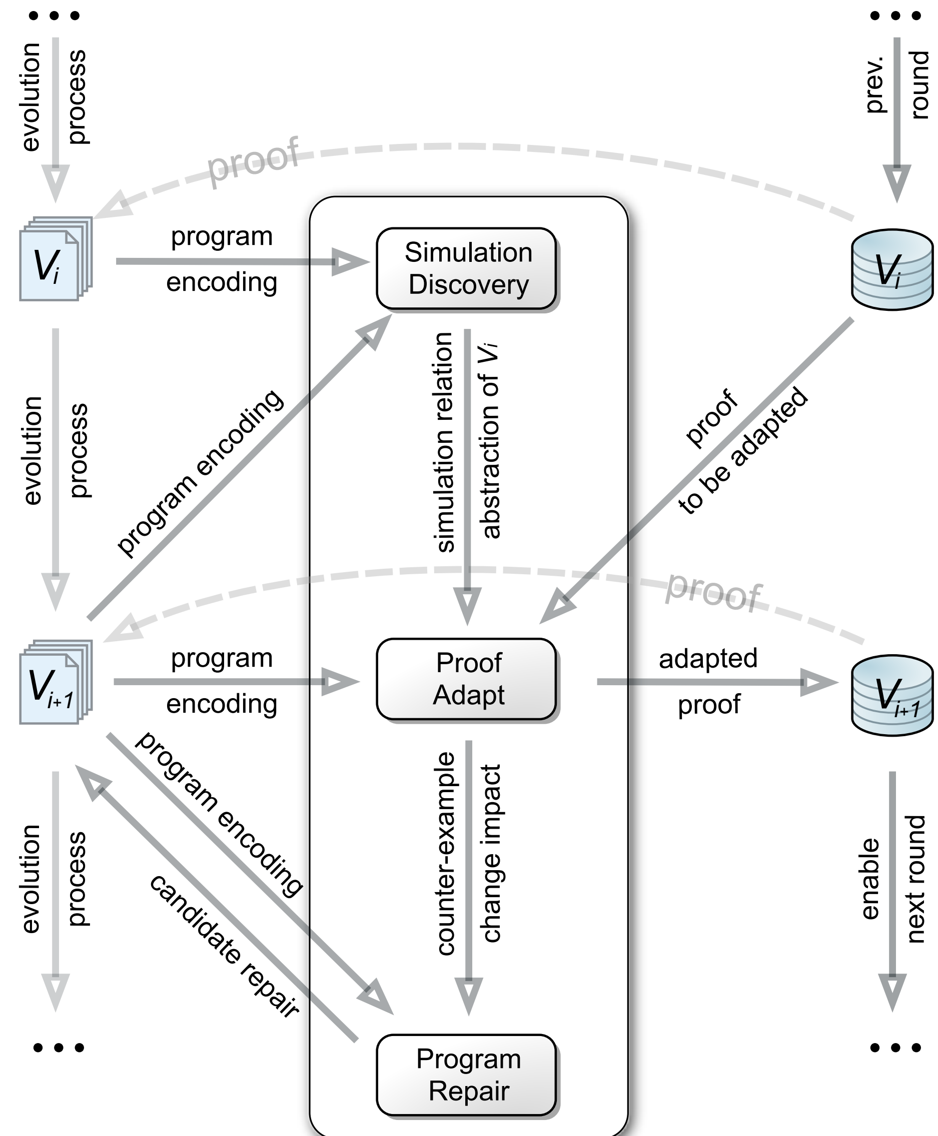
joint work with Arie Gurfinkel and Natasha Sharygina

5 of May, 2015, Attersee, Austria

Big picture

Niagara framework

- Simulation Discovery
  - to synthesize a mapping between variables
- Proof Adapt
  - to migrate the proof certificate across evolution boundaries
  - to obtain a counter-example and a change impact
- Program Repair
  - to provide a hint to the user how to fix the detected bug



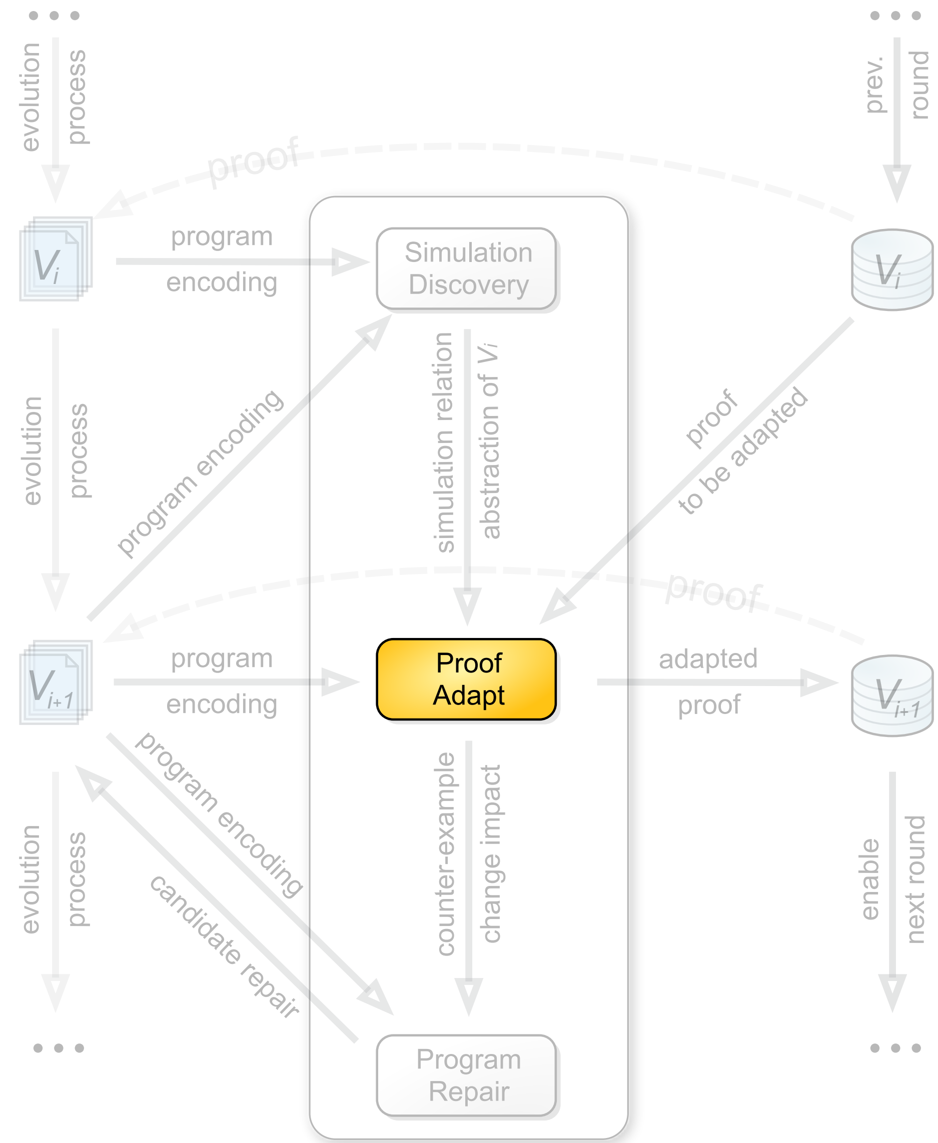
Today

Niagara framework

Università della Svizzera italiana

Faculty of Informatics

- Simulation Discovery
  - to synthesize a mapping between variables
- Proof Adapt
  - to migrate the proof certificate across evolution boundaries
  - to obtain a counter-example and a change impact
- Program Repair
  - to provide a hint to the user how to fix the detected bug



Motivation

- Changes by humans
  - fixing a bug, adding a feature, or improving performance
  - typically, can be localized and verified using existing techniques
  - see our previous work, [*e.g., TACAS' 2013*]
- Optimizations
  - many small changes along the control flow
  - localizing changes impractical / infeasible
  - need for an efficient upgrade checking technique

Our approach

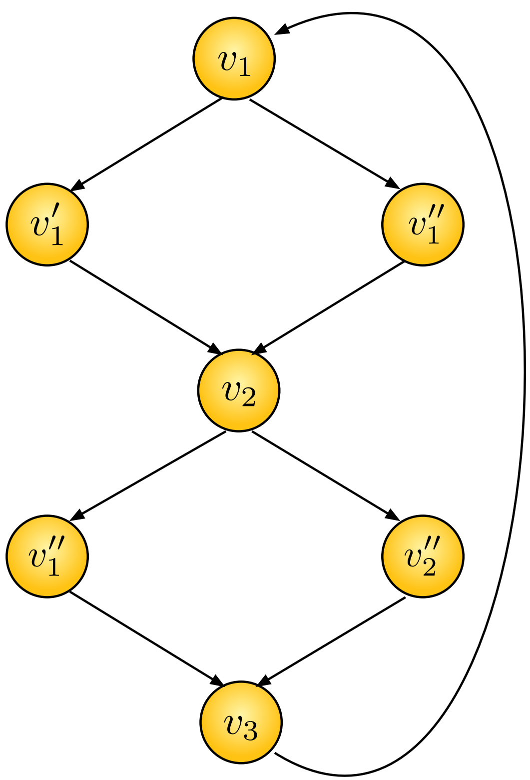
- Original Program
  - given with properties (i.e., assertions)
  - with a certificate of correctness (proof)
- Optimization
  - off-the-shelf compiler optimizer (we use LLVM)
  - "small" changes that do not change loop-structure
- Our approach
  - property-directed equivalence
  - based on Unbounded SMT-based Model Checking with Incremental Inductive Verification

