Temporal Property guided Program Analysis/Repair

Yahui Song

Research Fellow @ National University of Singapore (NUS)

September 2024

My Research

- PhD (2018 Aug – 2023 May)

Thesis: Symbolic Temporal Verification Techniques with Extended Regular Expressions

Keywords: Modularly (Scalability), Expressive Specification, Hoare-style Verification (source code level)

Applications
- Event-based reactive systems [ICFEM 2020]
- Synchronous languages like Esterel [VMCAI 2021]
- User-defined algebraic effects and handlers [APLAS 2022]
- Real-time systems [TACAS 2023]

- Research Fellow (2023 – now)

My Research

- PhD (2018 Aug – 2023 May)

  Thesis: Symbolic Temporal Verification Techniques with Extended Regular Expressions

  Keywords: Modularly (Scalability), Expressive Specification, Hoare-style Verification (source code level)

  Applications
  - Event-based reactive systems [ICFEM 2020]
  - Synchronous languages like Esterel [VMCAI 2021]
  - User-defined algebraic effects and handlers [APLAS 2022]
  - Real-time systems [TACAS 2023]

- Research Fellow (2023 – now)

  Staged Specification Logic (heap safety):

  Higher-order Imperative Programs [FM 2024]; Algebraic Effects and Handlers [ICFP 2024]

  Temporal Property guided Program Analysis/Repair:

  Linear Temporal Property [FSE 2024]

  Computation Tree Logic + Precise Loop Summaries [Under Submission]

ProveNFix: Temporal Property guided Program Repair

Yahui Song, Xiang Gao, Wenhua Li, Wei-Ngan Chin, Abhik Roychoudhury

National University of Singapore

北京航空航天大學
BEIHANG UNIVERSITY

Can temporal property analysis be modular?

“Each function is analysed only once and

can be replaced by their verified properties.”

Can temporal property analysis be modular?

“Each function is analysed only once and

can be replaced by their verified properties.”

Modular Analysis:

1. Assume-guarantee paradigm (divide and conquer)
2. A set of forward/backwards reasoning rules

Some Forward Reasoning Rules

Entailment Checking

$$\frac{\vdash \{\Phi_{pre}\}\ \mathtt{e}\ \{\Phi_C\} \qquad \vdash \Phi_C \sqsubseteq \Phi_{post}}{\vdash \tau\ \mathbf{mn}\ (\tau\ \mathbf{x})^*\ \{\mathbf{requires}\ \Phi_{pre}\ \mathbf{ensures}\ \Phi_{post}\}\ \{\mathtt{e}\}}\ [\mathtt{FV\text{-}Meth}]$$

$$\frac{\Phi'_C = \Phi_C \cdot \underline{\mathbf{a}}}{\vdash \{\Phi_C\}\ \mathbf{event}[\underline{\mathbf{a}}]\ \{\Phi'_C\}}\ [\mathtt{FV\text{-}Event}]$$

$$\frac{\vdash \{\Phi_C\}\ \mathtt{e_1}\ \{\Phi'_C\} \qquad \vdash \{\Phi'_C\}\ \mathtt{e_2}\ \{\Phi''_C\}}{\vdash \{\Phi_C\}\ \mathtt{e_1};\mathtt{e_2}\ \{\Phi''_C\}}\ [\mathtt{FV\text{-}Seq}]$$

$$\frac{\vdash \{\mathbf{v} \wedge \Phi_C\}\ \mathtt{e_1}\ \{\Phi'_C\} \qquad \vdash \{\neg\mathbf{v} \wedge \Phi_C\}\ \mathtt{e_2}\ \{\Phi''_C\}}{\vdash \{\Phi_C\}\ \mathbf{if\ v\ then}\ \mathtt{e_1}\ \mathbf{else}\ \mathtt{e_2}\ \{\Phi'_C \vee \Phi''_C\}}\ [\mathtt{FV\text{-}If\text{-}Else}]$$

? $[\mathtt{FV\text{-}Call}]$

Can temporal property analysis be modular?

“Each function is analysed only once and

can be replaced by their verified properties.”

Modular Analysis:

1. Assume-guarantee paradigm (divide and conquer)
2. A set of forward/backwards reasoning rules
3. Entailment/Inclusion Checking : $x > 1 \sqsubseteq x > 0$

Can temporal property analysis be modular?

“Each function is analysed only once and

can be replaced by their verified properties.”

