

# CHCVERIF: A PORTFOLIO-BASED SOLVER FOR CONSTRAINED HORN CLAUSES

Mihály Dobos-Kovács, Levente Bajczi, András Vörös

BME-MIT, Budapest



**Critical Systems  
Research Group**

# Agenda



# Agenda



Background & theory

Goals & contributions

Experiment design & results

Significance & summary

# Agenda



# Agenda



Background & theory

Goals & contributions

Experiment design & results

Significance & summary

# Background & Theory

What are **CHCs**?

How to **transform** them **to C**?

# CHC to Control Flow Automata

$$n = 0 \Rightarrow A(n)$$

$$A(n - 2) \Rightarrow A(n)$$

$$A(6) \Rightarrow \text{false}$$

# CHC to Control Flow Automata

L0

$$n = 0 \Rightarrow A(n)$$

$$A(n - 2) \Rightarrow A(n)$$

$$A(6) \Rightarrow \text{false}$$

LE

# CHC to Control Flow Automata

L0

A

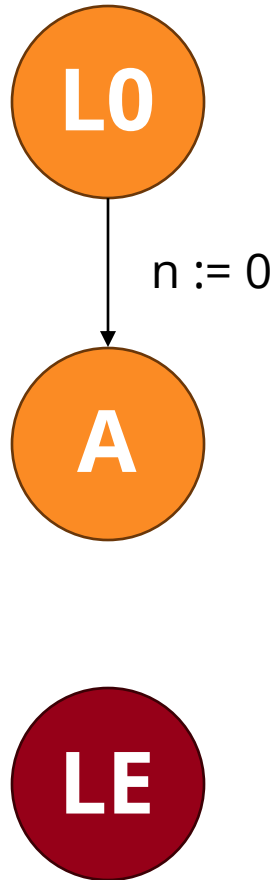
LE

$$n = 0 \Rightarrow A(n)$$

$$A(n - 2) \Rightarrow A(n)$$

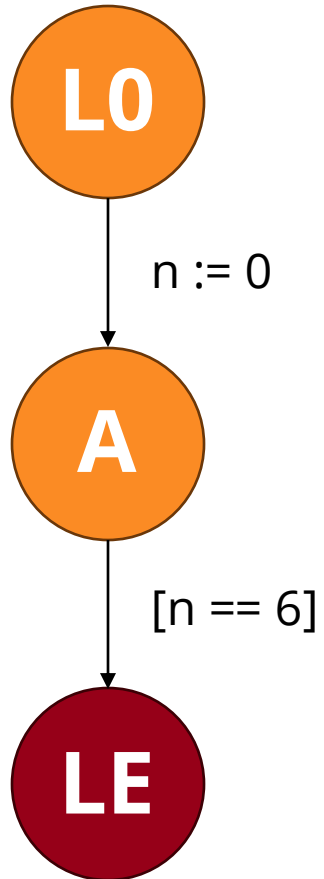
$$A(6) \Rightarrow \text{false}$$

# CHC to Control Flow Automata



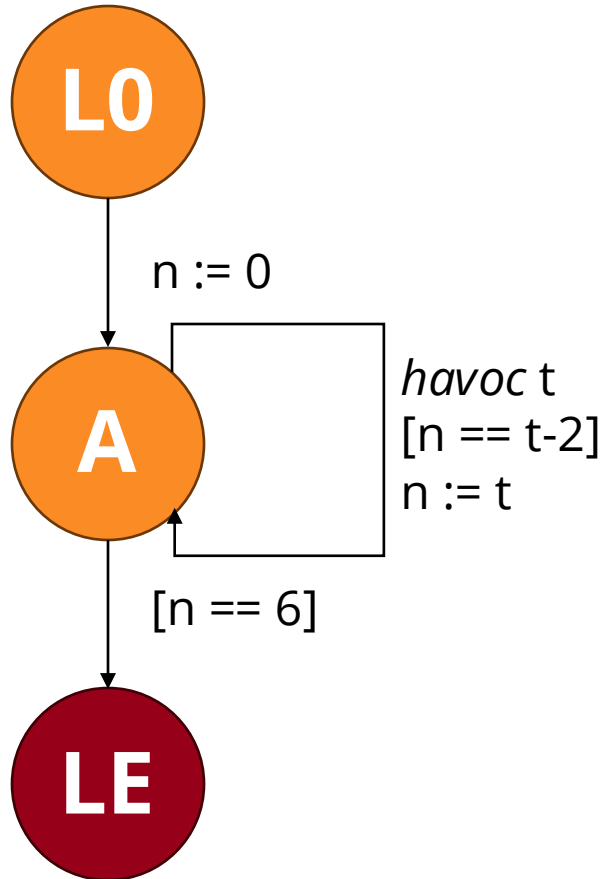
➔  $n = 0 \Rightarrow A(n)$   
 $A(n - 2) \Rightarrow A(n)$   
 $A(6) \Rightarrow \text{false}$

# CHC to Control Flow Automata



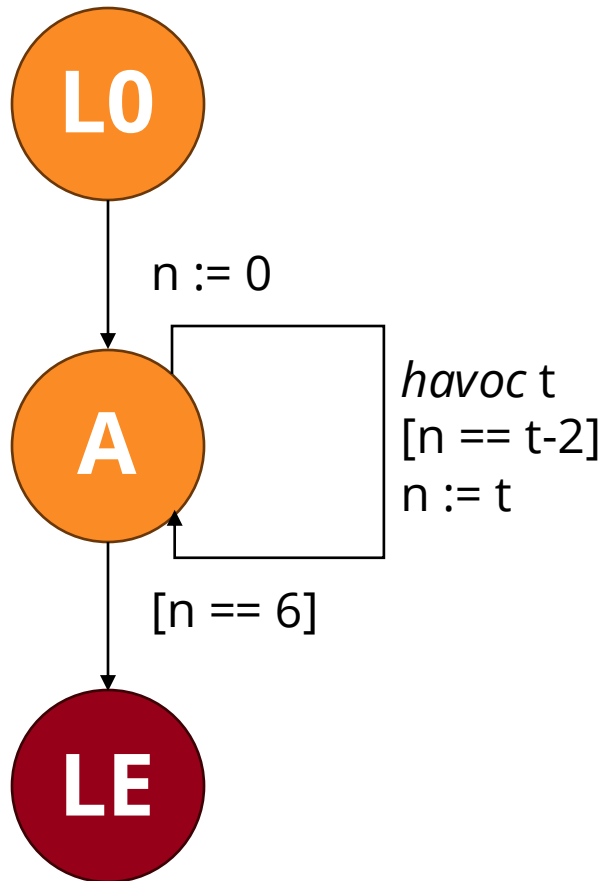
$n = 0 \Rightarrow A(n)$   
 $A(n - 2) \Rightarrow A(n)$   
**→**  $A(6) \Rightarrow \text{false}$