Programs

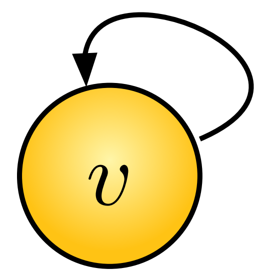
- Program $P = (V, en, err, E, \tau_P)$
  - V is set of cut-points
  - en, err is entry and error locations
  - $E \subseteq V \times V$ is control-flow relation
  - $\tau_P : E \rightarrow Stmt^*$ mapping controls to loop-free program statements
- Cut-Point graph (V, E)
  - generalization of classical Control-Flow graph
  - each edge is a loop-free program path

CFG vs CPG

- Control-Flow-Graph
  - 2-diamond structure with a cycle



- Cut-Point Graph
  - Just a cycle



Hoare triples

- Express the code specification $\{pre\}S\{post\}$
  - $pre, post \in Expr$ are expressions over program variables
  - $S \in Stmt^*$ is some code fragment
- Whenever S starts in a state satisfying pre, if S terminates then it ends in a state satisfying $post$
- Example
  - `{x>=0} x=x+1 {x>0}`

Safety Proof

- A safety proof is a mapping from nodes of CPG to expressions such that
  - each edge is a valid Hoare triple (inductive)
  - error location is mapped to False (safe)
- Formally, $\psi : V \to Expr$

$$\forall (u,v) \in E \cdot \vdash \{\psi(u)\}\ \tau_P(u,v)\ \{\psi(v)\}$$

$$\psi(err) \implies \bot$$

Example

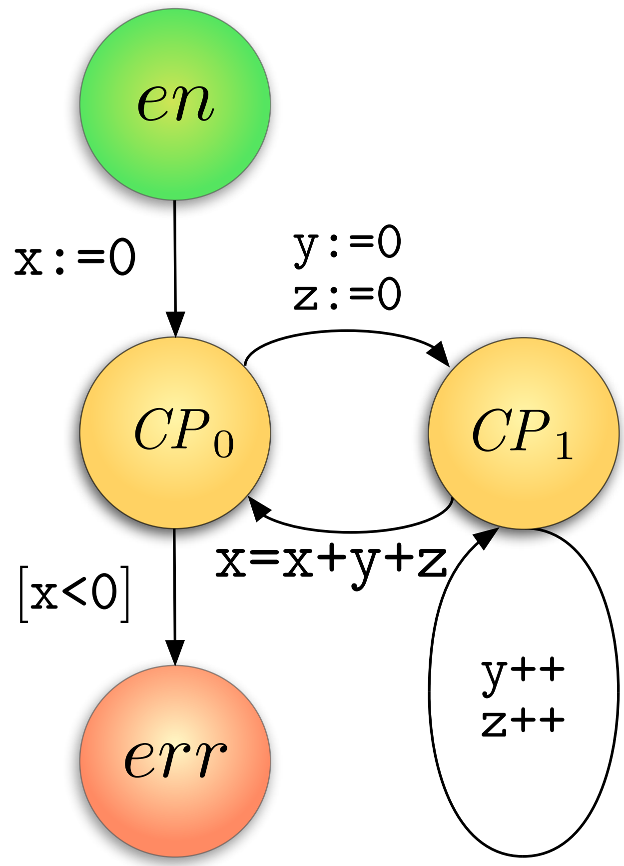
C program with unbounded loops


```
int x = 0;
while(*){
  int y = 0;
  int z = 0;
  while(*){
    y++;
    z++;
  }
  x = x + y + z;
}

assert(x >= 0);
```

assume no overflow

Example
CPG

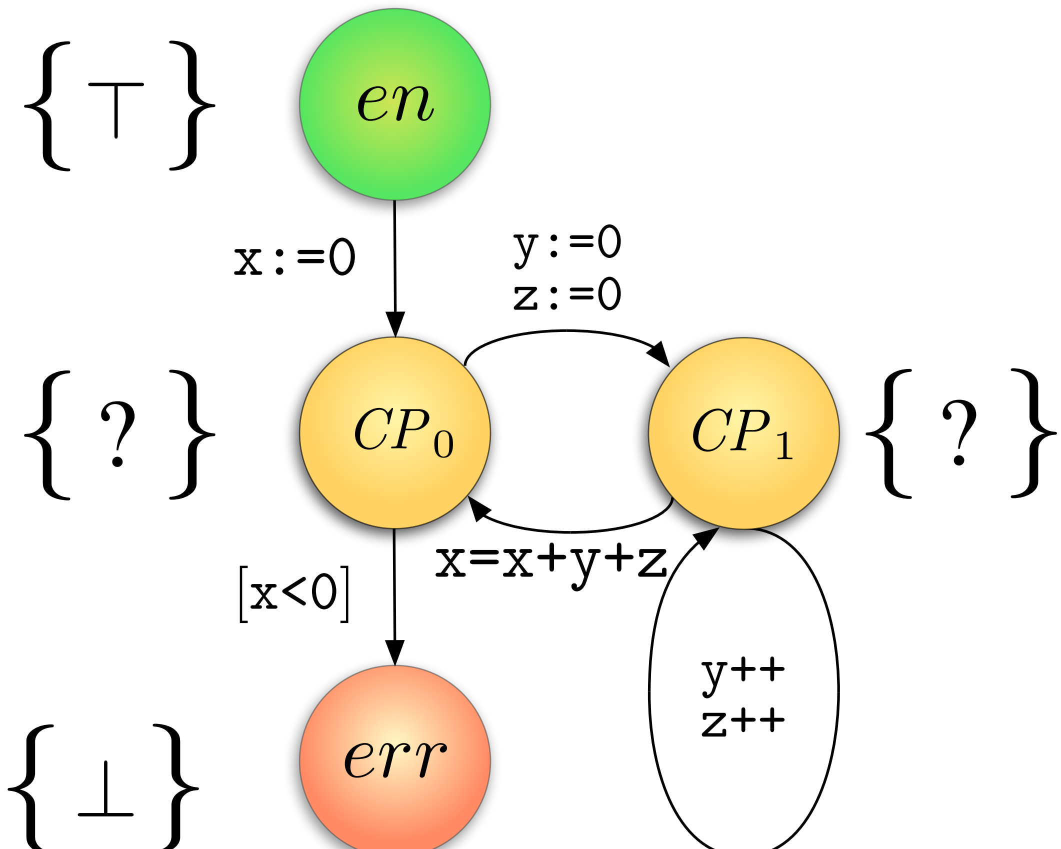
```
int x = 0;
while(*){
  int y = 0;
  int z = 0;
  while(*){
    y++;
    z++;
  }
  x = x + y + z;
}

assert(x >= 0);
```



assume no overflow

Example
CPG labeled by invariants

```
int x = 0;
while(*){
  int y = 0;
  int z = 0;
  while(*){
    y++;
    z++;
  }
  x = x + y + z;
}

assert(x >= 0);
```



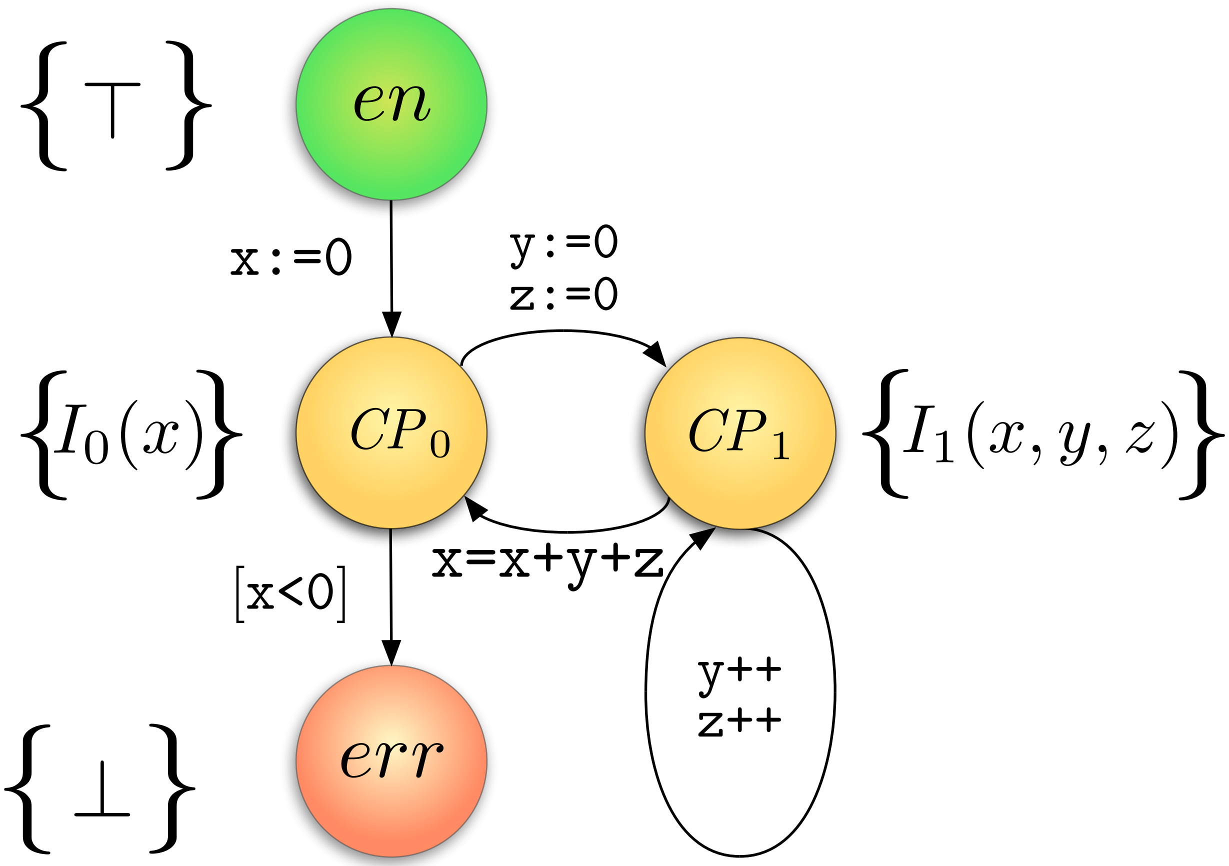
assume no overflow

Example

Safety condition as a Horn system

Università della Svizzera italiana
Faculty of Informatics

$$\begin{cases} \text{True} \land (x' = 0) \rightarrow I_0(x') \\ I_0(x) \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_1(x', y', z') \\ I_1(x, y, z) \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_1(x', y', z') \\ I_1(x, y, z) \land (x' = x + y + z) \rightarrow I_0(x') \\ I_0(x) \land (x < 0) \rightarrow \text{False} \end{cases}$$