Three main difficulties：

1. Temporal logic property entailment checker.
2. Writing temporal specifications for each function is tedious and challenging.
3. The classic pre/post-conditions is not enough, e.g.,

   “some meaningful operations can only happen if the return value of loading the certificate is positive”

Future-condition

Defined in header `<stdlib.h>`		
`void free( void* ptr );`		

```
void free (void *ptr);
// post: (ptr=null ∧ ε) ∨ (ptr≠null ∧ free(ptr))
// future: true ∧ 𝒢 (!_(ptr))
```

The behavior is undefined if after `free()` returns, an access is made through the pointer `ptr` (unless another allocation function happened to result in a pointer value equal to `ptr`).

Defined in header `<stdlib.h>`		
`void* malloc( size_t size );`		

On success, returns the pointer to the beginning of newly allocated memory. To avoid a memory leak, the returned pointer must be deallocated with `free()` or `realloc()`.

On failure, returns a null pointer.

```
void *malloc (size_t size);
// pre: size>0 ∧ _*
// post: (ret=null ∧ ε) ∨ (ret≠null ∧ malloc(ret))
// future: ret≠null → ℱ (free(ret))
```

Future-condition based modular analysis

$$\dfrac{nm(x^*) \mapsto (\Phi_{pre}, \Phi_{post}) \in \mathcal{E} \quad \Phi \sqsubseteq [y^*/x^*]\Phi_{pre} \quad \Phi'_{post} = [r/ret, y^*/x^*]\Phi_{post} \quad \mathcal{E} \vdash \{\Phi \cdot \Phi'_{post}\}\ e\ \{\Phi_e\}}{\mathcal{E} \vdash \{\Phi\}\ r = nm(y^*);\ e\ \{\Phi'_{post} \cdot \Phi_e\}}\ [\mathit{FR\text{-}Call}]$$

Entailment Checking → $\Phi \sqsubseteq [y^*/x^*]\Phi_{pre}$

A collection of specifications → $\mathcal{E}$

Future-condition based modular analysis

Entailment Checking

A collection of specifications

$$\frac{nm(x^*) \mapsto (\Phi_{pre}, \Phi_{post}, \Phi_{future}) \in \mathcal{E} \quad \Phi \sqsubseteq [y^*/x^*]\Phi_{pre} \quad \Phi'_{post} = [r/ret, y^*/x^*]\Phi_{post} \quad \mathcal{E} \vdash \{\Phi \cdot \Phi'_{post}\}\ e\ \{\Phi_e\} \quad \Phi_e \sqsubseteq [r/ret, y^*/x^*]\Phi_{future}}{\mathcal{E} \vdash \{\Phi\}\ r = nm(y^*);\ e\ \{\Phi'_{post} \cdot \Phi_e\}} \ [FR\text{-}Call]$$

Can temporal property analysis be modular?

"Each function is analysed only once and

can be replaced by their verified properties."

Three main difficulties:

1. Temporal logic property entailment checker.
2. Writing temporal specifications for each function is tedious and challenging.
3. ~~The classic pre/post-conditions is not enough, e.g.,~~ **Future-condition!**

   ~~"some meaningful operations can only happen if the return value of loading the certificate is positive"~~

Specification inference via bi-abduction

```
void *malloc (size_t size);
// future: (ret=null ∧ 𝒢 (!_(ret))) ∨ (ret≠null ∧ ℱ (free(ret))
```

```
void wrap_malloc_I (int* ptr)
// future: ptr=null ∧ 𝒢 (!_(ptr))
        ∨ ptr≠null ∧ ℱ (free(ptr))
{ ptr = malloc (4); return;}
```

Specification inference via bi-abduction

```
void *malloc (size_t size);
// future: (ret=null ∧ 𝒢 (!_(ret))) ∨ (ret≠null ∧ ℱ (free(ret))
```

```
void wrap_malloc_I (int* ptr)
// future: ptr=null ∧ 𝒢 (!_(ptr))
        ∨ ptr≠null ∧ ℱ (free(ptr))
{ ptr = malloc (4); return;}
```

```
int* wrap_malloc_II ()
// future: ret=null ∧ 𝒢 (!_(ret))
        ∨ ret≠null ∧ ℱ (free(ret))
{ int* ptr = malloc (4); return ptr;}
```

Specification inference via bi-abduction

```
void *malloc (size_t size);
// future: (ret=null ∧ 𝒢 (!_(ret))) ∨ (ret≠null ∧ ℱ (free(ret))
```

```
int* wrap_malloc_III ()
// future: true ∧ ℱ (free(ret))
{ int* ptr = malloc (4);
  if (ptr == NULL) exit(-1);
  return ptr;}
```

Specification inference via bi-abduction

```
void *malloc (size_t size);
// future: (ret=null ∧ 𝒢 (!_(ret))) ∨ (ret≠null ∧ ℱ (free(ret))
```

```
int* wrap_malloc_III ()
// future: true ∧ ℱ (free(ret))
{ int* ptr = malloc (4);
  if (ptr == NULL) exit(-1);
  return ptr;}
```

```
int* wrap_malloc_IV ()
// future: true ∧ _*
{ int* ptr = malloc (4);

  return NULL;}
```

Failed entailment: $\text{true} \wedge \mathcal{E} \not\sqsubseteq \text{ptr}\neq\text{null} \wedge \mathcal{F}\ (\text{free}(\text{ptr}))$

Can temporal property analysis be modular?