# CHC to Control Flow Automata



$n = 0 \Rightarrow A(n)$   
 $\Rightarrow A(n - 2) \Rightarrow A(n)$   
 $A(6) \Rightarrow \text{false}$

# CHC to Control Flow Automata



$n = 0 \Rightarrow A(n)$   
 $\rightarrow A(n - 2) \Rightarrow A(n)$   
 $A(6) \Rightarrow \text{false}$






```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

Forward

Bottom-up

# CHC to Control Flow Automata

## Bottoms Up for CHCs: Novel Transformation of Linear Constrained Horn Clauses to Software Verification

Márk Somorjai  Mihály Dobos-Kovács  Zsófia Ádám   
Levente Bajczi  András Vörös   
vori@mit.bme.hu

Department of Measurement and Information Systems  
Budapest University of Technology and Economics

Constrained Horn Clauses (CHCs) have conventionally been used as a low-level representation in formal verification. Most existing solvers use a diverse set of specialized techniques, including direct state space traversal or under-approximating abstraction, necessitating purpose-built complex algorithms. Other solvers successfully simplified the verification workflow by translating the problem to inputs for other verification tasks, leveraging the strengths of existing algorithms. One such approach transforms the CHC problem into a recursive program roughly emulating a *top-down* solver for the deduction task; and verifying the reachability of a safety violation specified as a control location. We propose an alternative *bottom-up* approach for linear CHCs, and evaluate the two options in the open-source model checking framework THETA on both synthetic and industrial examples. We find that there is a more than twofold increase in the number of solved tasks when the novel *bottom-up* approach is used in the verification workflow, in contrast with the *top-down* technique.

```
int main() {  
  int n, t = 0;  
  n = t;  
  while(true) {  
    t = nondet();  
    if(n == 6) reach_error();  
    else if(n == t-2) n = t;  
  }  
}
```

Forward

Bottom-up

HCVS'23: 10.4204/EPTCS.402.11

# CHC to Control Flow Automata #2

$n = 0 \Rightarrow A(n)$   
 $A(n - 2) \Rightarrow A(n)$   
 $A(6) \Rightarrow \text{false}$

```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

Forward

Bottom-up

# CHC to Control Flow Automata #2

```
_Bool A(int n) {  
    if(A(n-2)) return 1;  
    if(n == 0) return 1;  
    return 0;  
}  
...  
if(A(6)) reach_error();
```

Backward

Top-down

$n = 0 \Rightarrow A(n)$   
 $A(n - 2) \Rightarrow A(n)$   
 $A(6) \Rightarrow \text{false}$

```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

Forward


Bottom-up

# CHC to Programs

```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

# CHC to Programs

- What if  $t > \text{MAX\_INT}$ ?
  - Or array out of bounds, ...




```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

# CHC to Programs

- What if  $t > \text{MAX\_INT}$ ?
  - Or array out of bounds, ...

Tell the verification tool to use SMT semantics

- Not available with every tool
- Not *really* a C program any more



```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

# CHC to Programs


- What if  $t > \text{MAX\_INT}$ ?
  - Or array out of bounds, ...

Tell the verification tool to use SMT semantics

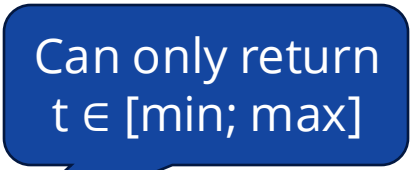
- Not available with every tool
- Not *really* a C program any more

Use *safeguarding* to prevent erroneous verdicts

- Limits verification power
  - Safe verdicts are dependent on all variables being bounded
  - Unsafe verdicts are still valid



```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```



# CHC to Programs

## Solving Constrained Horn Clauses as C Programs with CHC2C

Levente Bajczi<sup>ID</sup> and Vince Molnár<sup>ID</sup>

Department of Measurement and Information Systems,  
Budapest University of Technology and Economics, Hungary  
{bajczi,molnarv}@mit.bme.hu

**Abstract.** Solving Constrained Horn Clauses (CHC) is necessitated by numerous fields in formal methods, from verifying software and smart contracts to modeling systems, yet the competitive scene for academic tools remains fairly sparse, especially compared to more popular fields such as software verification. Comparative evaluation as a competition,

```
int main() {  
    int n, t = 0;  
    n = t;  
    while(true) {  
        t = nondet();  
        if(n == 6) reach_error();  
        else if(n == t-2) n = t;  
    }  
}
```

Can only return  $t \in [\min; \max]$

SPIN'24: 10.1007/978-3-031-66149-5\_8

# Goals & Contributions

What did we want to achieve?

# Goals of this Work

Broaden the field of **CHC solvers**  
with **SW verification tools**

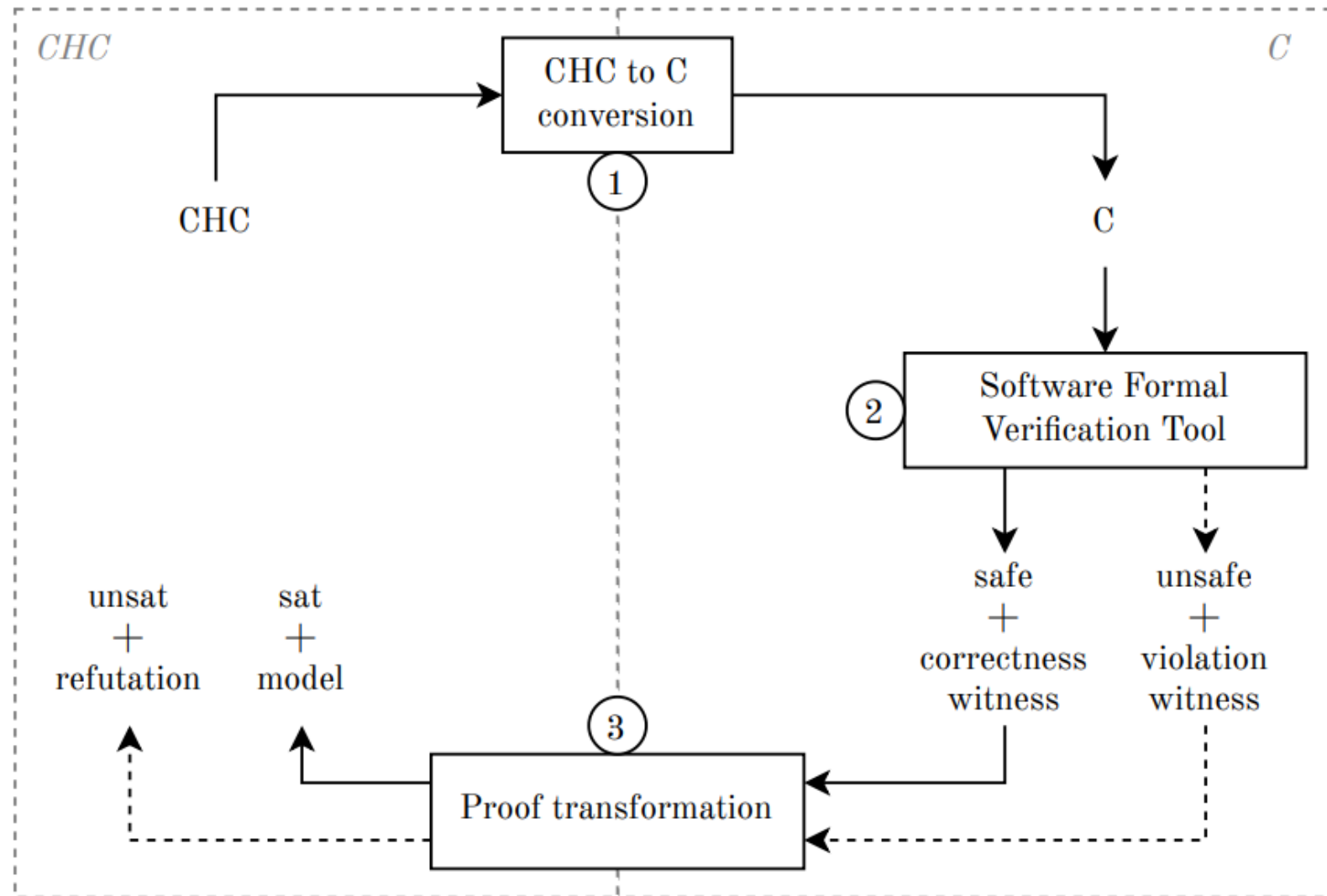
Provide **SW verification** tools with  
valuable benchmarks to **test** and **debug**

# Experiment design & results

**What** did **we** do?

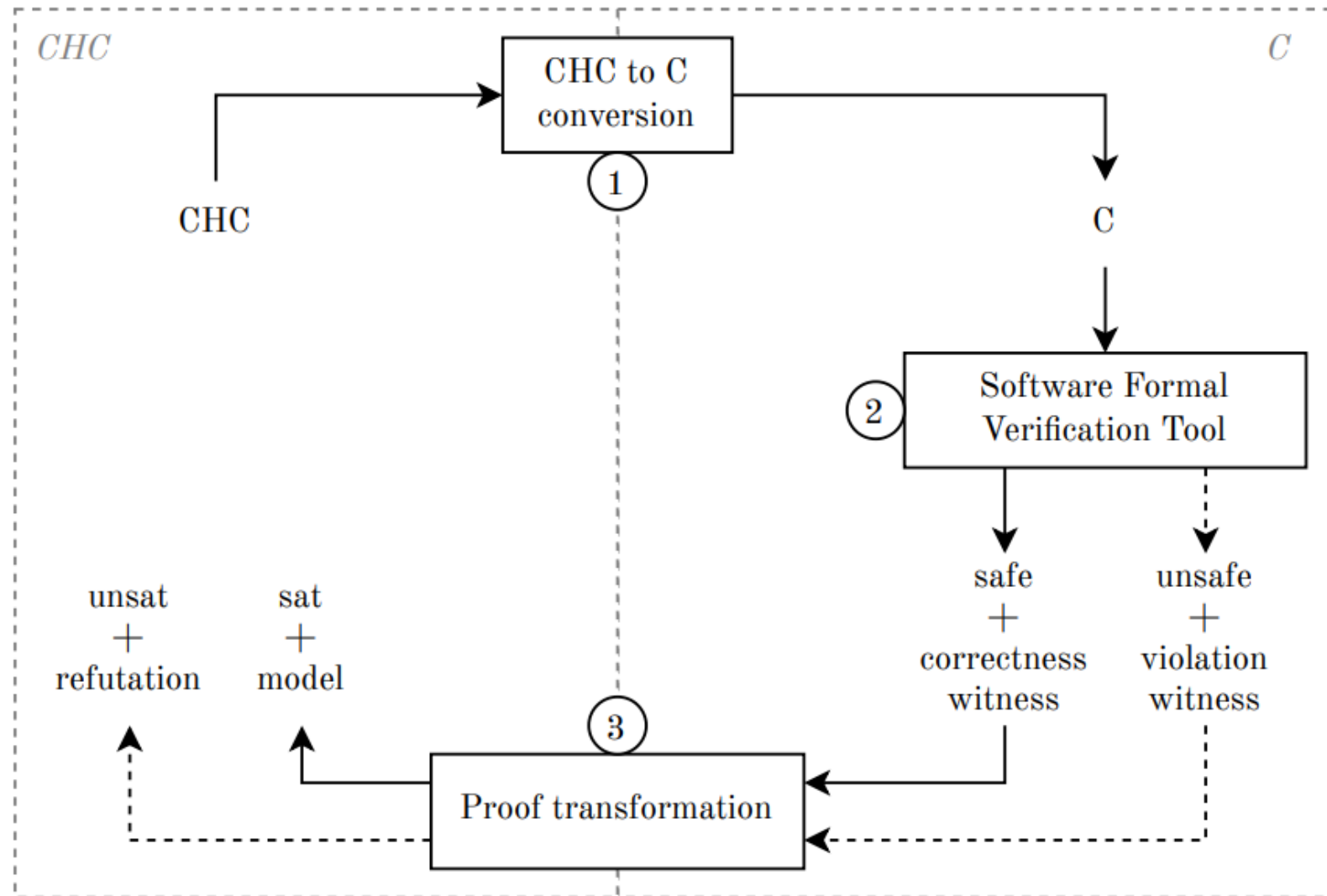
**How** did the **tools** do?

# CHCVERIF overview



# CHCVERIF overview

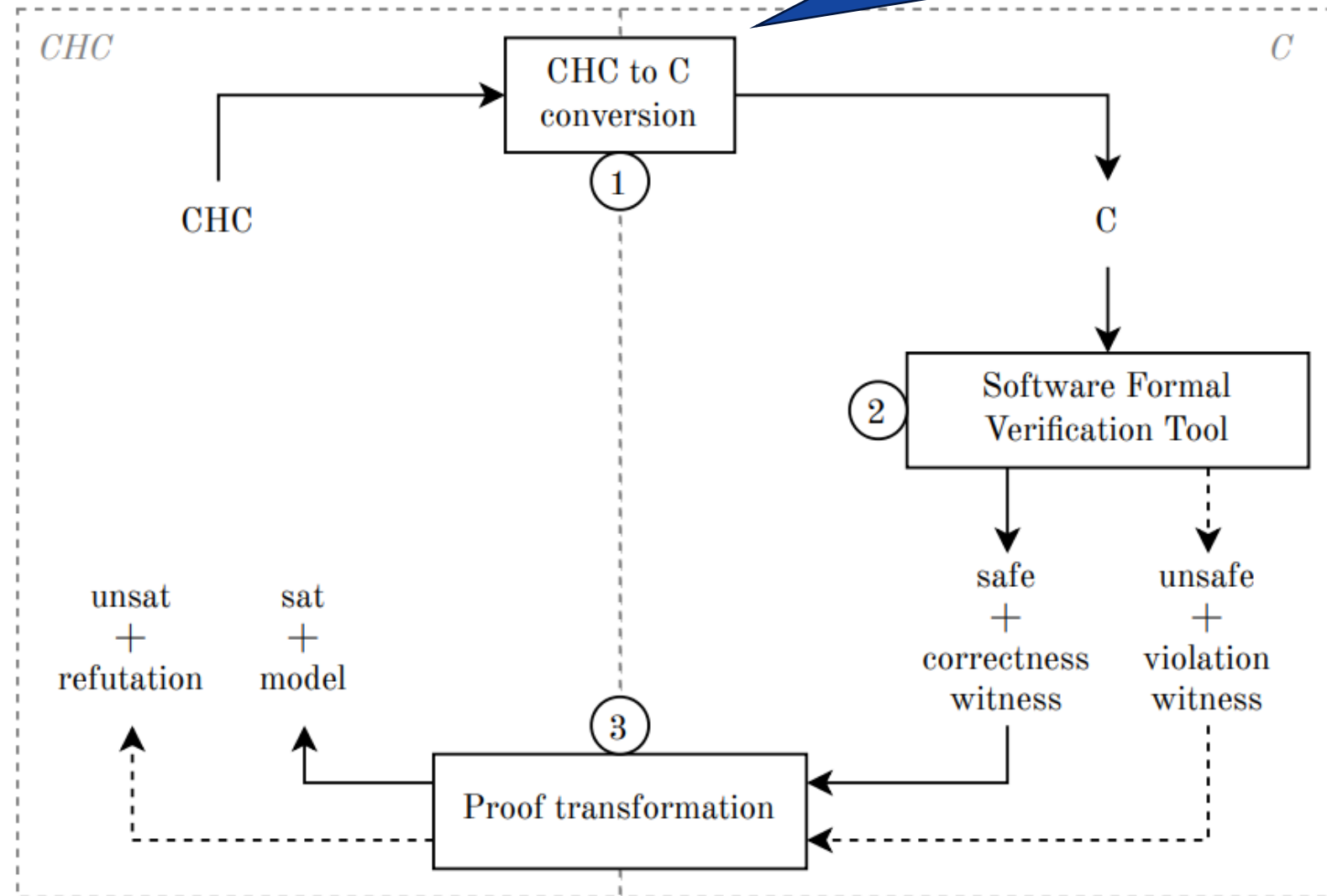
**No arrays, no ADTs**  
**LIA, BV(, Fp)**



# CHCVERIF overview

<http://github.com/ftsrg/chc2c>

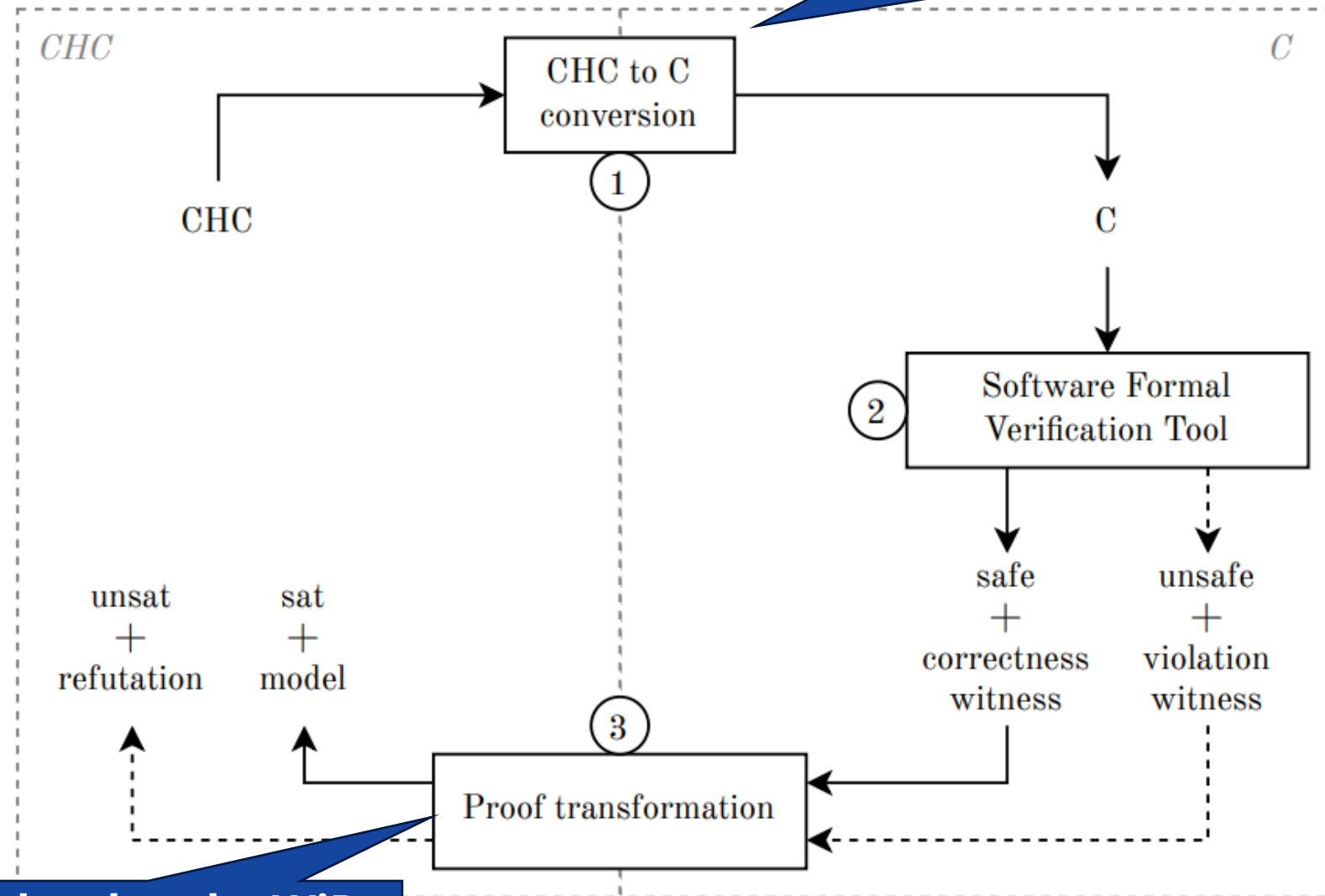
**No arrays, no ADTs**  
**LIA, BV(, Fp)**



# CHCVERIF overview

<http://github.com/ftsrg/chc2c>

**No arrays, no ADTs**  
**LIA, BV(, Fp)**



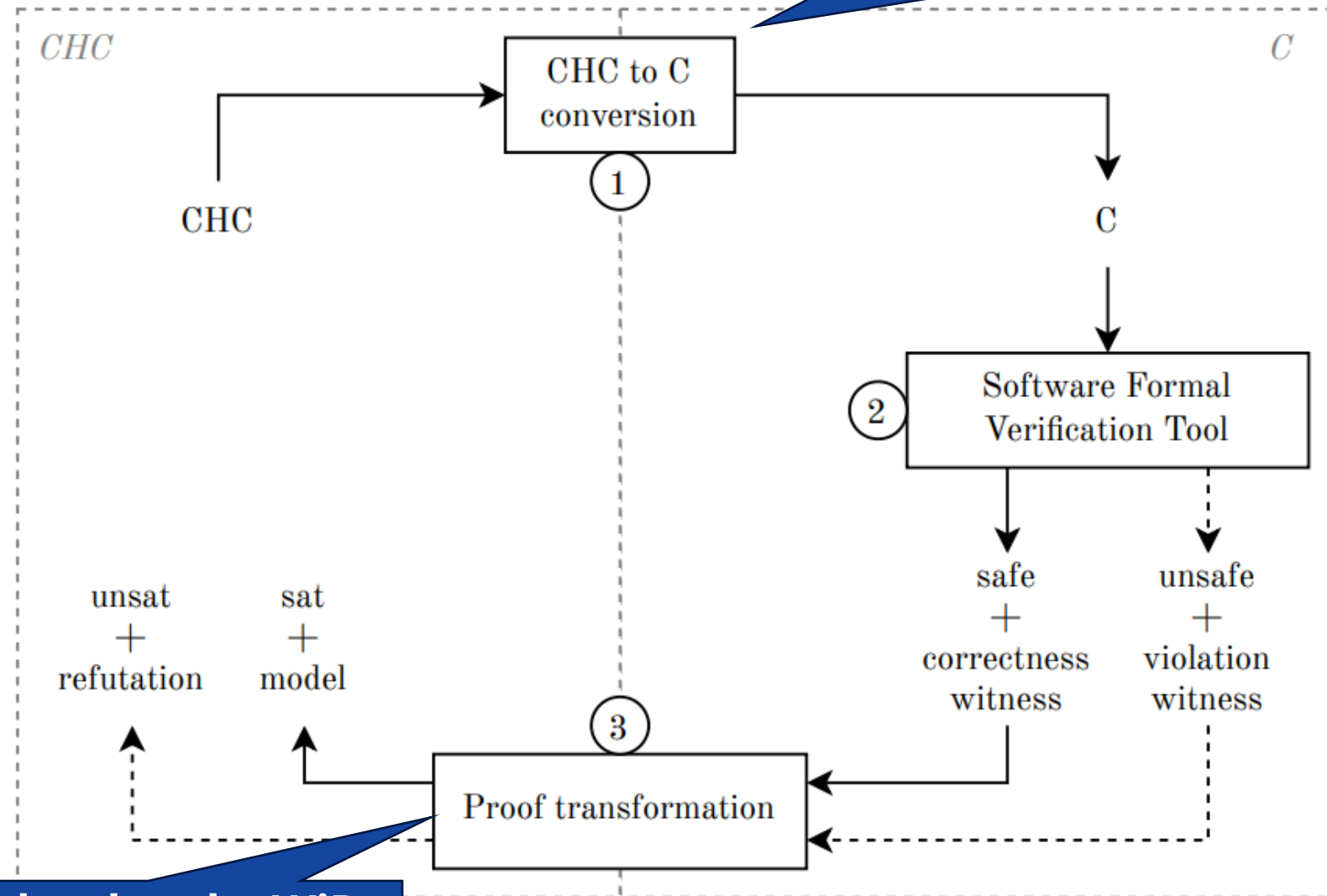
**Spoiler: hard to do, WiP**

# CHCVERIF overview

<http://github.com/ftsrg/chc2c>

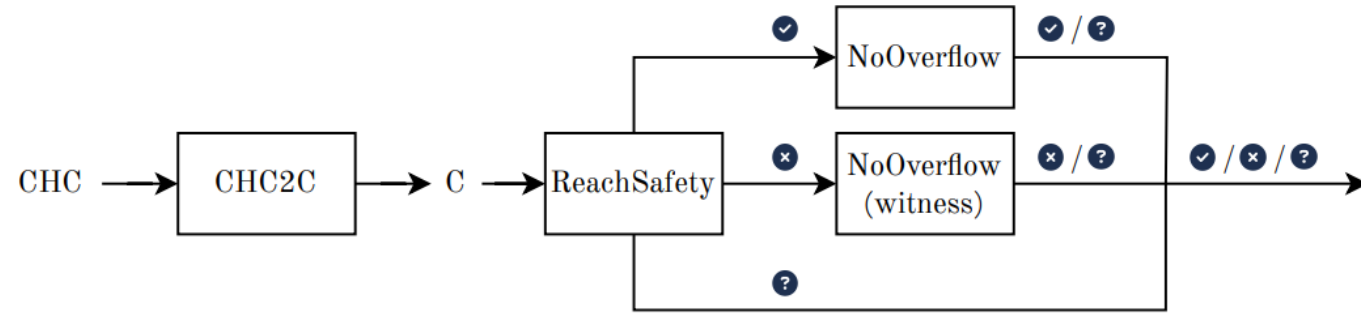
No arrays, no ADTs  
LIA, BV(, Fp)

Strength: **portfolio**  
*CoVeriTeam*

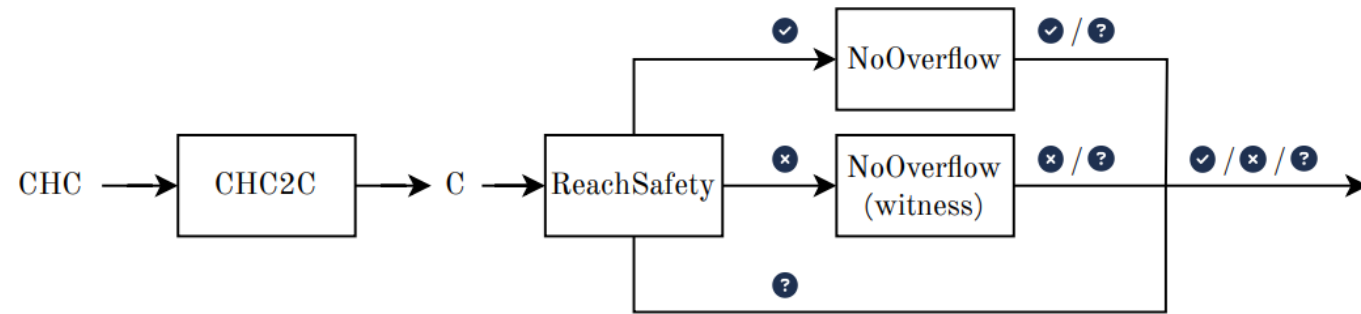


Spoiler: hard to do, WiP

# CHCVERIF: LIA

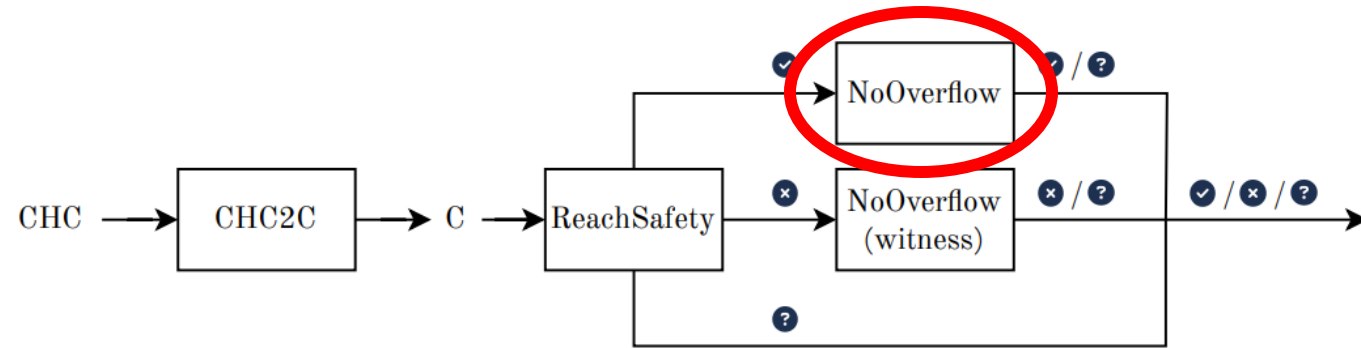


# CHCVERIF: LIA



Recursive	Verdict	Verdict type	goblint	mopsa	emergenttheta	sv-sanitizers	thorn	theta	svf-svc	bubaak-split	cpachecker	2ls	ukojak	bubaak	utaipan	symbiotic	uautomizer	esbmc-kind	Out of
False	No overflow	confirmed	26	24	22	9	22	22	81	21	27	28	32	21	33	21	36	0	195
	Overflow	confirmed	0	0	0	0	0	0	0	61	65	75	86	92	93	93	96	105	
		wrong	0	0	0	0	0	0	105	0	0	0	0	0	0	0	0	0	
True	No overflow	confirmed	45	28	43	43	43	43	224	28	65	72	79	29	76	66	93	88	557
	Overflow	confirmed	0	0	0	0	0	0	0	9	68	177	217	263	276	280	288	289	
		wrong	0	0	0	0	0	0	304	0	0	0	0	0	0	0	0	0	

# CHCVERIF: LIA



Recursive	Verdict	Verdict type	goblint	mopsa	emergenttheta	sv-sanitizers	thorn	theta	svf-svc	bubaak-split	cpachecker	2ls	ukojak	bubaak	utaipan	symbiotic	uautomizer	esbmc-kind	Out of
False	No overflow	confirmed	26	24	22	9	22	22	81	21	27	28	32	21	33	21	36	0	195
	Overflow	confirmed	0	0	0	0	0	0	0	61	65	75	86	92	93	93	96	105	
		wrong	0	0	0	0	0	0	105	0	0	0	0	0	0	0	0	0	
True	No overflow	confirmed	45	28	43	43	43	43	224	28	65	72	79	29	76	66	93	88	557
	Overflow	confirmed	0	0	0	0	0	0	0	9	68	177	217	263	276	280	288	289	
		wrong	0	0	0	0	0	0	304	0	0	0	0	0	0	0	0	0	

# CHCVERIF: LIA

Recursive	Verdict	Verdict type	2ls	aise	brick	bubaak	bubaak-split	cpachecker	cpv	emergenttheta	esbmc-kind	goblint	hornix	korn	mopsa	svf-svc	symbiotic	theta	thorn	uautomizer	ukojak	utaipan	Out of
False	safe	confirmed	14	3	0	115	118	43	70	43	51	11	12	0	0	9	127	2	73	111	85	85	195