Example

Producing safe inductive invariants

System:

$$\text{True} \wedge (x' = 0) \rightarrow I_0(x')$$

$$I_0(x) \wedge (x' = x) \wedge (y' = 0) \wedge (z' = 0) \rightarrow I_1(x', y', z')$$

$$I_1(x, y, z) \wedge (x' = x) \wedge (y' = y+1) \wedge (z' = z+1) \rightarrow I_1(x', y', z')$$

$$I_1(x, y, z) \wedge (x' = x + y + z) \rightarrow I_0(x')$$

$$I_0(x) \wedge (x < 0) \rightarrow \text{False}$$

Solution:

$$I_0(x) = x \geq 0$$

$$I_1(x, y, z) = x + y + z \geq 0$$

Example
Producing safe inductive invariants

System:

$$
\begin{cases}
\text{True} \wedge (x' = 0) \rightarrow I_0(x') \\
I_0(x) \wedge (x' = x) \wedge (y' = 0) \wedge (z' = 0) \rightarrow I_1(x', y', z') \\
I_1(x, y, z) \wedge (x' = x) \wedge (y' = y+1) \wedge (z' = z+1) \rightarrow I_1(x', y', z') \\
I_1(x, y, z) \wedge (x' = x + y + z) \rightarrow I_0(x') \\
I_0(x) \wedge (x < 0) \rightarrow \text{False}
\end{cases}
$$

Solution:

$$I_0(x) = x \geq 0$$

$$I_1(x, y, z) = x + y + z \geq 0$$

These invariants are not factored and therefore are not practical enough

Example

Invariants in CNF

System:

$$\left\{ \begin{array}{l} \text{True} \land (x' = 0) \rightarrow I_0(x') \\ \\ I_0(x) \quad \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{11}(x') \\ I_0(x) \quad \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{12}(y') \\ I_0(x) \quad \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{13}(z') \\ \\ I_{11}(x) \land I_{12}(y) \land I_{13}(z) \quad \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{11}(x') \\ I_{11}(x) \land I_{12}(y) \land I_{13}(z) \quad \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{12}(y') \\ I_{11}(x) \land I_{12}(y) \land I_{13}(z) \quad \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{13}(z') \\ \\ I_{11}(x) \land I_{12}(y) \land I_{13}(z) \quad \land (x' = x + y + z) \rightarrow I_0(x') \\ \\ I_0(x) \land (x < 0) \rightarrow \text{False} \end{array} \right.$$

Example

Invariants in CNF

Università della Svizzera italiana
Faculty of Informatics

System:

$$\text{True} \land (x' = 0) \rightarrow I_0(x')$$

$$I_0(x) \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{11}(x')$$
$$I_0(x) \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{12}(y')$$
$$I_0(x) \land (x' = x) \land (y' = 0) \land (z' = 0) \rightarrow I_{13}(z')$$

$$I_{11}(x) \land I_{12}(y) \land I_{13}(z) \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{11}(x')$$
$$I_{11}(x) \land I_{12}(y) \land I_{13}(z) \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{12}(y')$$
$$I_{11}(x) \land I_{12}(y) \land I_{13}(z) \land (x' = x) \land (y' = y+1) \land (z' = z+1) \rightarrow I_{13}(z')$$

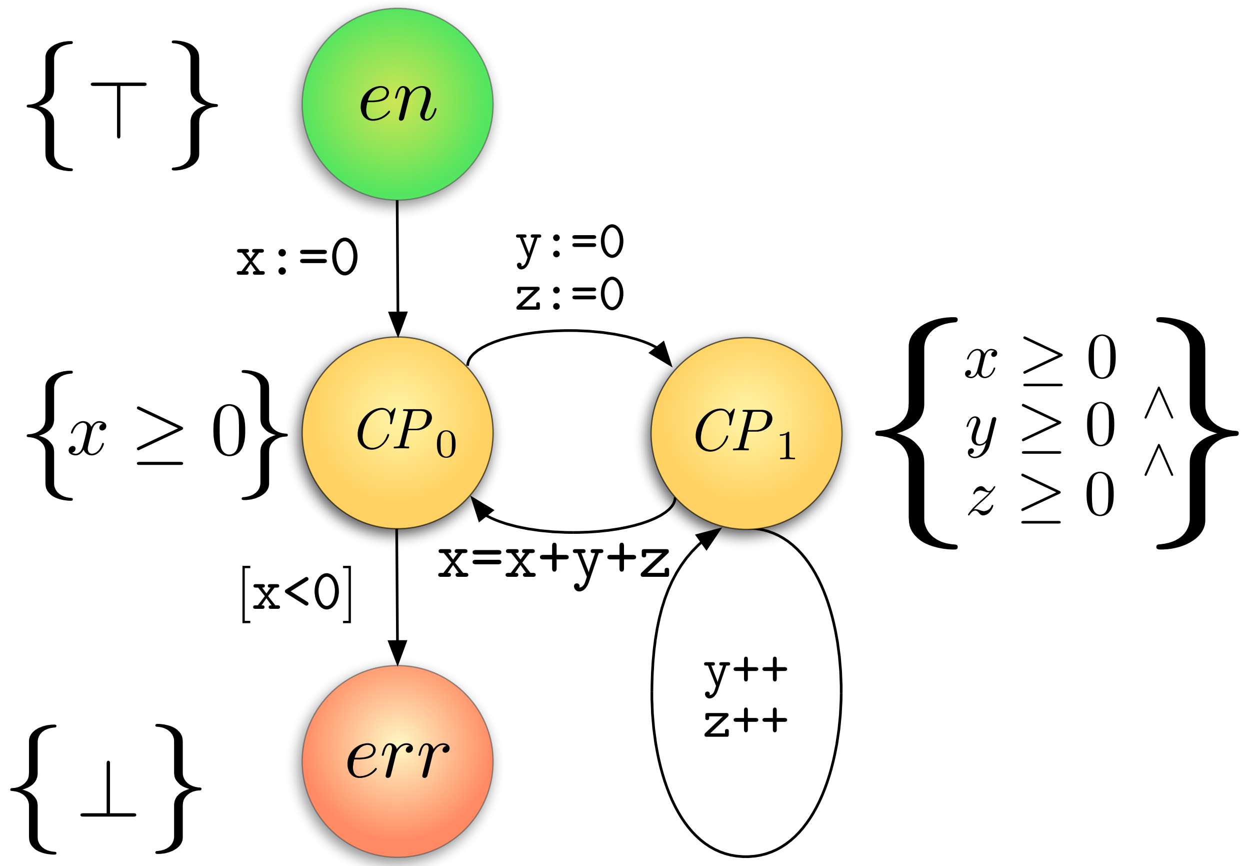
$$I_{11}(x) \land I_{12}(y) \land I_{13}(z) \land (x' = x + y + z) \rightarrow I_0(x')$$

$$I_0(x) \land (x < 0) \rightarrow \text{False}$$

Solution:

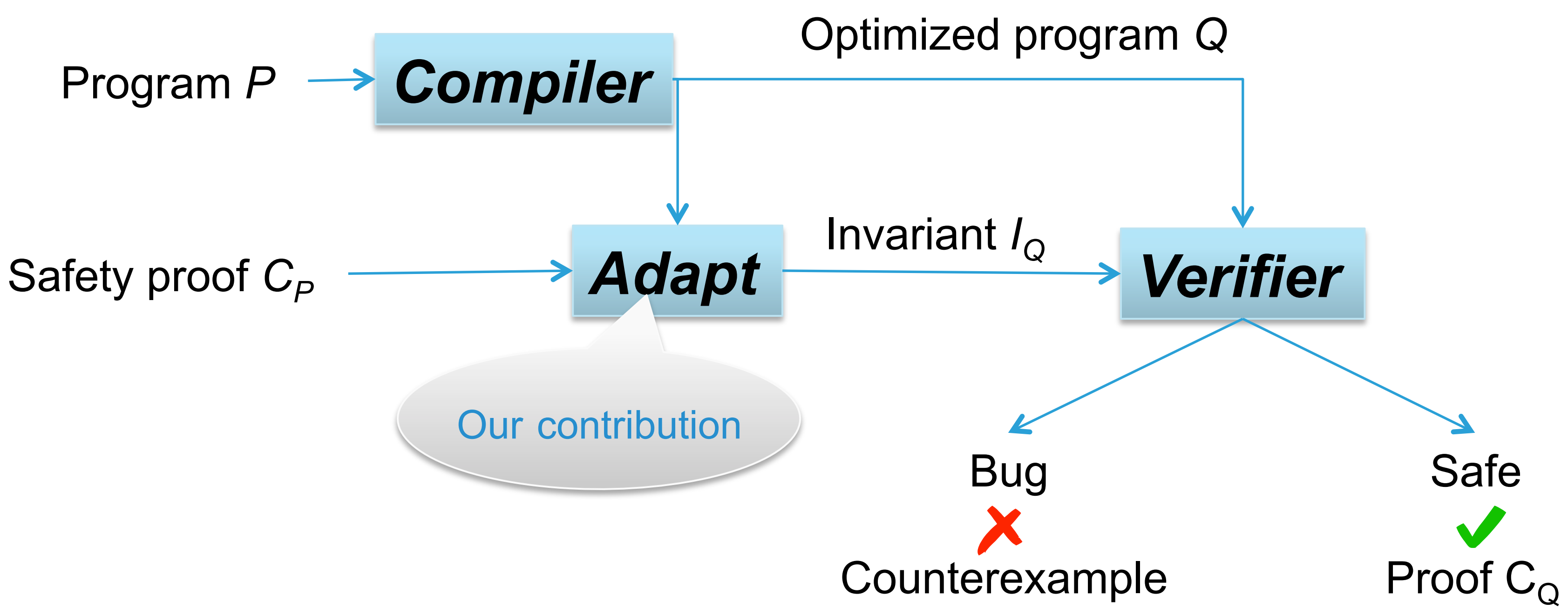
$$I_0(x) = x \geq 0$$
$$I_{11}(x) = x \geq 0$$
$$I_{12}(y) = y \geq 0$$
$$I_{13}(z) = z \geq 0$$