"Each function is analysed only once and

can be replaced by their verified properties."

Three main difficulties：

1. Temporal logic property entailment checker. **Primitive spec + spec inference!**
2. ~~Writing temporal specifications for each function is tedious and challenging.~~
3. ~~The classic pre/post-conditions is not enough, e.g.,~~ **Future-condition!**

   ~~"some meaningful operations can only happen if the return value of loading the certificate is positive"~~

Term rewriting system for regular expressions

- Flexible specifications, which can be combined with other logic;
- Efficient entailment checker with inductive proofs.

$$
\begin{array}{llll}
(\mathit{IntRE}) & \Phi & ::= & \bigvee(\pi \wedge \theta) \\
(\mathit{Traces}) & \theta & ::= & \bot \mid \epsilon \mid \mathrm{I} \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \theta^\star \\
(\mathit{Events}) & \mathrm{I} & ::= & \mathrm{A}(v) \mid \mathrm{A}(\_) \mid !\mathrm{A}(v) \mid !\_(v) \mid \_ \mid \mathrm{I}_1 \wedge \mathrm{I}_2 \\
(\mathit{Pure}) & \pi & ::= & T \mid F \mid bop(t_1, t_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg\pi \mid \exists x.\pi \\
(\mathit{Terms}) & t & ::= & v \mid t_1 + t_2 \mid t_1 - t_2 \\
(\mathit{Values}) & v & ::= & c \mid x \mid null
\end{array}
$$

Fig. 10. Syntax of the spec language, *IntRE*.

Term rewriting system for regular expressions

- Flexible specifications, which can be combined with other logic;
- Efficient entailment checker with inductive proofs.

Examples:

$x>2 \land E \sqsubseteq x>1 \land (E \lor F)$

$x>0 \land E \not\sqsubseteq x>1 \land (E \lor F)$

$true \land E \not\sqsubseteq true \land (E . F)$

$$\dfrac{\dfrac{\dfrac{(a \lor b)^{\star} \sqsubseteq (a \lor b \lor bb)^{\star} \quad [\text{Reoccur}]}{\epsilon \cdot (a \lor b)^{\star} \sqsubseteq \epsilon \cdot (a \lor b \lor bb)^{\star}}}{a \cdot (a \lor b)^{\star} \sqsubseteq (a \lor b \lor bb)^{\star}} \qquad \dfrac{[\text{Reoccur}]}{b \cdot (a \lor b)^{\star} \sqsubseteq \ldots}}{(a \lor b)^{\star} \sqsubseteq (a \lor b \lor bb)^{\star}}$$

Can temporal property analysis be modular?

Can!

"Each function is analysed only once and

can be replaced by their verified properties." ✓

Three main difficulties:

A term rewriting system for regular expressions

1. ~~Temporal logic property entailment checker.~~

Primitive spec + spec inference!

2. ~~Writing temporal specifications for each function is tedious and challenging.~~

3. ~~The classic pre/post-conditions is not enough, e.g.,~~ **Future-condition!**

   ~~"some meaningful operations can only happen if the return value of loading the certificate is positive"~~

Experiment 1: detecting bugs

Primitive APIs	Pre	Post	Future	Targeted Bug Type
open/socket/fopen/fdopen/opendir	✗	✗	✓	Resource Leak
close/fclose/endmntent/fflush/closedir	✗	✓	✗	
malloc/realloc/calloc/localtime	✗	✗	✓	Null Pointer Dereference
→ *(pointer dereference)*	✗	✓	✗	
malloc	✓	✓	✓	Memory Usage (Leak, Use-After-Free, Double Free)
free	✓	✓	✓	

- ❖ 17 predefined primitive specs.
- ❖ ProveNFix is finding 72.2% more true bugs, with a 17% loss of missing true bugs.