		unconfirmed	0	0	0	20	20	0	0	1	1	0	0	0	0	0	22	0	0	2	1	1	
		wrong	0	0	0	18	22	0	0	0	1	0	0	0	0	0	37	0	4	5	4	2	
	unsafe	confirmed	15	24	0	19	7	25	26	13	21	25	0	0	0	0	0	20	24	28	27	27	
		unconfirmed	1	2	0	2	2	4	1	2	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	13	14	0	12	9	14	6	13	0	18	0	0	0	0	0	14	0	0	0	0	
True	safe	confirmed	21	6	0	24	25	60	96	19	18	21	0	13	13	12	336	23	24	19	132	132	557
		unconfirmed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	
		wrong	0	0	0	0	0	2	3	0	0	0	0	0	0	0	142	0	1	0	0	0	
	unsafe	confirmed	13	2	0	80	41	61	69	8	11	110	0	7	1	0	0	84	24	12	107	101	
		unconfirmed	0	0	0	2	2	1	0	0	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	3	0	0	20	16	16	1	3	0	24	0	0	0	0	0	20	1	0	0	0	

# CHCVERIF: LIA

Recursive	Verdict	Verdict type	2ls	aise	brick	bubaak	bubaak-split	cpachecker	cpv	emergenttheta	esbmc-kind	goblint	hornix	korn	mopsa	svf-svc	symbiotic	theta	thorn	uautomizer	ukojak	utaipan	Out of
False	safe	confirmed	14	3	0	115	118	43	70	43	51	11	12	0	0	9	127	2	73	111	85	85	195
		unconfirmed	0	0	0	20	20	0	0	1	1	0	0	0	0	0	22	0	0	2	1	1	
		wrong	0	0	0	18	22	0	0	0	1	0	0	0	0	0	37	0	4	5	4	2	
	unsafe	confirmed	15	24	0	19	7	25	26	13	21	25	0	0	0	0	0	20	24	28	27	27	
		unconfirmed	1	2	0	2	2	4	1	2	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	13	14	0	12	9	14	6	13	0	18	0	0	0	0	0	14	0	0	0	0	
True	safe	confirmed	21	6	0	24	25	60	96	19	18	21	0	13	13	12	336	23	24	19	132	132	557
		unconfirmed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	
		wrong	0	0	0	0	0	2	3	0	0	0	0	0	0	0	142	0	1	0	0	0	
	unsafe	confirmed	13	2	0	80	41	61	69	8	11	110	0	7	1	0	0	84	24	12	107	101	
		unconfirmed	0	0	0	2	2	1	0	0	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	3	0	0	20	16	16	1	3	0	24	0	0	0	0	0	20	1	0	0	0	

- Quite a lot wrong results
- Best single tool: around 100 UNSAT, 130 SAT

# CHCVERIF: LIA

Recursive	Verdict	Verdict type	2ls	aise	brick	bubaak	bubaak-split	cpachecker	cpv	emergenttheta	esbmc-kind	goblint	hornix	korn	mopsa	svf-svc	symbiotic	theta	thorn	uautomizer	ukojak	utaipan	Out of
False	safe	confirmed	14	3	0	115	118	43	70	43	51	11	12	0	0	9	127	2	73	111	85	85	195
		unconfirmed	0	0	0	20	20	0	0	1	1	0	0	0	0	0	22	0	0	2	1	1	
		wrong	0	0	0	18	22	0	0	0	1	0	0	0	0	0	37	0	4	5	4	2	
	unsafe	confirmed	15	24	0	19	7	25	26	13	21	25	0	0	0	0	0	20	24	28	27	27	
		unconfirmed	1	2	0	2	2	4	1	2	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	13	14	0	12	9	14	6	13	0	18	0	0	0	0	0	14	0	0	0	0	
True	safe	confirmed	21	6	0	24	25	60	96	19	18	21	0	13	13	12	336	23	24	19	132	132	557
		unconfirmed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	
		wrong	0	0	0	0	0	2	3	0	0	0	0	0	0	0	142	0	1	0	0	0	
	unsafe	confirmed	13	2	0	80	41	61	69	8	11	110	0	7	1	0	0	84	24	12	107	101	
		unconfirmed	0	0	0	2	2	1	0	0	0	2	0	0	0	0	0	2	1	0	0	0	
		wrong	3	0	0	20	16	16	1	3	0	24	0	0	0	0	0	20	1	0	0	0	

- Quite a lot wrong results
- Best single tool: around 100 UNSAT, 130 SAT
- In comparison, leading CHC solvers: 150+ UNSAT, 250+ SAT results

# CHCVERIF: BV

Recursive	Verdict	Verdict type	2ls	bubaak-split	cpachecker	emergentheta	esbmc-kind	symbiotic	theta	Out of
False	safe	confirmed	9	39	27	16	5	0	38	213
		unconfirmed	4	28	6	21	4	0	32	
		wrong	0	16	0	4	0	0	8	
	unsafe	confirmed	31	62	81	69	79	75	68	
		unconfirmed	5	8	12	3	10	10	3	
False	safe	confirmed	0	0	24	0	0	0	0	396
		unconfirmed	1	1	13	5	1	1	6	
		wrong	0	0	0	2	0	0	4	
	unsafe	confirmed	69	88	91	67	101	99	73	
		unconfirmed	4	9	10	0	18	16	4	

Verdict	eldarica	theta	Out of
sat	103	44	396
unsat	160	161	

# CHCVERIF: BV

Recursive	Verdict	Verdict type	2ls	bubaak-split	cpachecker	emergentheta	esbmc-kind	symbiotic	theta	Out of
False	safe	confirmed	9	39	27	16	5	0	38	213
		unconfirmed	4	28	6	21	4	0	32	
		wrong	0	16	0	4	0	0	8	
	unsafe	confirmed	31	62	81	69	79	75	68	
		unconfirmed	5	8	12	3	10	10	3	
	False	safe	confirmed	0	0	24	0	0	0	
unconfirmed			1	1	13	5	1	1	6	
wrong			0	0	0	2	0	0	4	
unsafe		confirmed	69	88	91	67	101	99	73	
		unconfirmed	4	9	10	0	18	16	4	

Verdict	eldarica	theta	Out of
sat	103	44	396
unsat	160	161	

- Few wrong results
- Many '*unconfirmed*' – **these are not solved by CHC solvers!**

# CHCVERIF: BV

Recursive	Verdict	Verdict type	2ls	bubaak-split	cpachecker	emergenttheta	esbmc-kind	symbiotic	theta	Out of
False	safe	confirmed	9	39	27	16	5	0	38	213
		unconfirmed	4	28	6	21	4	0	32	
		wrong	0	16	0	4	0	0	8	
	unsafe	confirmed	31	62	81	69	79	75	68	
		unconfirmed	5	8	12	3	10	10	3	
False	safe	confirmed	0	0	24	0	0	0	0	396
		unconfirmed	1	1	13	5	1	1	6	
		wrong	0	0	0	2	0	0	4	
	unsafe	confirmed	69	88	91	67	101	99	73	
		unconfirmed	4	9	10	0	18	16	4	

- Few wrong results
- Many '*unconfirmed*' – **these are not solved by CHC solvers!**

Verdict	eldarica	theta	Out of
sat	103	44	396
unsat	160	161	

**Portfolio:**  
**131 UNSAT, 86 SAT**  
*(non-refuted)*

# Significance & Summary

Why should you care?

# Has this benefitted CHC solving?

# Has this benefitted CHC solving?

- We believe so...

# Has this benefitted CHC solving?

- We believe so...

**New solvers (for BV out-of-the-box)**

# Has this benefitted CHC solving?

- We believe so...

**New solvers (for BV out-of-the-box)**

**Previously unsolved tasks**

# Has this benefitted CHC solving?

- We believe so...

**New solvers (for BV out-of-the-box)**

**Previously unsolved tasks**

**More competition, more visibility**

# And software verification?

- We are sure!

# And software verification?

- We are sure!

## 6.3.1.2 Boolean type

ISO/IEC 9899:202x

- 1 When any scalar value is converted to `_Bool`, the result is 0 if the value compares equal to 0; otherwise, the result is 1.<sup>60)</sup>

# And software verification?

- We are sure!

## 6.3.1.2 Boolean type

ISO/IEC 9899:202x

- 1 When any scalar value is converted to `_Bool`, the result is 0 if the value compares equal to 0; otherwise, the result is 1.<sup>60)</sup>

```
extern _Bool __VERIFIER_nondet_Bool();
extern void reach_error();
int main() {
    _Bool b = __VERIFIER_nondet_Bool();
    switch(b) {
        case 0: return 0;
        case 1: return 0;
    }
    reach_error(); // never called?
}
```

# And software verification?

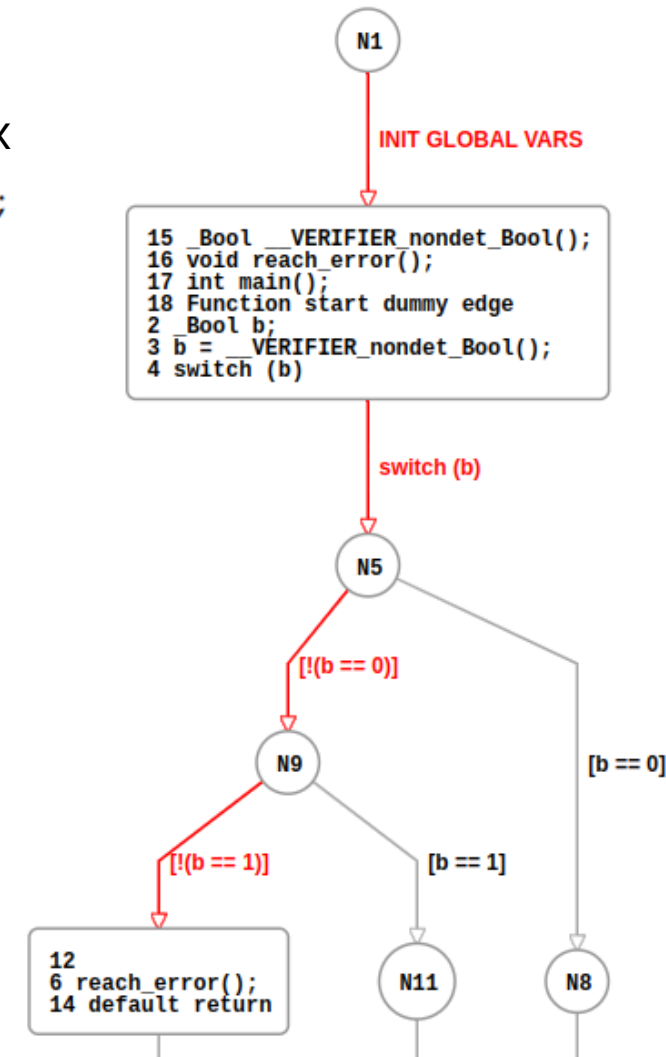
- We are sure!

## 6.3.1.2 Boolean type

ISO/IEC 9899:202x

- 1 When any scalar value is converted to **\_Bool**, the result is 0 if the value compares equal to 0; otherwise, the result is 1.<sup>60)</sup>

```
extern _Bool __VERIFIER_nondet_Bool();
extern void reach_error();
int main() {
    _Bool b = __VERIFIER_nondet_Bool();
    switch(b) {
        case 0: return 0;
        case 1: return 0;
    }
    reach_error(); // never called?
}
```



# And software verifi

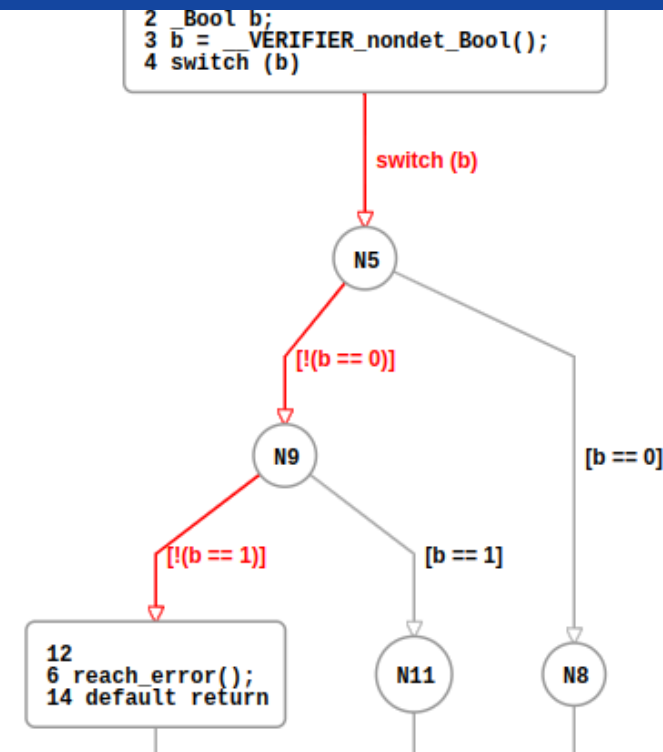
- We are sure!

## 6.3.1.2 Boolean type

- 1 When any scalar value is converted to `_Bool`, the result is 1. <sup>60)</sup> otherwise, the result is 0.

```
extern _Bool __VERIFIER_nondet_Bool();
extern void reach_error();
int main() {
    _Bool b = __VERIFIER_nondet_Bool();
    switch (b) {
        case 0: return 0;
        case 1: return 0;
    }
    reach_error(); // never called?
}
```

2ls	bubaak	bubaak-split	cpachecker	cpv	emergenttheta	esbmc-kind	goblint	infer	mopsa	symbiotic	theta	uautomizer	ukojak	utaiipan	veriabs	veriabsl
✓	✓	✓	×	?	✓	✓	✓	?	?	✓	✓	×	×	×	✓	✓



# And software verifi

- We are sure!

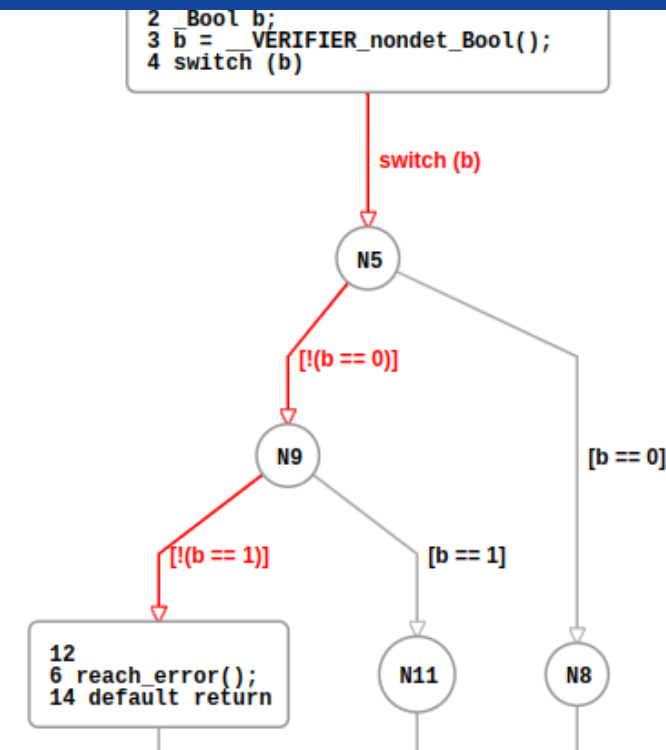
## 6.3.1.2 Boolean type

- 1 When any scalar value is converted to **\_Bool**, the result is otherwise, the result is 1.<sup>60)</sup>

```
extern _Bool __VERIFIER_nondet_Bool();
extern void reach_error();
int main() {
    _Bool b = __VERIFIER_nondet_Bool();
    switch(b) {
        case 0: return 0;
        case 1: return 0;
    }
    reach_error(); // never called?
}
```

**CPAchecker: already fixed!**


2ls	bubaak	bubaak-split	cpachecker	cpv	emergenttheta	esbmc-kind	goblint	infer	mopsa	symbiotic	theta	uautomizer	ukojak	utaiipan	veriabs	veribsl
✓	✓	✓	×	?	✓	✓	✓	?	?	✓	✓	×	×	×	✓	✓



# And software verification?

- We are sure!

## Added CHC benchmarks

 Open Levente Bajczi requested to merge [levente.bajczi/sv-benchm...](#) into [main](#) 5 months ago

Overview **4** Commits **10** Pipelines **6** Changes **333+**

This PR adds 2774 new benchmarks to the repository. These are transformed CHC benchmarks, originally sourced from the <https://github.com/chc-comp/> organization to be used for CHC-COMP, which have been converted into CFAs and then to C files. See our (accepted, pending publication) paper at HCVS'23 on the transformation of CHC problems to CFAs: [submitted.pdf](#)

**New benchmarks**