Example
Safety proof

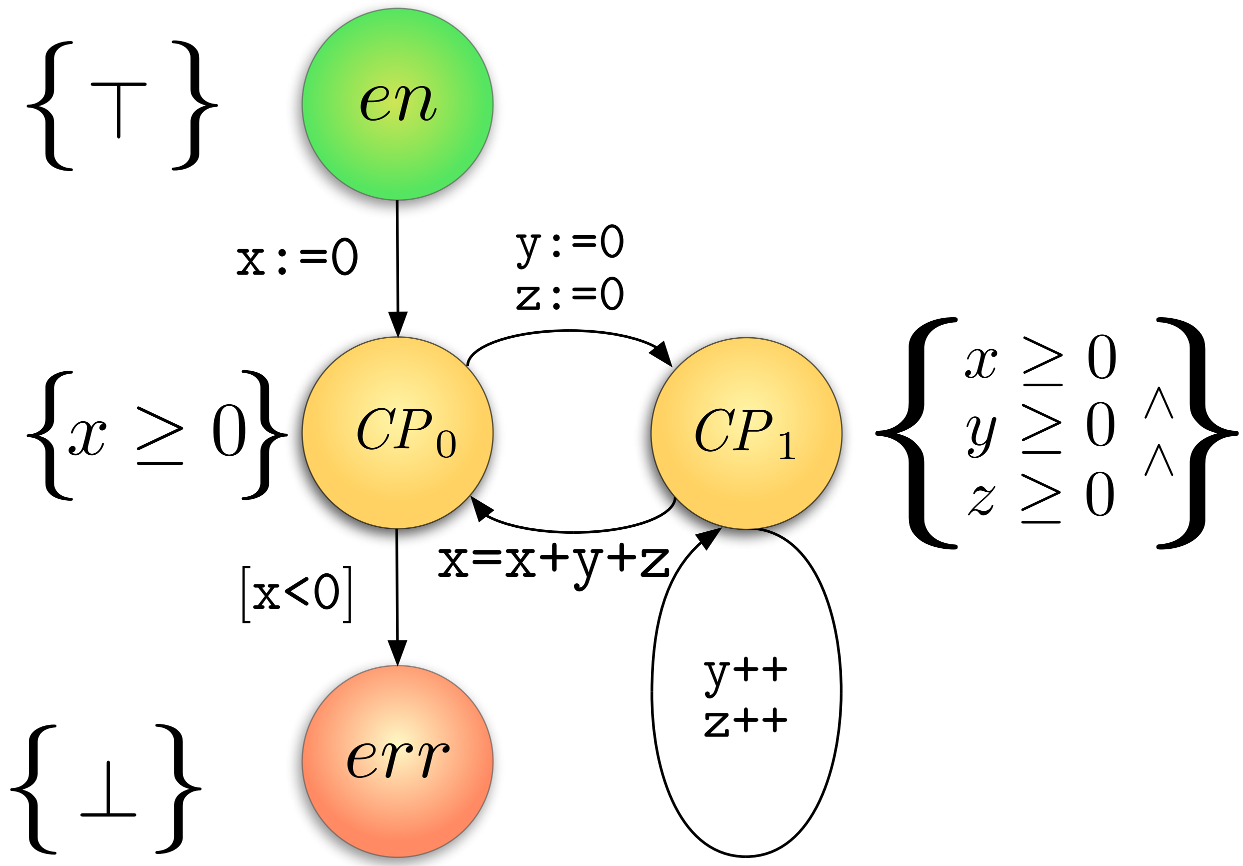


Our approach

- Adapt verification certificate across simple complier optimizations

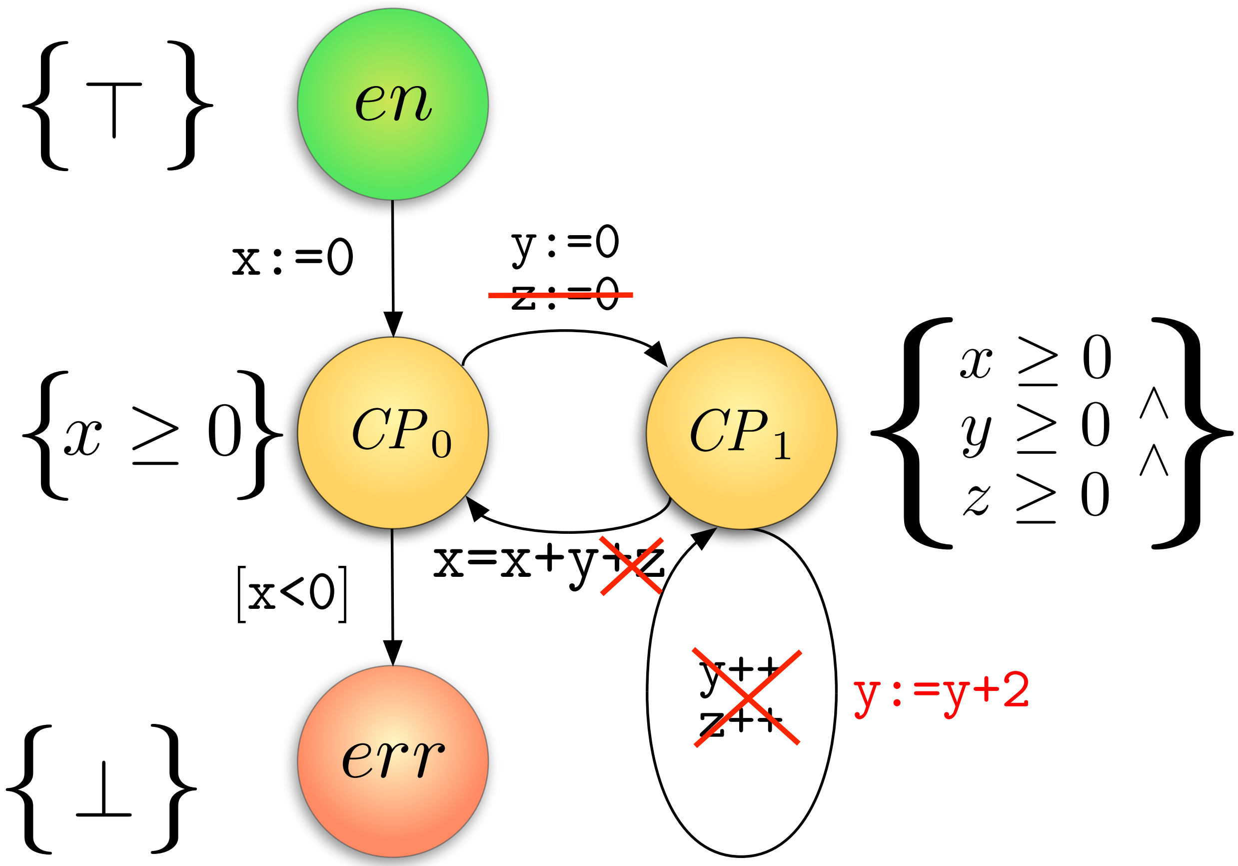


Example
Optimization



Example

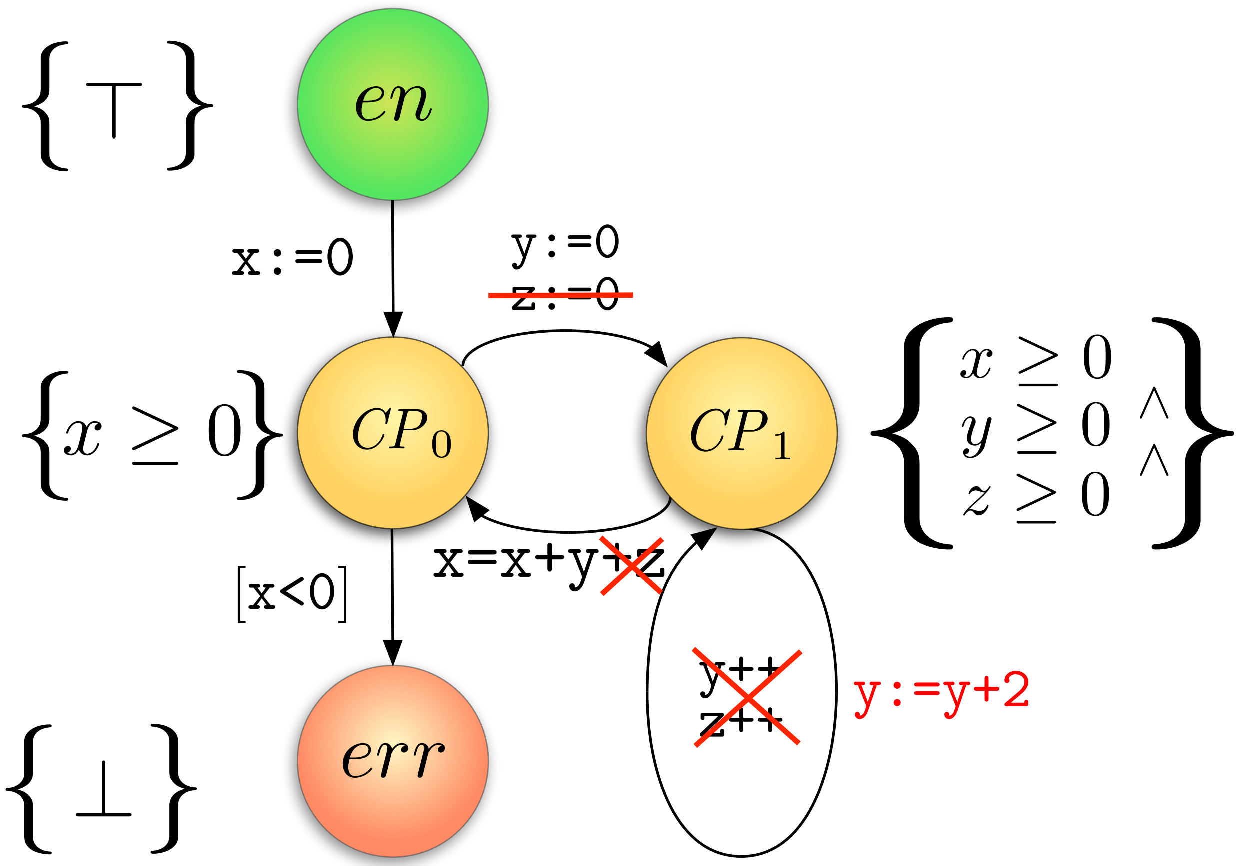
Optimization removes variable **z**



There is a need to adapt the proof

Example

Re-checking Hoare triple for $CP_1 \rightarrow CP_1$

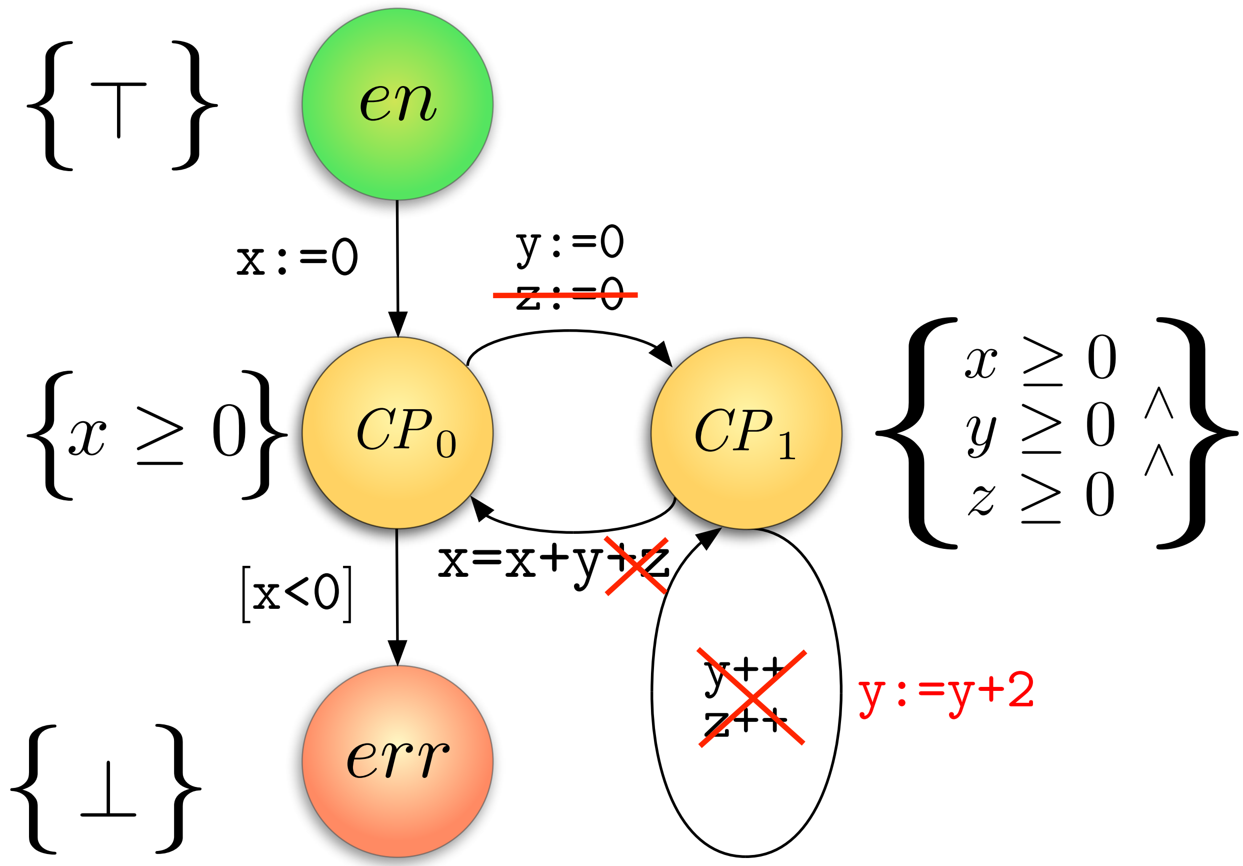


Is is still valid?

$$(x \geq 0) \wedge (y \geq 0) \wedge (z \geq 0) \wedge (y' = y + 2) \wedge (x' = x) \rightarrow (x' \geq 0) \wedge (y' \geq 0) \wedge (z' \geq 0)$$

Example

Re-checking Hoare triple for $CP_1 \rightarrow CP_1$

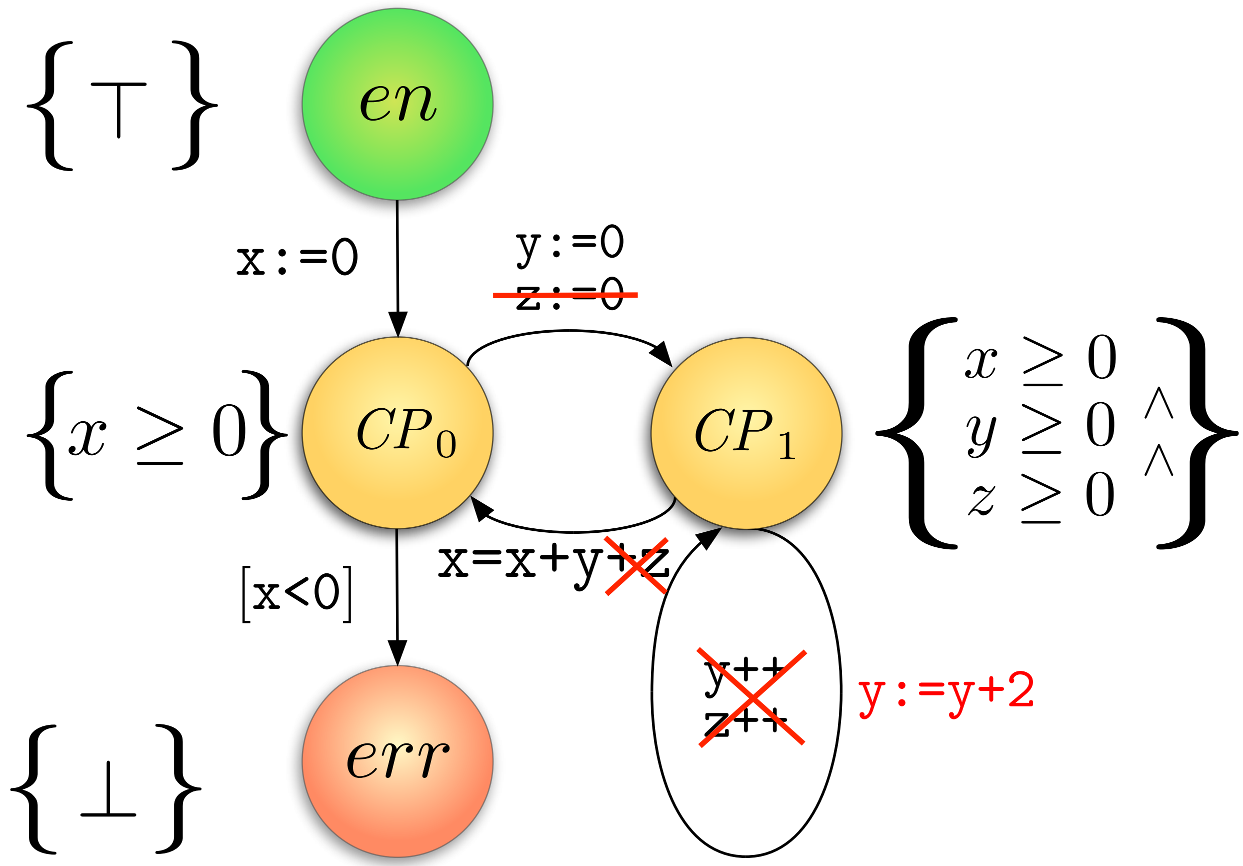


Is is still unsatisfiable?

$$(x \geq 0) \wedge (y \geq 0) \wedge (z \geq 0) \wedge (y' = y + 2) \wedge (x' = x) \wedge \big(\sim (x' \geq 0) \wedge (y' \geq 0) \wedge (z' \geq 0)\big)$$

Example

Re-checking Hoare triple for $CP_1 \rightarrow CP_1$



Is is still unsatisfiable? No!

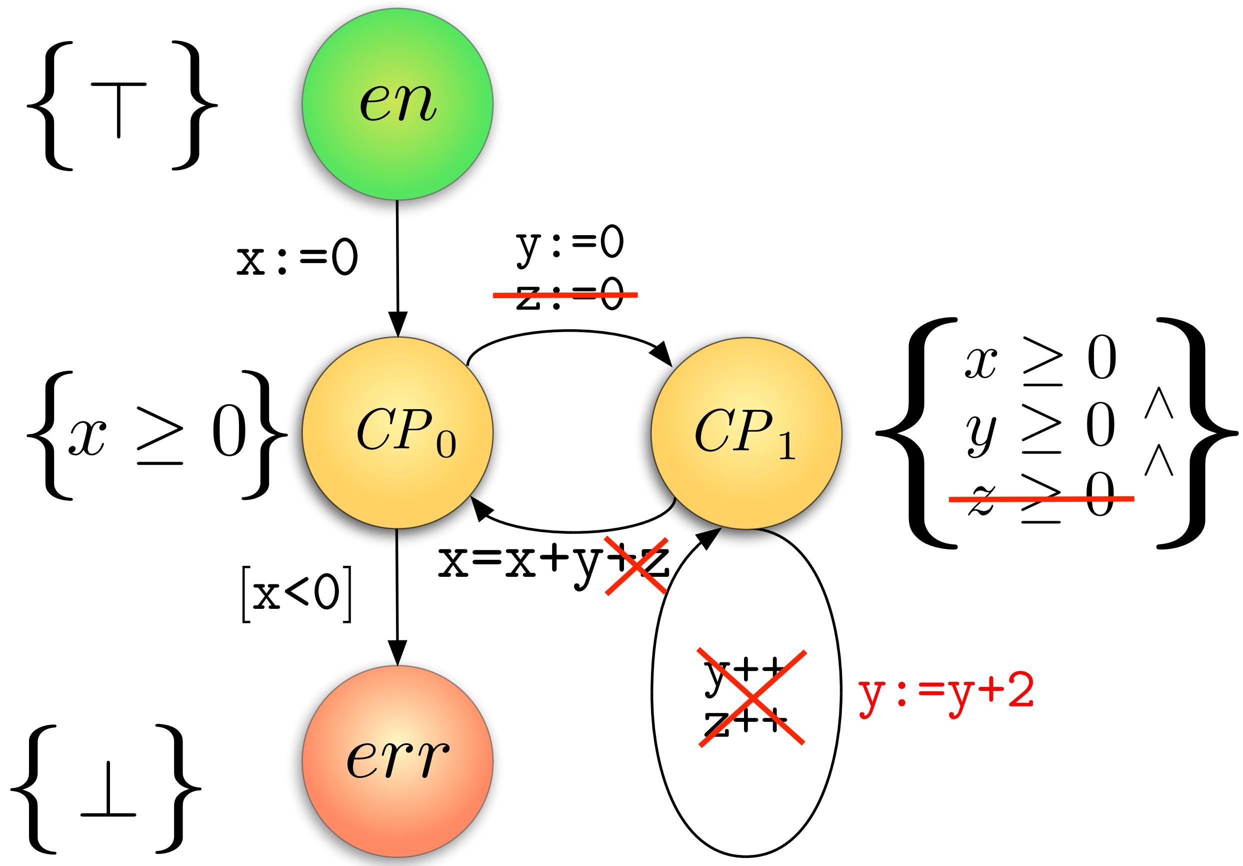
$$(x \geq 0) \wedge (y \geq 0) \wedge (z \geq 0) \wedge (y' = y + 2) \wedge (x' = x) \wedge \left(\sim (x' \geq 0) \wedge (y' \geq 0) \wedge (z' \geq 0)\right)$$

Weakening the invariants

Università della Svizzera italiana
Faculty of Informatics

- Satisfiable formula
  - $(x \geq 0) \wedge (y \geq 0) \wedge (z \geq 0) \wedge (y' = y + 2) \wedge (x' = x) \wedge (\sim (x' \geq 0) \wedge (y' \geq 0) \wedge (z' \geq 0))$
- Introduce assumptions
  - $(a \rightarrow x \geq 0)$
  - $(b \rightarrow y \geq 0)$
  - $(c \rightarrow z \geq 0)$
- (Re)-solve with assumptions and get a model M
  - $(a \rightarrow x \geq 0) \wedge (b \rightarrow y \geq 0) \wedge (c \rightarrow z \geq 0)$
    $\wedge (y' = y + 2) \wedge (x' = x) \wedge$
    $(\sim (a \rightarrow x' \geq 0) \wedge (b \rightarrow y' \geq 0) \wedge (c \rightarrow z' \geq 0))$
- Evaluate a, b, c in the model M, block satisfied assumptions and repeat the check
  - $M \not\models a$
  - $M \not\models b$
  - $M \models c$
- Finally, we calculated Maximal Inductive Subset

Example
Inductive weakening the invariants



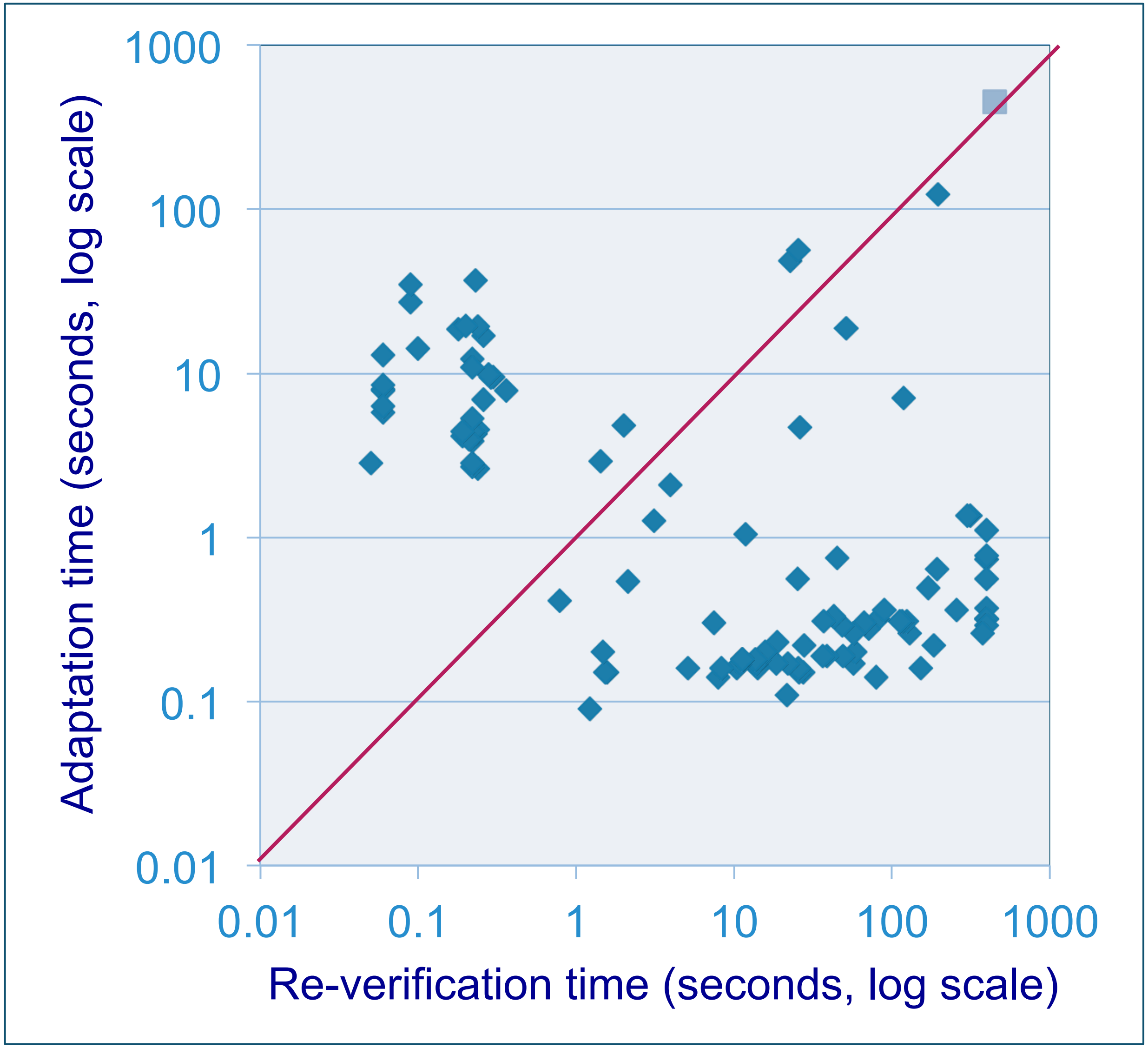
Finally, we weaken the invariant
Need to check other edges, which is trivial in this case

Evaluation

- Tools
  - LLVM compiler + `instcombine` optimization pass
  - UFO model checker (Z3/PDR engine)

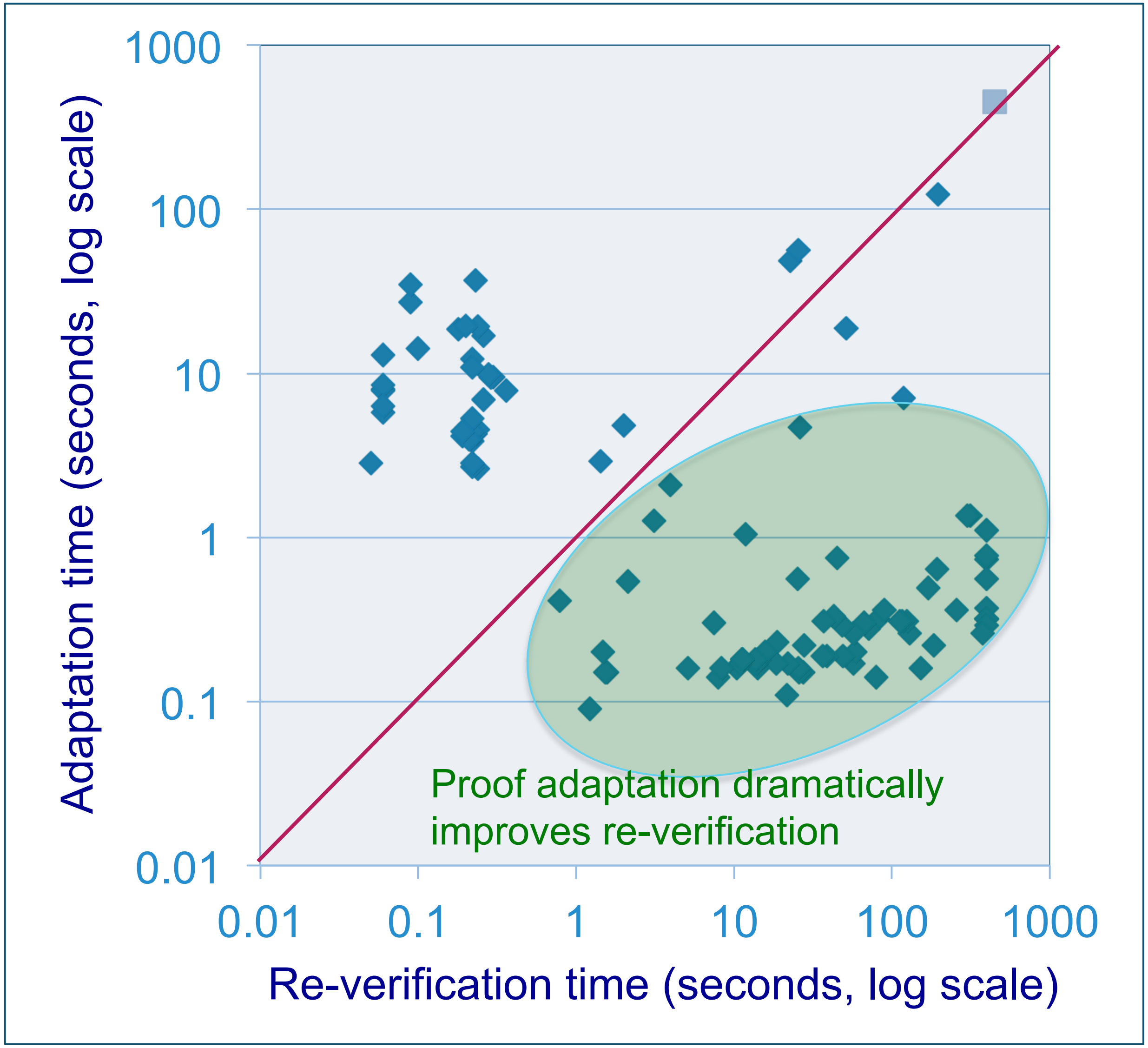
- Benchmarks
  - SV-COMP (0.3 – 5K LOC)
  - 397 Safe C programs
    - 108 – non-trivial verification time
    - 102 – significant time for re-verification (> 1 sec)
    - 67 – candidate invariant is already safe
    - 52 – proof adaptation is an order of magnitude faster

Evaluation


1000
100
10
1
0.1
0.01
Adaptation time (seconds, log scale)
0.01
0.1
1
10
100
1000
Re-verification time (seconds, log scale)

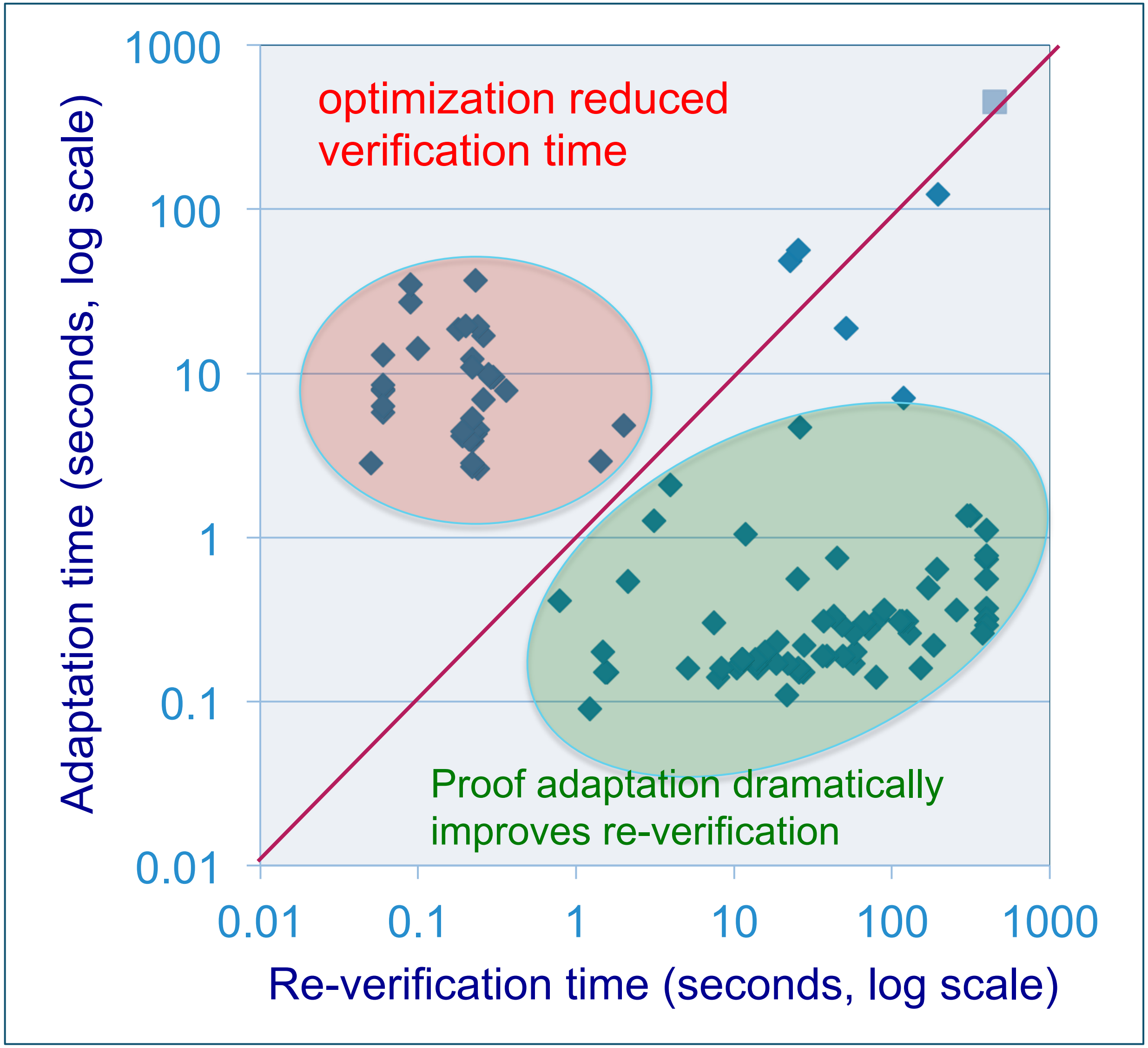
Evaluation

Università
della
Svizzera
italiana
Faculty
of Informatics



Evaluation

Università della Svizzera italiana
Faculty of Informatics



Next Steps of the Niagara project

- Extend to optimizations that modify CPG
  - searching for the least common unrolling
- Automated discovery of a possible simulation
  - using simulation relation as a mapping between variables
- Symbolic verification of simulation relation
  - to check stronger equivalence property
- Explain/repair the detected bugs
  - using the change impact

Thank you!

Emergency slides

benchmark (name)	oV (sec)	optVerify mkInd (sec)	optVerify proof candidate	optVerify Verify (sec)	speedup (X)	uV (sec)
s3_srvr.blast.15_safe.i.cil.o3.bc	248.2	0.23	safe	0.06	∞	*timeout*
s3_srvr_7_safe.cil.o3.bc	324	0.25	safe	0.07	∞	*timeout*
s3_srvr_2_safe.cil.o3.bc	281.95	0.24	safe	0.08	∞	*timeout*
s3_srvr_3_safe.cil.o3.bc	398.65	0.27	safe	0.1	∞	*timeout*
s3_srvr.blast.13_safe.i.cil.o3.bc	270.63	0.44	safe	0.12	∞	*timeout*
token_ring.08_safe.cil.o3.bc	322.45	0.6	safe	0.18	∞	*timeout*
pc_sfifo_2_safe.cil.o3.bc	308.8	0.51	safe	0.22	∞	*timeout*
token_ring.09_safe.cil.o3.bc	468.19	0.82	safe	0.28	∞	*timeout*
s3_srvr_8_safe.cil.o3.bc	243.41	0.2	safe	0.06	1450.23	377.06
s3_srvr.blast.16_safe.i.cil.o3.bc	60.04	0.12	safe	0.04	953	152.48
s3_srvr.blast.10_safe.i.cil.o3.bc	77.82	0.18	safe	0.04	833.5	183.37
s3_srvr.blast.11_safe.i.cil.o3.bc	280.88	0.28	safe	0.08	715.11	257.44
s3_srvr.blast.16_safe.i.cil.o0.bc	14.92	0.1	safe	0.04	571.57	80.02
s3_srvr.blast.06_safe.i.cil.o3.bc	181.96	0.2	safe	0.06	499.96	129.99
s3_clnt_3_safe.cil.o3.bc	94.88	0.23	safe	0.08	401	124.31
s3_srvr.blast.01_safe.i.cil.o3.bc	320.2	0.25	safe	0.06	375.84	116.51
s3_srvr.blast.09_safe.i.cil.o0.bc	92.75	0.25	safe	0.06	361.23	111.98
token_ring.06_safe.cil.o3.bc	136.59	0.37	safe	0.12	346.78	169.92
s3_srvr.blast.15_safe.i.cil.o0.bc	22.69	0.13	safe	0.04	337.88	57.44
token_ring.07_safe.cil.o3.bc	159.64	0.48	safe	0.16	301.69	193.08
s3_srvr.blast.02_safe.i.cil.o0.bc	54.64	0.16	safe	0.04	294.3	58.86
s3_srvr.blast.12_safe.i.cil.o0.bc	19.75	0.14	safe	0.05	257.37	48.9
s3_srvr.blast.13_safe.i.cil.o0.bc	91.51	0.26	safe	0.06	256.13	81.96
s3_srvr.blast.11_safe.i.cil.o0.bc	52.65	0.22	safe	0.06	256.04	71.69
s3_clnt.blast.03_safe.i.cil.o3.bc	75.36	0.28	safe	0.08	249.61	89.86
token_ring.06_safe.cil.o0.bc	265.38	1.18	safe	0.18	231.2	314.43
s3_srvr.blast.07_safe.i.cil.o0.bc	57.76	0.22	safe	0.08	221.7	66.51
token_ring.07_safe.cil.o0.bc	262.47	1.22	safe	0.14	221.35	301.04
s3_clnt_2_safe.cil.o3.bc	49.67	0.2	safe	0.06	219.38	57.04
s3_clnt_1_safe.cil.o3.bc	32.6	0.15	safe	0.04	203.47	38.66
s3_srvr_4_safe.cil.o3.bc	19.05	0.08	safe	0.03	197.55	21.73
s3_srvr.blast.14_safe.i.cil.o0.bc	44.75	0.15	safe	0.04	191.47	36.38
s3_srvr_3_safe.cil.o0.bc	19.91	0.11	safe	0.04	182.93	27.44
s3_srvr.blast.12_safe.i.cil.o3.bc	23.13	0.12	safe	0.03	173.4	26.01
s3_clnt.blast.02_safe.i.cil.o3.bc	41.78	0.23	safe	0.06	167.52	48.58
s3_srvr.blast.01_safe.i.cil.o0.bc	13.64	0.11	safe	0.05	160.5	25.68
s3_srvr.blast.14_safe.i.cil.o3.bc	192.93	0.28	safe	0.05	130.91	43.2
s3_clnt_1_safe.cil.o0.bc	11.55	0.13	safe	0.04	129.47	22.01

benchmark (name)	oV (sec)	optVerify			speedup (X)	uV (sec)
		mkInd (sec)	proof candidate	Verify (sec)		
elevator_spec2_product01_safe.cil.o3.bc	9.54	2.54	ind	0.1	0.09	0.24
elevator_spec2_product17_safe.cil.o3.bc	8.69	2.76	ind	0.08	0.08	0.22
elevator_spec2_product03_safe.cil.o3.bc	8.64	2.62	ind	0.08	0.08	0.22
elevator_spec9_product27_safe.cil.o3.bc	10.34	3.75	ind	0.1	0.06	0.22
elevator_spec2_product19_safe.cil.o3.bc	8.84	4.48	ind	0.08	0.05	0.24
elevator_spec2_product27_safe.cil.o3.bc	10.36	4.03	ind	0.13	0.05	0.19
elevator_spec2_product25_safe.cil.o3.bc	10.7	4.04	ind	0.13	0.05	0.2
elevator_spec9_product09_safe.cil.o3.bc	11.48	4.3	ind	0.14	0.05	0.22
elevator_spec2_product09_safe.cil.o3.bc	10.4	4.13	ind	0.14	0.05	0.23
elevator_spec2_product29_safe.cil.o3.bc	19.62	7.7	ind	0.18	0.05	0.36
elevator_spec9_product25_safe.cil.o3.bc	10.4	3.76	ind	0.18	0.05	0.21
elevator_spec2_product11_safe.cil.o3.bc	8.94	5.18	ind	0.12	0.04	0.22
elevator_spec9_product11_safe.cil.o3.bc	7.71	4.33	ind	0.12	0.04	0.19
elevator_spec2_product31_safe.cil.o3.bc	17.73	6.72	ind	0.16	0.04	0.26
elevator_spec9_product31_safe.cil.o3.bc	26.38	9.72	ind	0.12	0.03	0.28
elevator_spec2_product23_safe.cil.o3.bc	26.54	9.29	ind	0.15	0.03	0.29
elevator_spec2_product21_safe.cil.o3.bc	18.29	9.37	ind	0.15	0.03	0.3
elevator_spec1_product25_safe.cil.o3.bc	34.56	2.78	ind	0.08	0.02	0.05
elevator_spec9_product29_safe.cil.o3.bc	23.7	10.81	ind	0.14	0.02	0.22
elevator_spec1_product03_safe.cil.o0.bc	231.96	11.9	ind	0.39	0.02	0.22
elevator_spec1_product27_safe.cil.o0.bc	298.7	16.6	ind	0.48	0.02	0.26
elevator_spec1_product19_safe.cil.o3.bc	50.83	8.46	ind	0.07	0.01	0.06
elevator_spec1_product17_safe.cil.o3.bc	33.41	5.68	ind	0.07	0.01	0.06
elevator_spec1_product27_safe.cil.o3.bc	31.42	6.26	ind	0.07	0.01	0.06
elevator_spec1_product01_safe.cil.o3.bc	94	7.9	ind	0.08	0.01	0.06
elevator_spec1_product03_safe.cil.o3.bc	68.34	7.77	ind	0.08	0.01	0.06
elevator_spec1_product31_safe.cil.o3.bc	123.46	14.01	ind	0.12	0.01	0.1
elevator_spec1_product19_safe.cil.o0.bc	200.74	36.72	ind	0.29	0.01	0.23
elevator_spec1_product11_safe.cil.o0.bc	246.57	18.87	ind	0.33	0.01	0.24
elevator_spec1_product17_safe.cil.o0.bc	410.57	19.2	ind	0.4	0.01	0.2
elevator_spec1_product01_safe.cil.o0.bc	376.01	18.1	ind	0.44	0.01	0.18
elevator_spec1_product11_safe.cil.o3.bc	70.06	12.81	ind	0.09	0	0.06
elevator_spec1_product23_safe.cil.o3.bc	333.93	34.86	ind	0.13	0	0.09
elevator_spec1_product21_safe.cil.o3.bc	217.8	26.96	ind	0.2	0	0.09

Algorithm 1: $\textsc{optVerify}(P, \psi, Q)$

Input: P, ψ, Q

Output: res, φ

```
σ ← guessMap(P, Q)
π ← mkInd(ψσ, Q, P)
res, φ ← Verify(Q, π)
return ⟨res, φ⟩
```

Algorithm 2: $\textsc{mkInd}(\psi, Q, P)$

Input: ψ, P, Q

Output: π

$\pi \leftarrow \psi$; $W \leftarrow \{(u,v) \in E \mid \tau_P(e) \neq \tau_Q(e)\}$;

while $W \neq \emptyset$ **do**

  $(u,v) \leftarrow \textsc{getWtoSmallestEdge}(W)$;

  $pre \leftarrow \pi(u)$; $post \leftarrow \pi(v)$;

  if $(\vdash \{pre\}\ \tau_Q(u,v)\ \{post\})$ **then** $W \leftarrow W \setminus \{(u,v)\}$;

  else

    $\pi(v) \leftarrow \textsc{weakPost}(pre, \tau_Q(u,v), post)$;

    $W \leftarrow (W \setminus \{(u,v)\}) \cup \{(v,x) \in E \mid x \in V\}$

return π

Algorithm 3: $\textsc{weakPost}(\mathit{pre}, S, \mathit{post})$

Input: $\mathit{pre}, \mathit{post} \in \mathit{Expr}$; $\mathit{post} = \bigwedge_{i=0}^{n} \ell_i$; $S \in \mathit{Stmt}^*$

Output: $\mathit{post}' \in \mathit{Expr}$, such that $\vdash \{\mathit{pre}\}\ S\ \{\mathit{post}'\}$ is valid

let $\{x_i \mid 0 \leq i \leq n\}$ be fresh Boolean variables; $U \leftarrow \{0, \ldots, n\}$;

$\mathit{vc} \leftarrow \mathit{pre} \wedge \textsc{mkVC}(S) \wedge \neg(\bigwedge_{i=0}^{n}(x_i \rightarrow \ell_i))$;

$\textsc{SmtAssert}(\mathit{vc})$;

while $\textsc{SmtSolve}(\,) = \mathrm{SAT}$ **do**

    $M \leftarrow \textsc{SmtModel}(\,)$;

    foreach $\{0 \leq i \leq n \mid M \models x_i\}$ **do** $\textsc{SmtAssert}(\neg x_i); U \leftarrow U \setminus \{i\}$;

$\mathit{post}' \leftarrow \wedge\{\ell_i \mid i \in U\}$;

return post'