Project	kLoC	#NPD		#ML		#RL		Time	
		Infer	ProveNFix	Infer	ProveNFix	Infer	ProveNFix	Infer	ProveNFix
Swoole(a4256e4)	44.5	30+7	30+23	16+4	12+16	13+1	13+6	2m 50s	39.54s
lxc(72cc48f)	63.3	7+9	5+19	11+6	10+12	5+1	5+5	55.62s	1m 28s
WavPack(22977b2)	36	23+7	20+21	3	3+9	0+2	0	27.99s	23.77s
flex(d3de49f)	23.9	14+4	14+4	3	3+1	0	0+1	32.25s	47.75s
p11-kit	76.2	3+5	2+2	13+3	12+15	5	5+1	1m 57s	1m 4s
x264(d4099dd)	67.7	0	0	12	11+5	2	2+3	2m 33s	23.168s
recutils-1.8	81.9	25	22+8	13+10	11+29	1	1+7	9m 10s	38.29s
inetutils-1.9.4	117.2	7+4	5+8	9+3	7+10	1	1+5	30.26s	1m 5s
snort-2.9.13	378.2	44+12	33+34	26+4	15+16	1+2	1+1	8m 49s	3m 13s
grub(c6b9a0a)	331.1	13+12	6+5	1	1	0+3	0	3m 27s	1m 1s
Total	1,220.00	166+60	137+124	107+30	85+113	26+9	27+29	31m 12s	10m 44s

Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

Require: $\mathcal{E}, (\pi \land \theta_{target})$

Ensure: An expression e_R such that $\mathcal{E} \vdash \{T \land \epsilon\}\ e_R\ \{\pi \land \theta_{target}\}$

$e_{acc} = ()$
for each $nm\ (x^*) \mapsto [\Phi_{pre}, \Phi_{post}, \Phi_{future}] \in \mathcal{E}$ **do**
  if $\theta_{target} = \epsilon$ **then return** `if` π `then` e_{acc} `else` $()$
  else
    `// there exist a set of program variables` y^*
    $\theta'_{target} = (\pi \land [y^*/x^*]\Phi_{post})^{-1}\theta_{target}$
    $e_{acc} = e_{acc};\ nm(y^*)$
  end if
end for
return without any suitable patches

Example: `true` $\land\ \mathcal{E} \not\models$ `ptr≠null ∧ _^*.(free(ptr))`

$\Rightarrow$ synthesis(`ptr≠null ∧ _^*.(free(ptr))`) $\Rightarrow$ `if (ptr != NULL) free(ptr);`

Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

Require: $\mathcal{E}, (\pi \wedge \theta_{target})$

Ensure: An expression e_R such that $\mathcal{E} \vdash \{T \wedge \epsilon\}\ e_R\ \{\pi \wedge \theta_{target}\}$

$e_{acc} = ()$

for each $nm\ (x^*) \mapsto [\Phi_{pre}, \Phi_{post}, \Phi_{future}] \in \mathcal{E}$ **do**

    if $\theta_{target} = \epsilon$ **then return** `if` π `then` e_{acc} `else` $()$

    else

        `// there exist a set of program variables` y^*

        $\theta'_{target} = (\pi \wedge [y^*/x^*]\Phi_{post})^{-1}\theta_{target}$

        $e_{acc} = e_{acc};\ nm(y^*)$

    end if

end for

return without any suitable patches

- Only supporting inserting/deleting calls.
- Do need re-analysis.

Example: `true` $\wedge$ $\mathcal{E}$ $\not\sqsubseteq$ `ptr≠null` $\wedge$ `_^*.(free(ptr))`

$\Rightarrow$ synthesis(`ptr≠null` $\wedge$ `_^*.(free(ptr))`) $\Rightarrow$ `if (ptr != NULL) free(ptr);`

Experiment 2: Repairing bugs

Project	NPD		ML		RL		Time	Infer-v0.9.3			
	#	ProveNFix	#	ProveNFix	#	ProveNFix		#ML	SAVER	#RL	FootPatch
Swoole	53	53	32	28	19	19	4.33s	15+3	11	6+1	6
lxc	26	24	23	22	10	10	3.882s	3+5	3	2+1	0
WavPack	44	41	12	12	0	0	11.435s	1+2	0	2	1
flex	18	18	4	4	1	1	39.38s	3+4	0	0	0
p11-kit	5	4	28	27	6	6	2.452s	33+9	24	2	1
x264	0	0	17	14	5	5	6.375s	10	10	0	0
recutils-1.8	33	30	42	36	8	8	1.261s	10+11	8	1	0
inetutils-1.9.4	15	13	19	17	6	6	1.517s	4+5	4	2+1	1
snort-2.9.13	78	67	42	13	2	2	10.57s	16+27	10	0	0
grub	18	11	1	1	0	0	40.626s	0	0	0	0
Total(Fix Rate)	290	261(90%)	220	174 (79%)	57	57 (100%)	2m 2s	95+66	70(73.7%)	15+3	9(60%)

- 90% fix - null pointer dereferences,
- 79% fix - memory leaks
- 100% fix - resource leaks.

SAVER's pre-analysis time:
26.3 seconds for the flex project
39.5 minutes for the snort-2.9.13 project

Experiment 4: usefulness of spec inference

- 2 predefined primitive specs, OpenSSL-3.1.2, 556.3 kLoC,
- 143.11 seconds to generate future-conditions for 128 OpenSSL APIs
- Example: SSL_CTX_new (meth) ; // future : ((ret=0) /\ return (ret))

OpenSSL Applications	kLoC	Issue ID	Target API	Github Status	ProveNFix	Time
keepalive(843ffc80)	59.1	1003	SSL_CTX_new	✓	✓	5.62s
		1004	SSL_new	✓	✓	
thc-ipv6(011376c)	30.9	28	BN_new	✓	✓	3.32s
		29	BN_set_word	✓	✗	
FreeRADIUS(94149dc)	258.9	2309	BIO_new	✓	✓	38.89s
		2310	i2a_ASN1_OBJECT	✓	✓	
trafficserver(5ee6a5f)	34.1	4292	SSL_CTX_new	✓	✓	21.55s
		4293	SSL_new	✓	✓	
		4294	SSL_write	✓	✓	
sslsplit(19a16bd)	18.7	224	SSL_CTX_use_certificate	✓	✓	2.69s
		225	SSL_use_PrivateKey	✓	✓	
proxytunnel(f7831a2)	3.1	36	SSL_connect	✓	✓	0.62s
		37	SSL_new	✓	✓	

Summary

- Compositional static analyzer via temporal properties.
- Specified 17 APIs; found 515 bugs from 1 million LOC; (on average) 90% fix rate.
- Specification: a novel *future-condition.*
- Specification inference via bi-abduction.
- The inferred spec can be used to analysis protocol applications, e.g., OpenSSL.

Take away

❖ Specify a small set of properties once and analyse/repair a large number of programs

❖ Specification inference enabled by projecting global spec to local spec.

Computation Tree Logic Guided Program Repair With Precise Loop Summaries

Yu Liu*, Yahui Song*, Martin Mirchev, Sergey Mechtaev, Abhik Roychoudhury

Computational Tree Logic

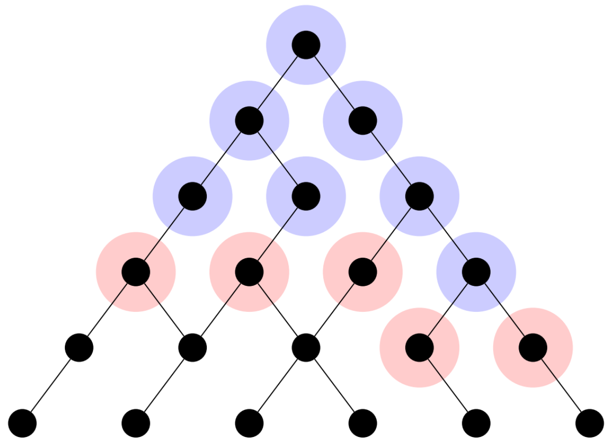
- Branching-time logic:

$$\phi \ ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi$$
$$\mid \mathbf{AX}\ \phi \mid \mathbf{EX}\ \phi \mid \mathbf{AF}\ \phi \mid \mathbf{EF}\ \phi \mid \mathbf{AG}\ \phi \mid \mathbf{EG}\ \phi$$
$$\mid \mathbf{A}[\phi\ \mathbf{U}\ \phi] \mid \mathbf{E}[\phi\ \mathbf{U}\ \phi]$$

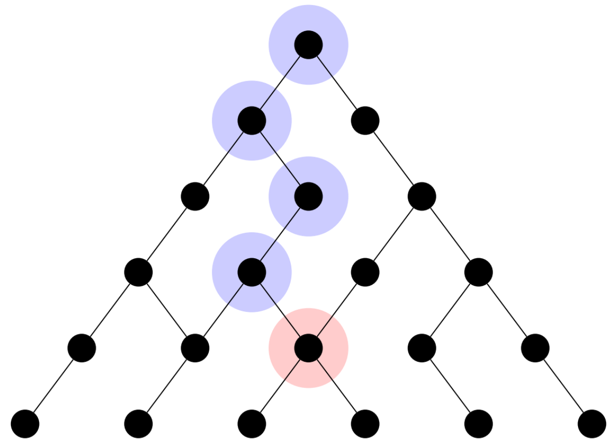
- