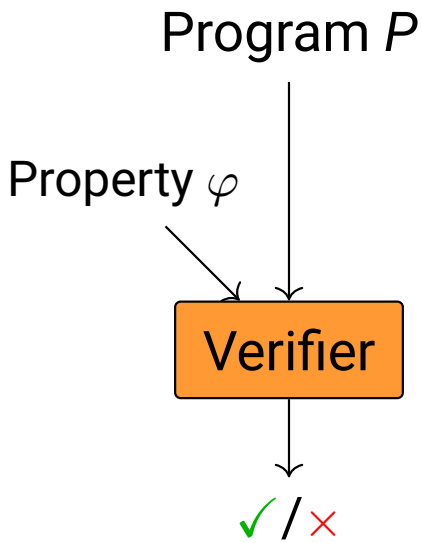
TECHNISCHE UNIVERSITÄT DARMSTADT

Taking Advantage of Properties and Dataflow in Difference Verification

Marie-Christine Jakobs
joint work with Tim Pollandt

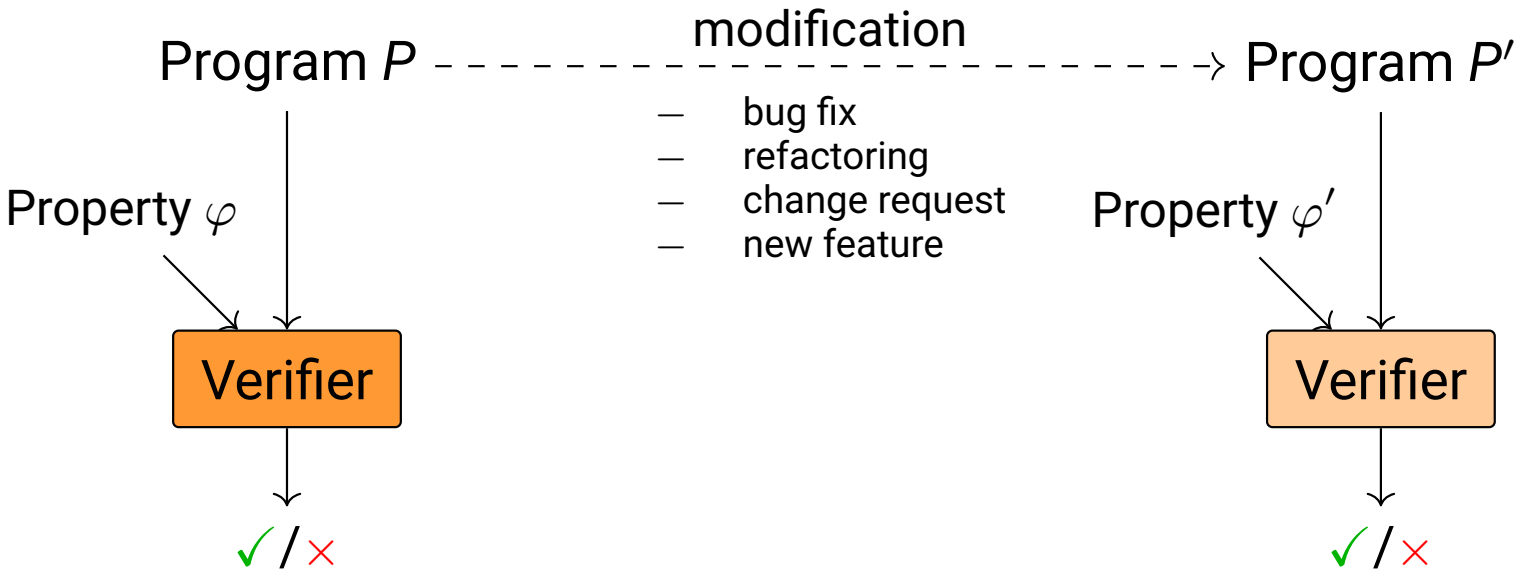
The Modification Challenge

TECHNISCHE UNIVERSITÄT DARMSTADT

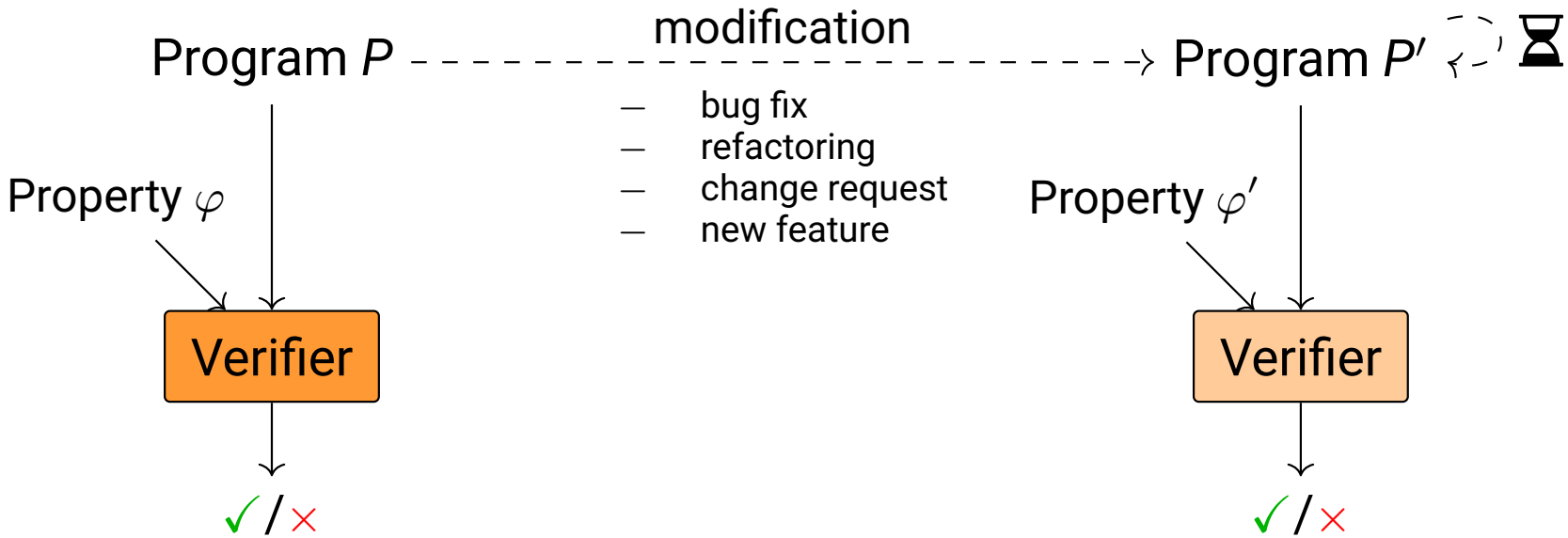


The Modification Challenge

TECHNISCHE UNIVERSITÄT DARMSTADT



The Modification Challenge



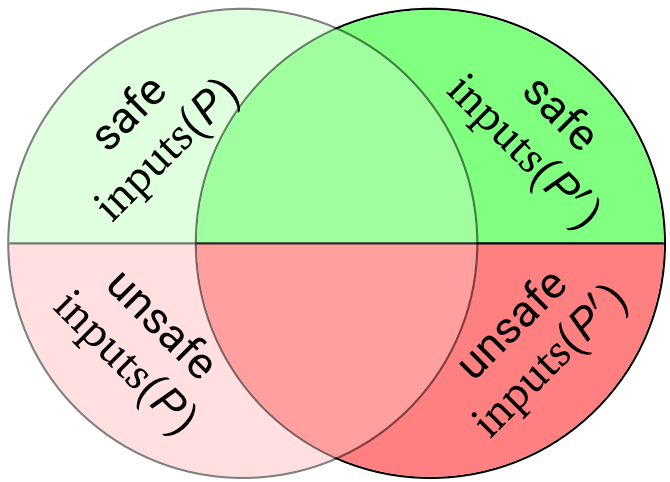
Challenge:

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

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

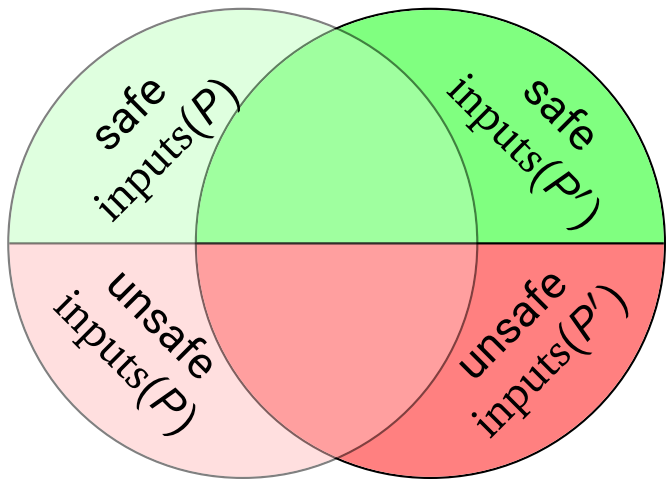
Suggested Solution: Only Look for New Bugs

TECHNISCHE UNIVERSITÄT DARMSTADT



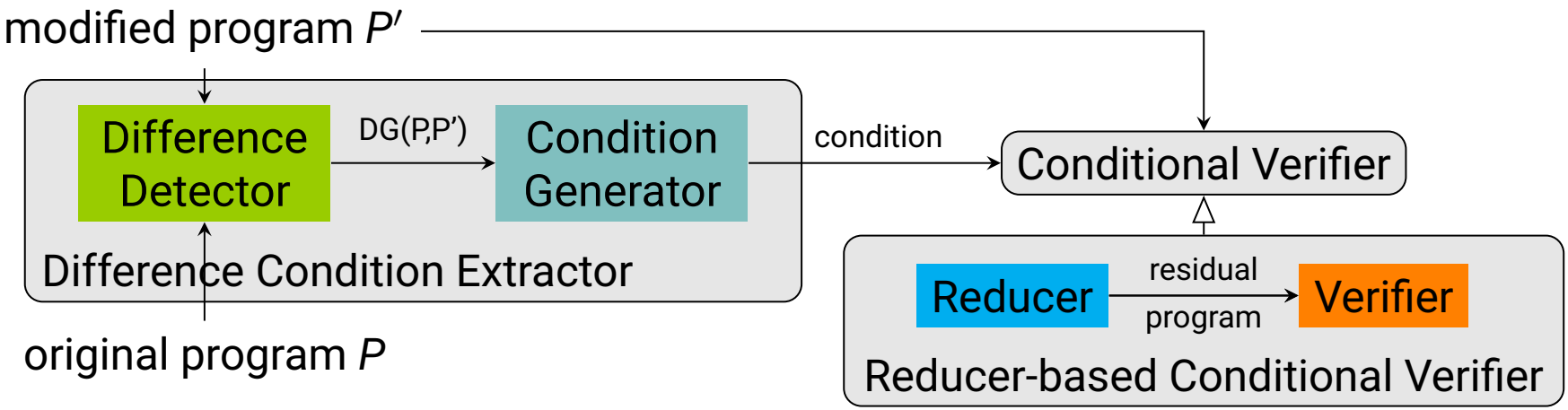
Suggested Solution: Only Look for New Bugs

TECHNISCHE UNIVERSITÄT DARMSTADT



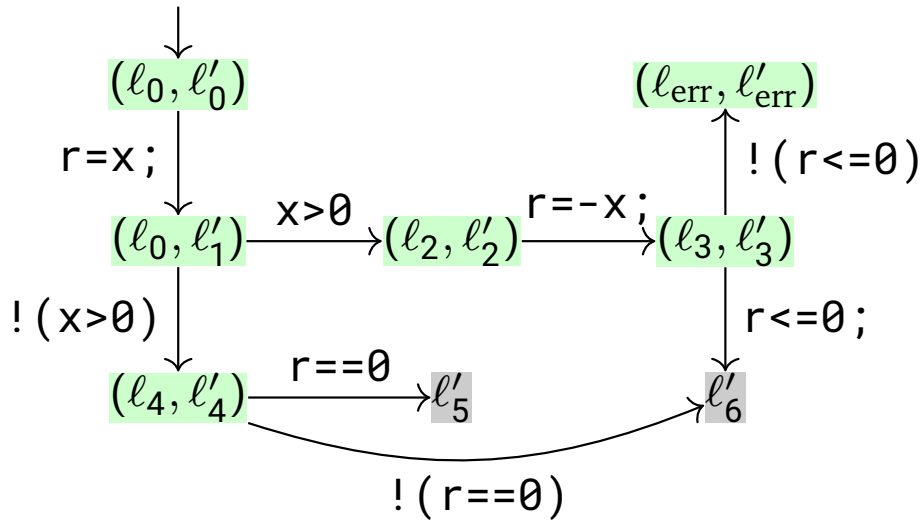
Approach: Overapproximate and then verify (syntactical) paths with new bugs

Difference Verifier



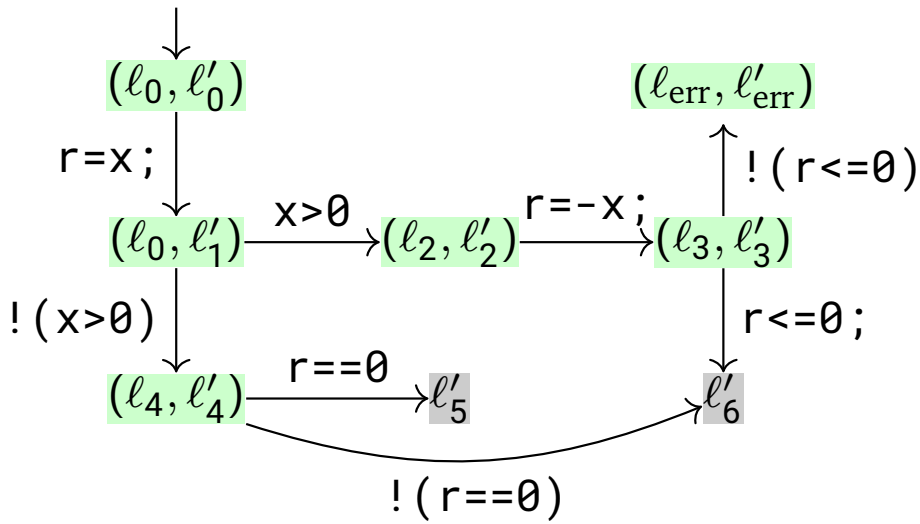
- Difference condition extractor overapproximates relevant paths
  - Difference detector determine relevant paths
  - Condition generator encodes paths into conditions
- Conditional verifier restricts analysis to identified paths

Difference Graphs Characterizing New Bugs



- Paths relate to execution paths of the modified program
- Any path that matches to a prefix of a new bug must be extendable s.t.
  1. it stil matches to a prefix of the new bug
  2. ends in a relevant (Δ-)node (grey nodes)

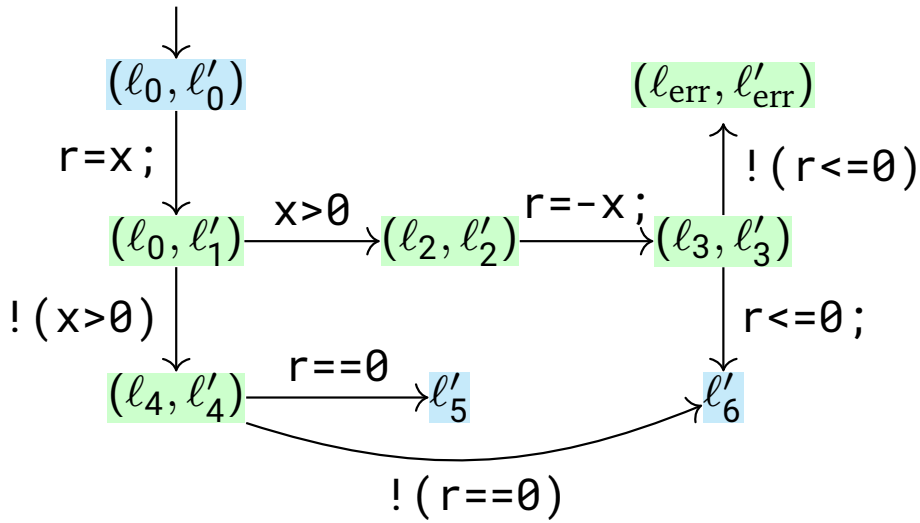
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

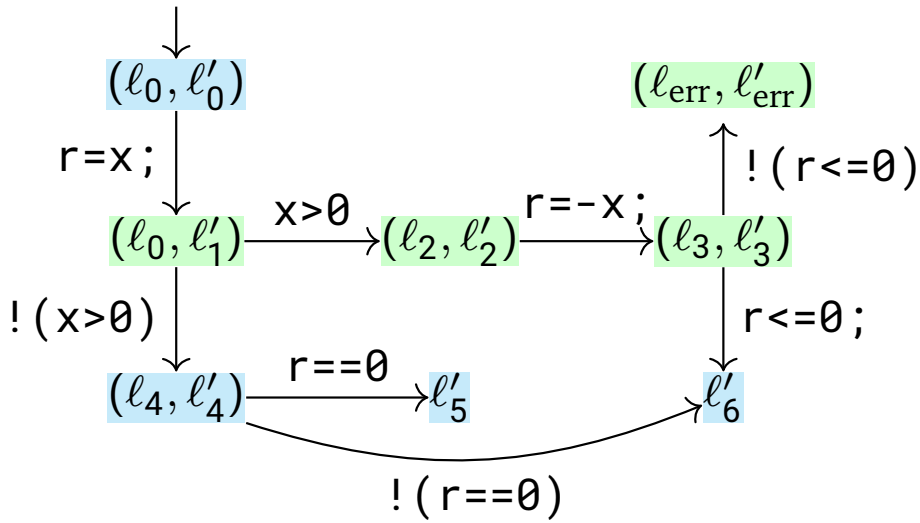
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

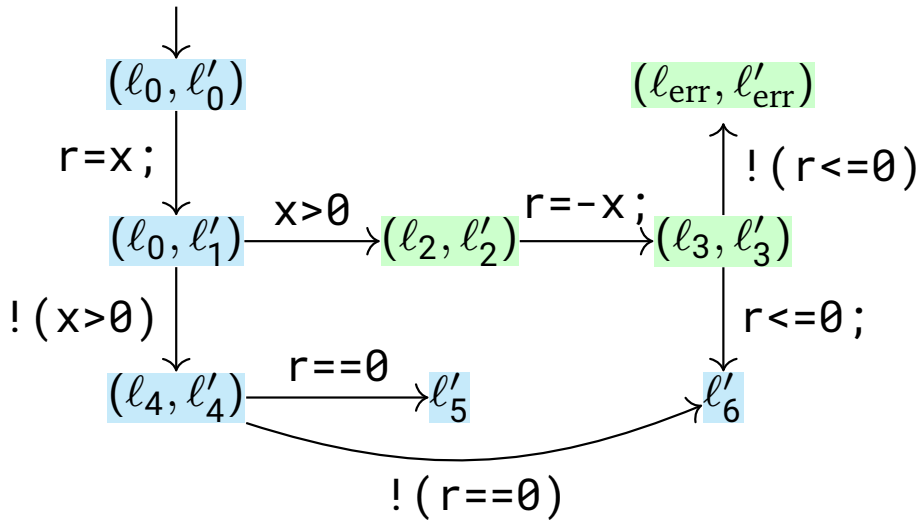
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; waitlist $= \{n_0\} \cup \Delta$;
2: **while** (waitlist $\neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: waitlist $=$ waitlist $\cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

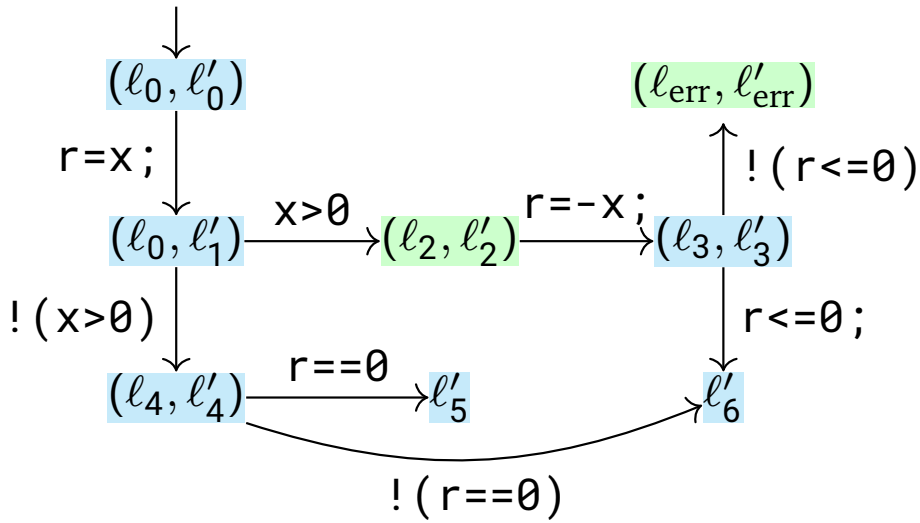
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

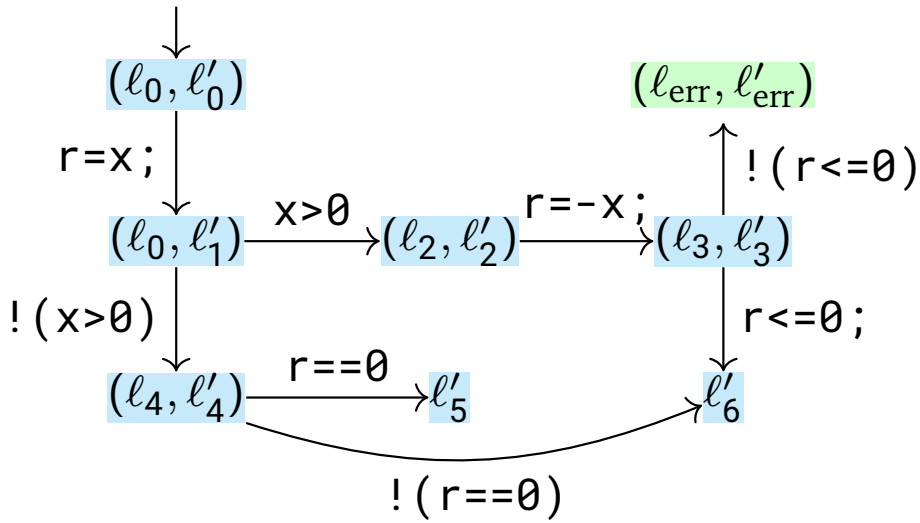
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

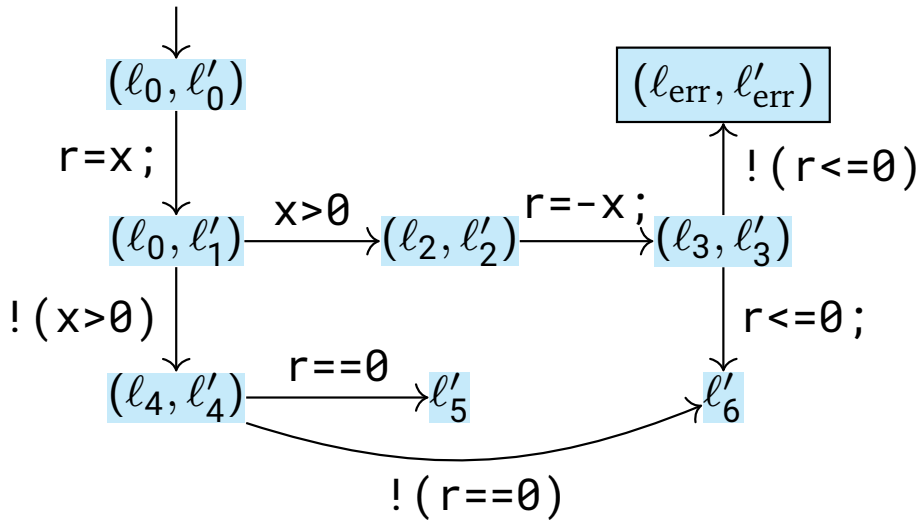
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

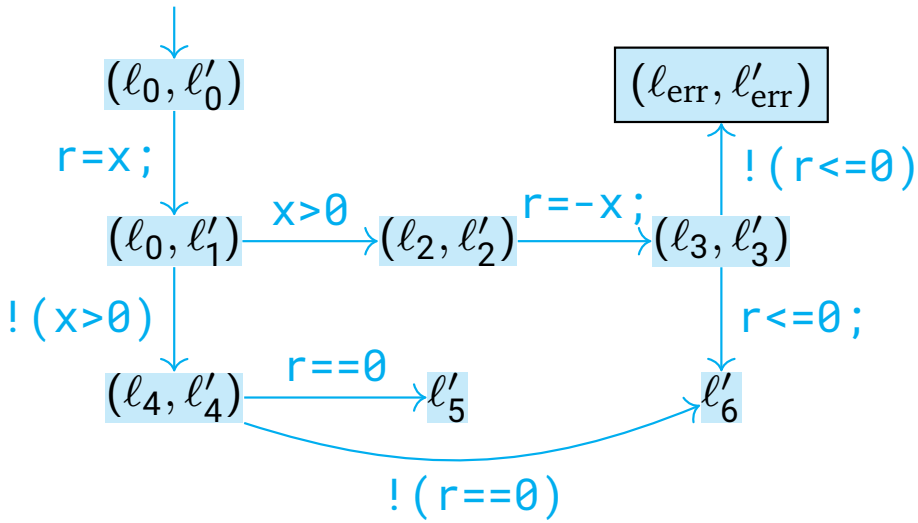
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

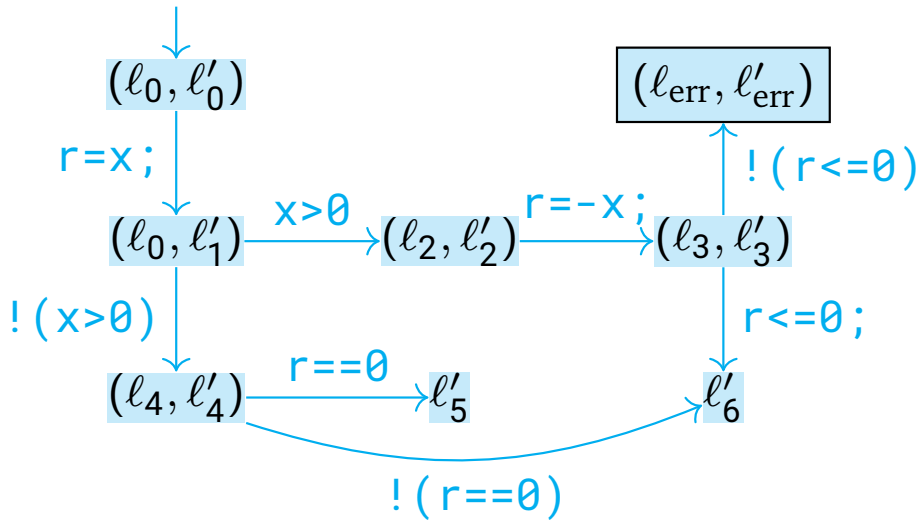
Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

1: $Q = \{n_0\} \cup \Delta$; $\texttt{waitlist} = \{n_0\} \cup \Delta$;
2: **while** ($\texttt{waitlist} \neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: $\texttt{waitlist} = \texttt{waitlist} \cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

Condition Generation from Difference Graph



Input: difference graph (N, E, n_0, Δ)
Output: extracted condition

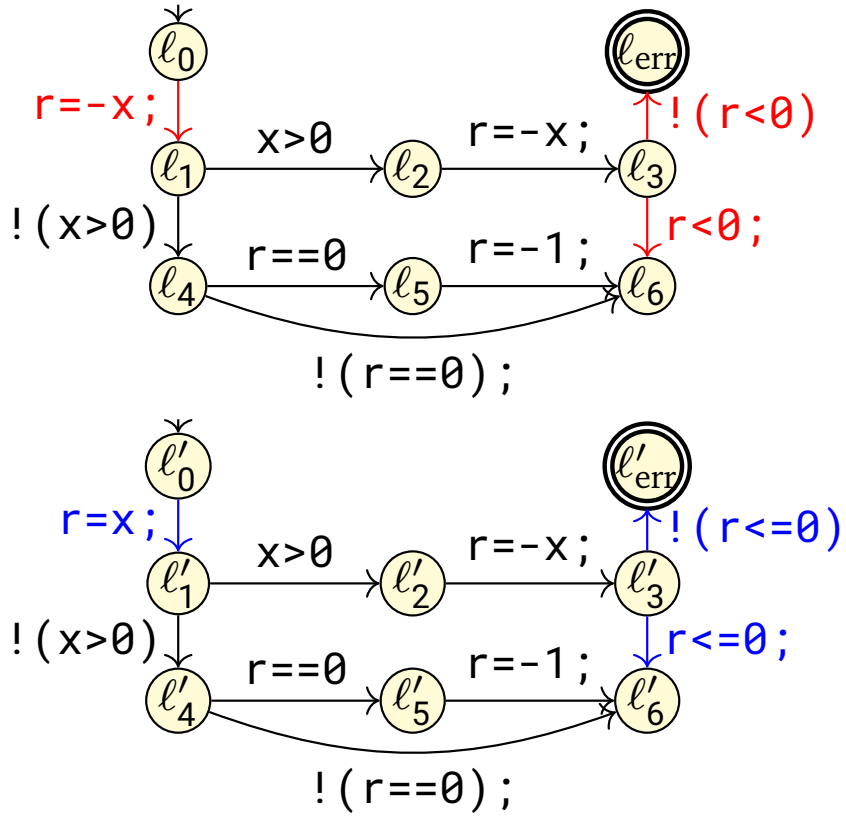
1: $Q = \{n_0\} \cup \Delta$; waitlist $= \{n_0\} \cup \Delta$;
2: **while** (waitlist $\neq \emptyset$) **do**
3: pop n_s from waitlist
4: **for** each $(n_p, (\ell, op, \ell'), n_s) \in E$ with $n_p \notin Q$ **do**
5: $Q = Q \cup \{n_p\}$;
6: waitlist $=$ waitlist $\cup \{n_p\}$;
7: F = $\{n_s \mid \exists (n_p, g, n_s) \in E \wedge n_p \in Q \wedge n_s \notin Q\}$;
8: **return** $(Q \cup F, E \cap (Q \times G' \times (Q \cup F)), n_0, F)$;

Theorem

Let $DG(P, P')$ be a difference graph for programs P and P'. The condition generator outputs a condition that does not cover paths that cause new bugs.

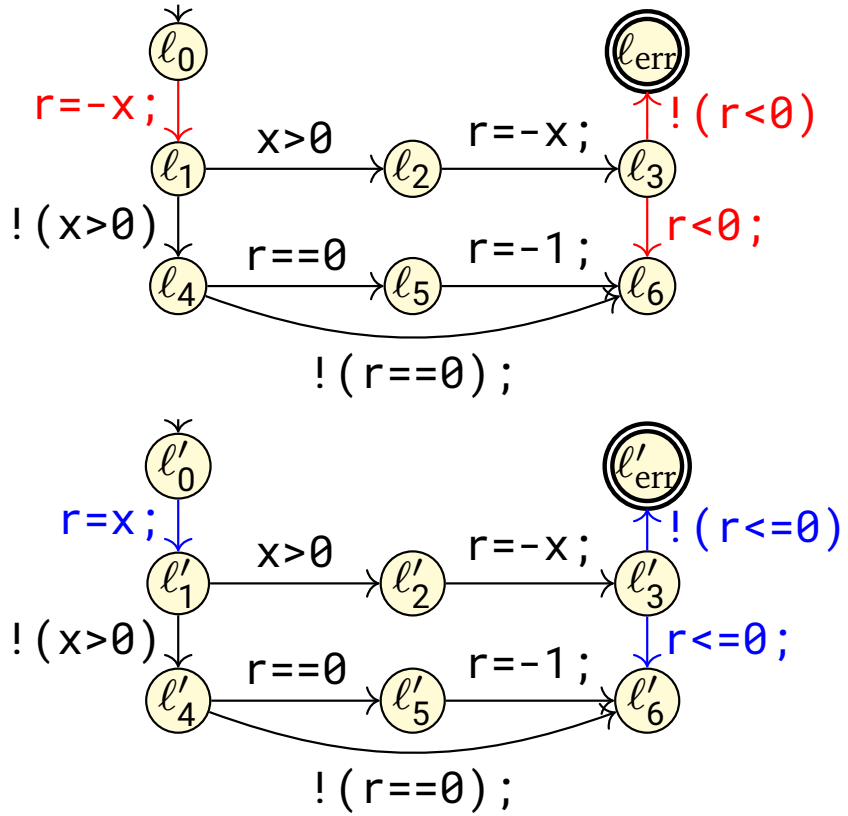
DIFFCOND: Computing Difference Graphs Based on Syntactic Changes

TECHNISCHE UNIVERSITÄT DARMSTADT



DIFFCOND: Computing Difference Graphs Based on Syntactic Changes

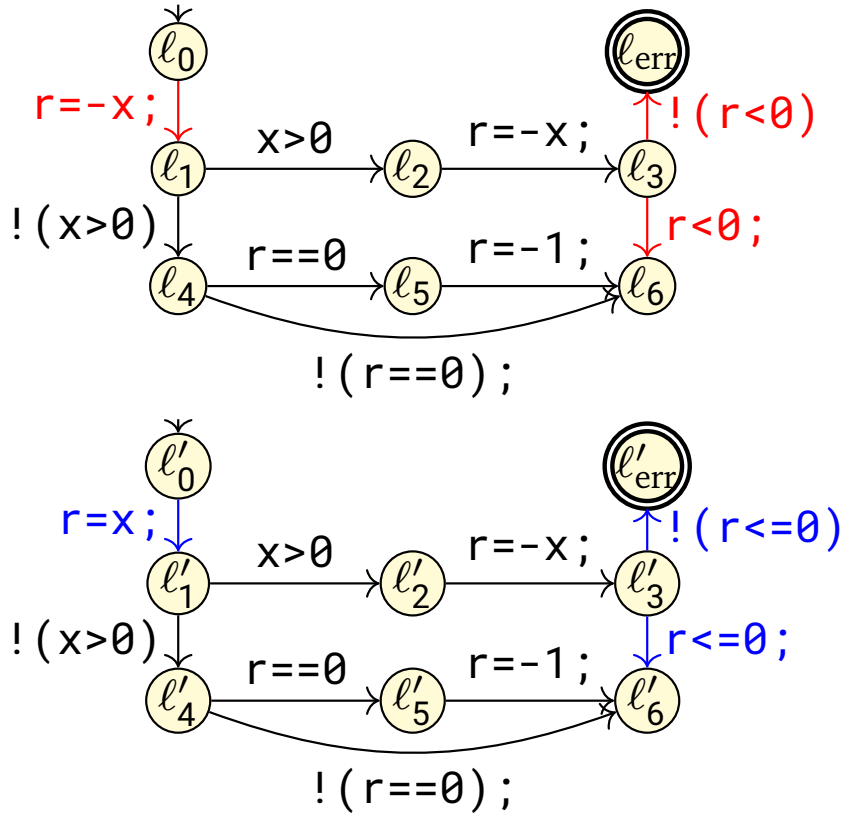
TECHNISCHE UNIVERSITÄT DARMSTADT

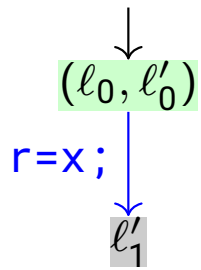


(ℓ_0, ℓ'_0)

DIFFCOND: Computing Difference Graphs Based on Syntactic Changes

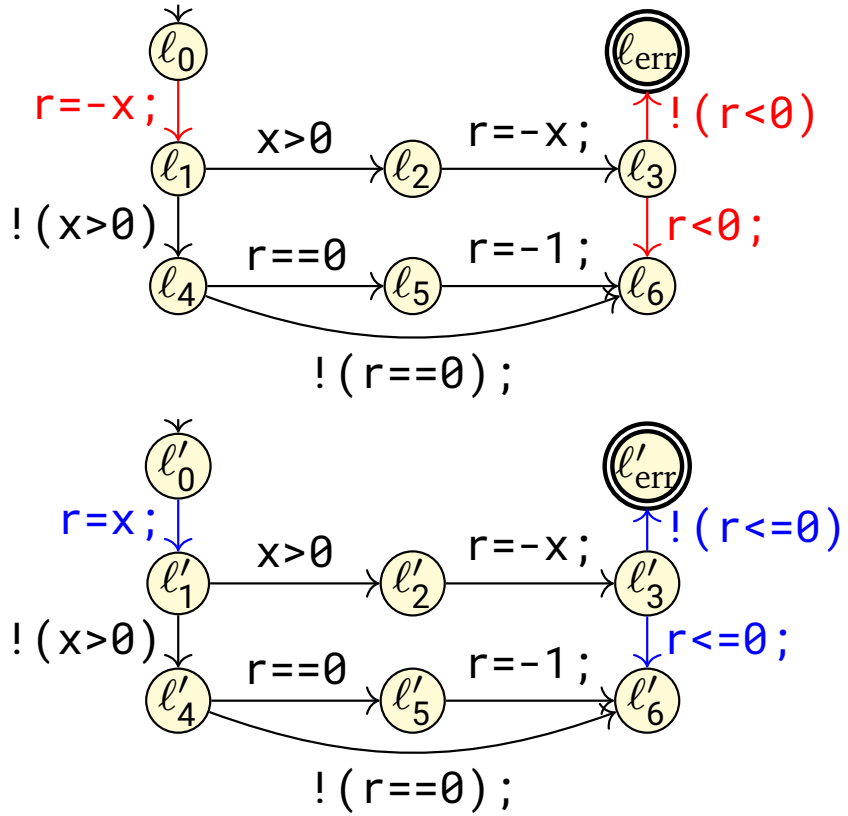
TECHNISCHE UNIVERSITÄT DARMSTADT





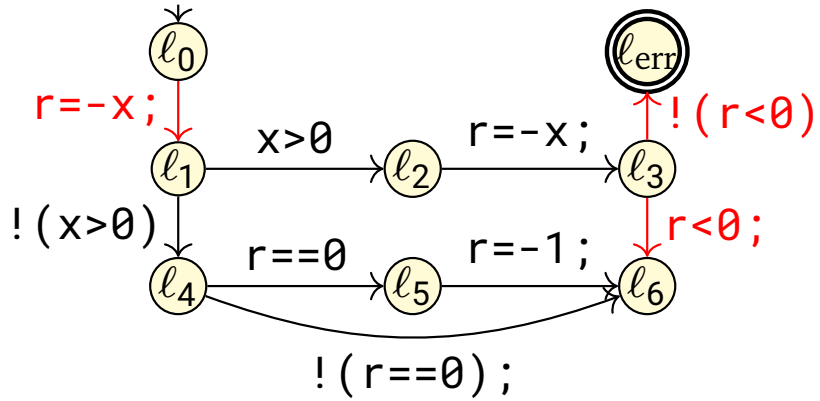
DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

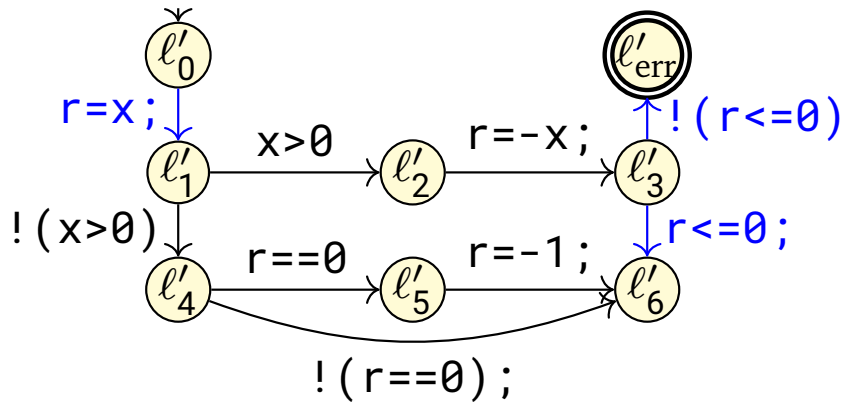
TECHNISCHE UNIVERSITÄT DARMSTADT

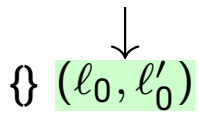


DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

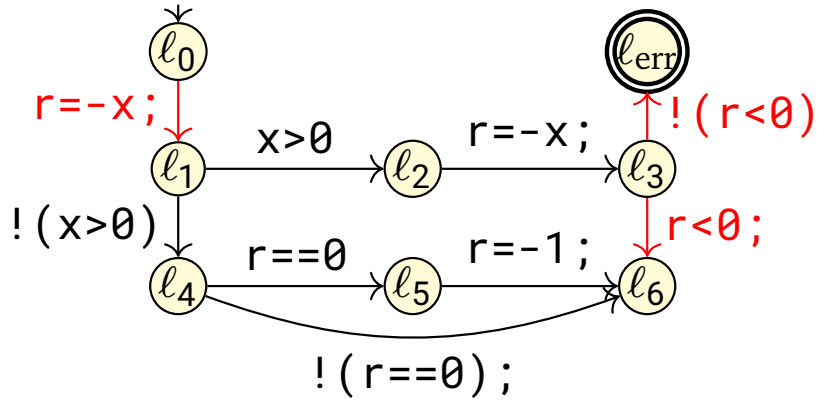


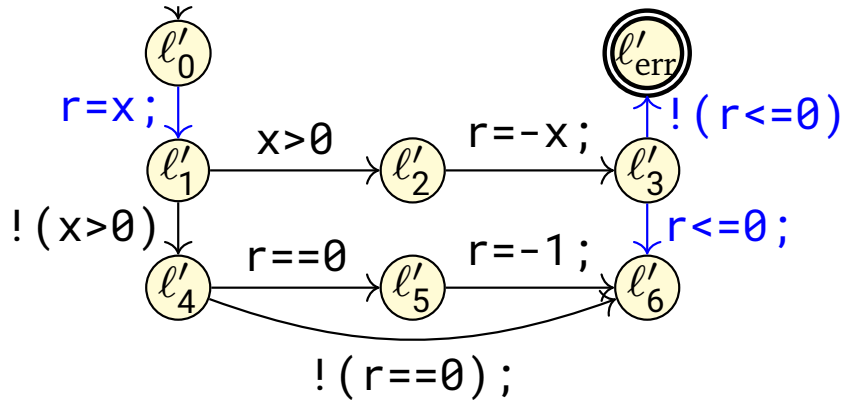


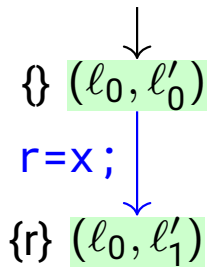


DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

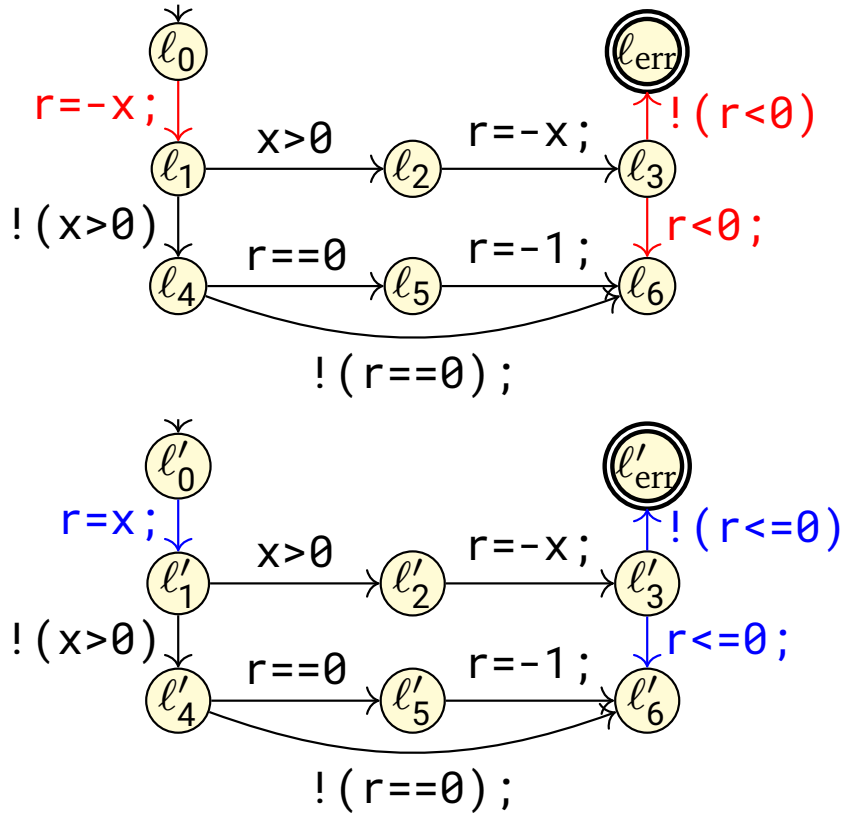


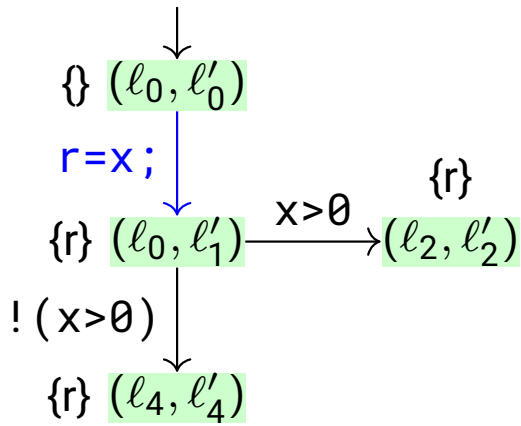




DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

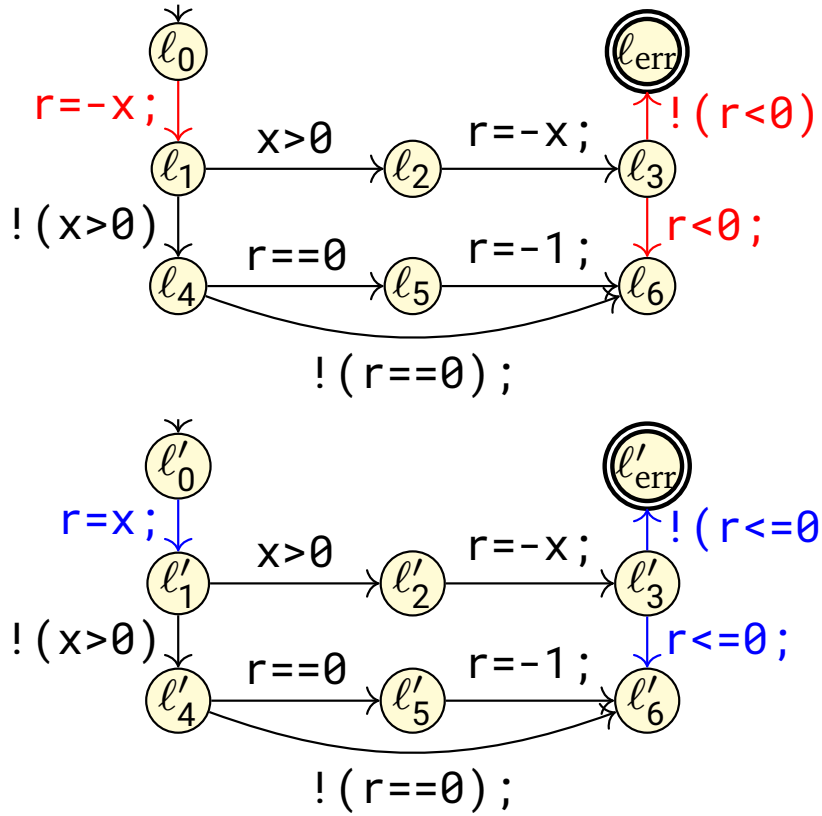
TECHNISCHE UNIVERSITÄT DARMSTADT

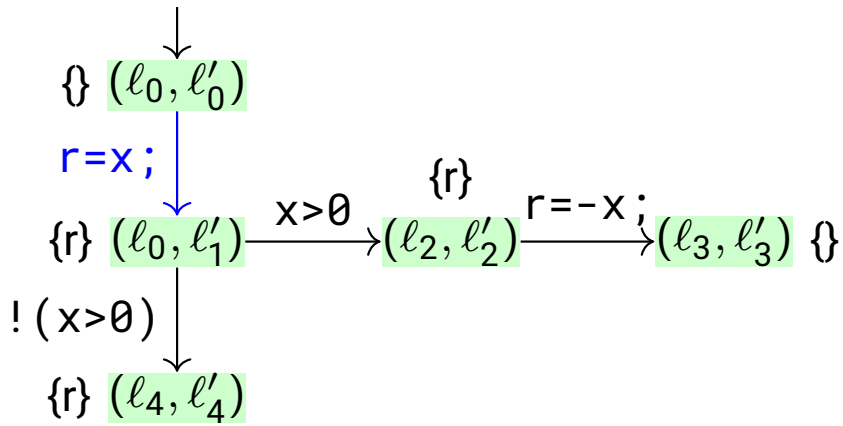




DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

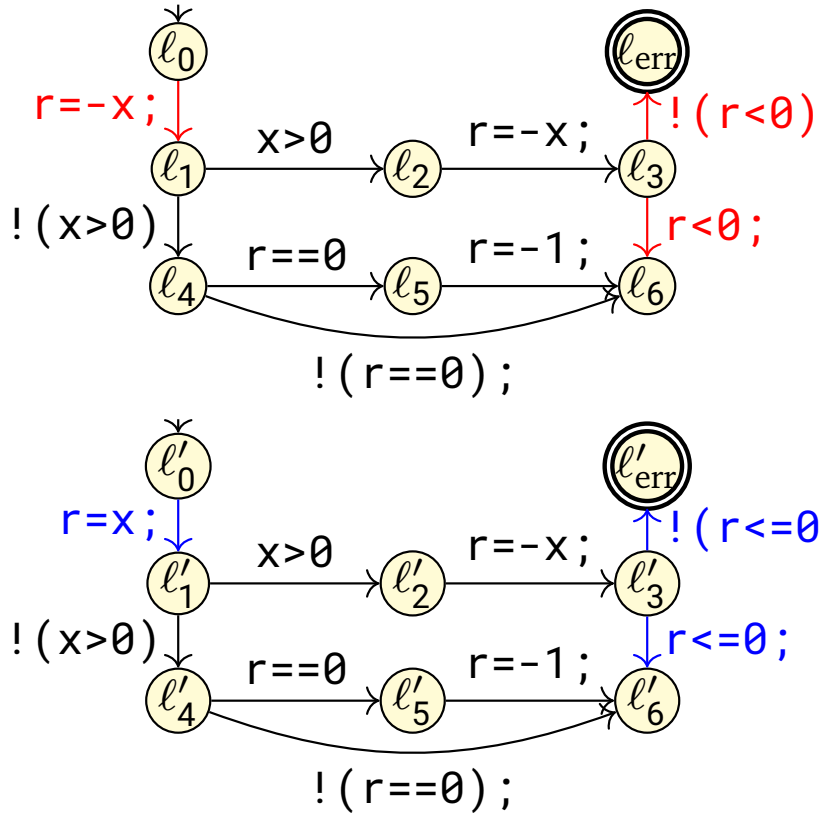
TECHNISCHE UNIVERSITÄT DARMSTADT

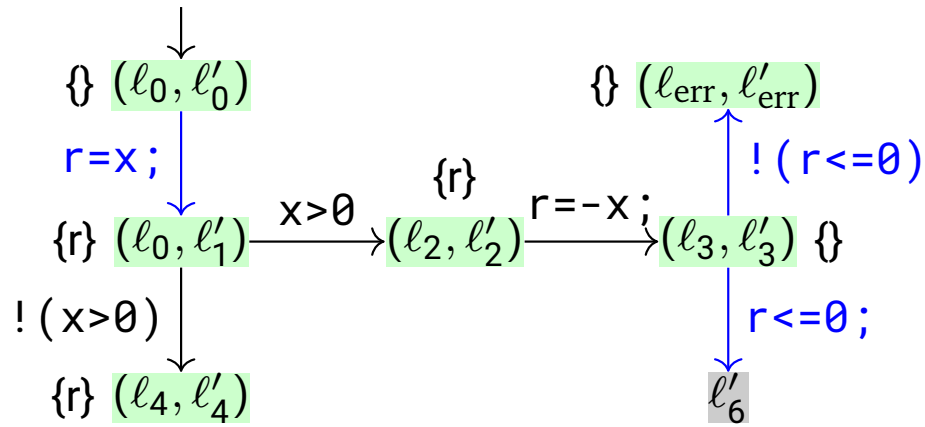




DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

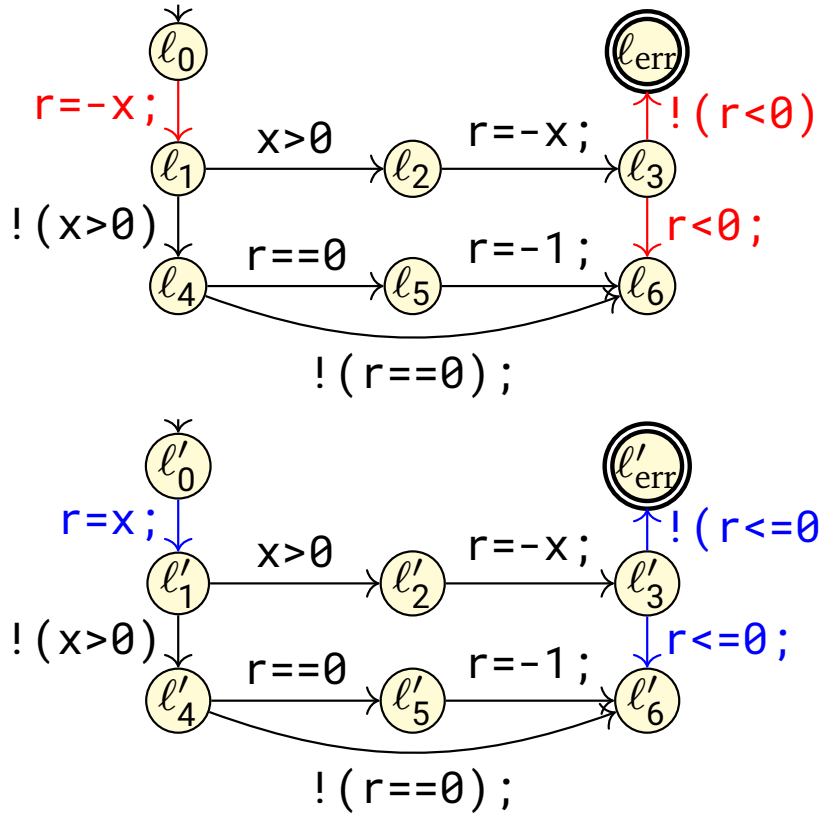
TECHNISCHE UNIVERSITÄT DARMSTADT

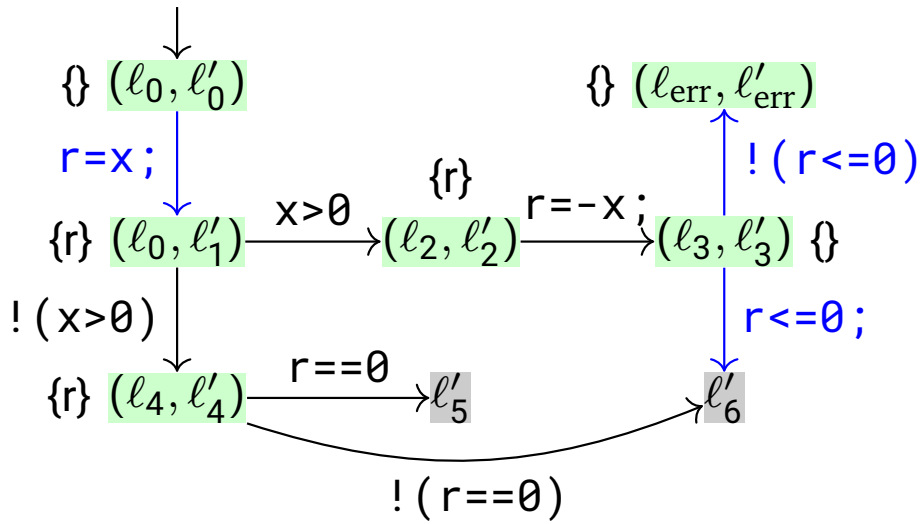




DIFFDP: Utilizing Property and Dataflow to Compute a More Precise Difference Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT





Soundness Result for DIFFDP

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem

DIFFDP *returns a difference graph.*

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Verifiers: ESBMC, CPAchecker, Predicate

Difference Detectors: All, DIFFCOND, DIFFDP

Tasks:
- combination tasks eca05+token, gcd+newton, pals+eca12, sfifo+token, square+softflt
- regression verification tasks

Environment: Intel Xeon E3-1230 v5 CPU, 33 GB, Ubuntu 20.04
- 4 processing units, 15 GB of memory
- 15 min of CPU time

More Effective Than Full Verification? (All vs. DIFFDP)

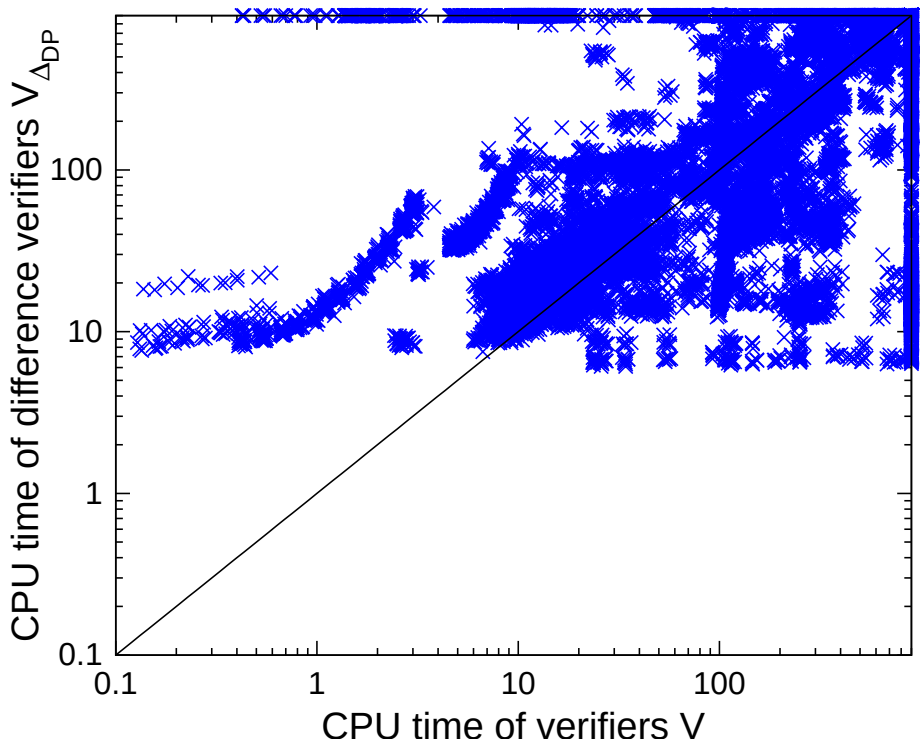
TECHNISCHE UNIVERSITÄT DARMSTADT

	eca05+token (2 340+1 300)			gcd+newton (1 352+572)			pals+eca12 (1 700+1 050)			sfifo+token (1 206+663)			square+softflt (165+75)			regression (3 936+0)		
	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$
PREDICATE	912	1040	0	0	520	0	0	50	0	558	**507**	0	22	51	0	2595	0	0
PREDICATE$^{\Delta^{\text{nat}}_{\text{DP}}}$	**1400**	**985**	0	**156**	520	0	**125**	**75**	0	**594**	488	0	**115**	**75**	0	**2663**	0	0
PREDICATE	912	**1040**	0	0	520	0	0	50	0	558	**507**	0	22	51	0	2595	0	0
PREDICATE$^{\Delta^{\text{red}}_{\text{DP}}}$	**1390**	955	0	**156**	**572**	0	**125**	50	0	**639**	456	0	**135**	**60**	0	**2685**	0	0
CPACHECKER	633	**1296**	0	0	494	0	0	100	0	434	510	0	0	**75**	0	**3931**	0	0
CPACHECKER$^{\Delta^{\text{red}}_{\text{DP}}}$	**1119**	1252	0	**156**	**520**	0	**671**	**500**	0	**480**	**638**	0	**117**	60	0	3618	0	0
ESBMC	0	**1125**	0	570	572	0	**198**	**530**	**0**	0	**663**	0	165	75	0	0	0	0
ESBMC$^{\Delta^{\text{red}}_{\text{DP}}}$	0	900	0	**880**	572	0	0	70	10	**101**	618	0	156	75	0	0	0	0

Difference verification with DIFFDP condition extractor often more effective

More Efficient Than Full Verification? (All vs. DIFFDP)

TECHNISCHE UNIVERSITÄT DARMSTADT



- Conditional verifier faster
- Detector time sometimes too costly to be beneficial

Difference Verification With DIFFDP More Effective Then With DIFFCOND?

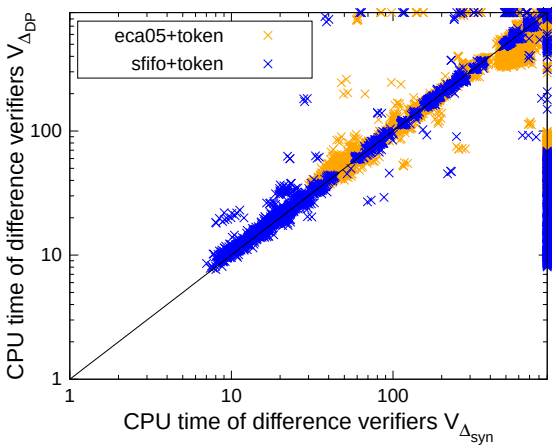
TECHNISCHE UNIVERSITÄT DARMSTADT

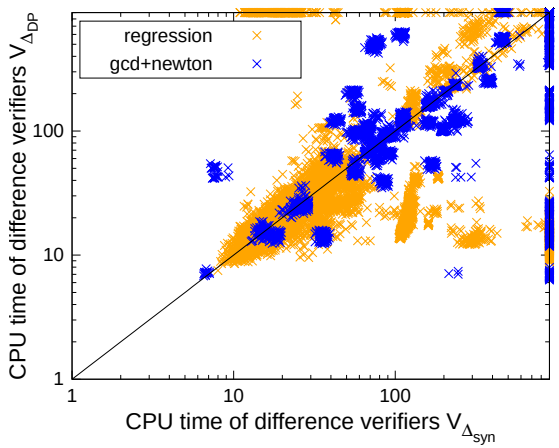
	eca05+token (2 340+1 300)			gcd+newton (1 352+572)			pals+eca12 (1 700+1 050)			sfifo+token (1 206+663)			square+softflt (165+75)			regression (3 936+0)		
	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$	✓	✗	ϟ$_3$
PREDICATE$^{\Delta^{\text{nat}}_{\text{syn}}}$	1395	**987**	0	48	520	0	10	52	0	589	481	0	61	75	0	2599	0	0
PREDICATE$^{\Delta^{\text{nat}}_{\text{DP}}}$	**1400**	985	0	**156**	520	0	**125**	**75**	0	**594**	**488**	0	**115**	75	0	**2663**	0	0
PREDICATE$^{\Delta^{\text{red}}_{\text{syn}}}$	**1394**	**975**	0	48	474	0	10	**95**	0	**642**	**468**	0	81	45	0	2639	0	0
PREDICATE$^{\Delta^{\text{red}}_{\text{DP}}}$	1390	955	0	**156**	**572**	0	**125**	50	0	639	456	0	**135**	**60**	0	**2685**	0	0
CPACHECKER$^{\Delta^{\text{red}}_{\text{syn}}}$	1000	**1282**	0	48	469	0	41	140	0	480	**663**	0	61	45	0	**3906**	0	0
CPACHECKER$^{\Delta^{\text{red}}_{\text{DP}}}$	**1119**	1252	0	**156**	**520**	0	**671**	**500**	0	480	638	0	**117**	**60**	0	3618	0	0
ESBMC$^{\Delta^{\text{red}}_{\text{syn}}}$	0	0	0	333	572	0	0	0	**0**	0	78	0	105	75	0	0	0	0
ESBMC$^{\Delta^{\text{red}}_{\text{DP}}}$	0	**900**	0	**880**	572	0	0	**70**	10	101	618	0	**156**	75	0	0	0	0

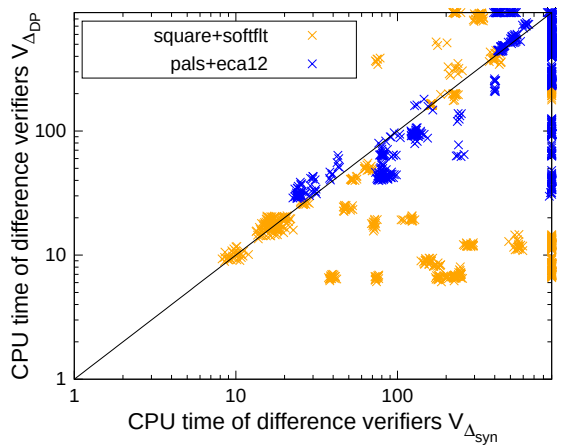
Difference verification with DIFFDP often more effective

Difference Verification With DIFFDP More Efficient Then With DIFFCOND?

TECHNISCHE UNIVERSITÄT DARMSTADT



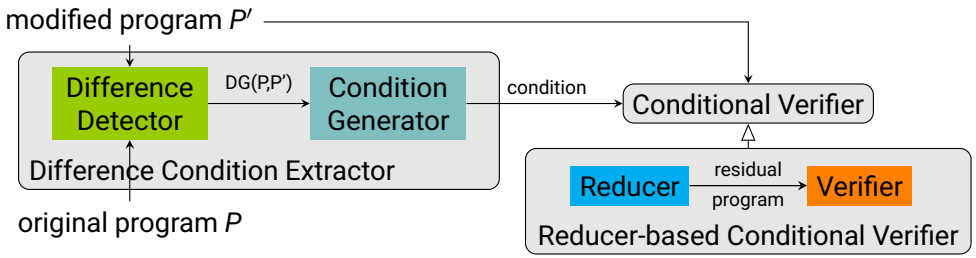




Difference verification with DIFFDP more efficient for several tasks

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT



- More sophisticated, sound difference detector
- More effective and efficient than full verification on several tasks
- Better than existing syntax based difference detector on several tasks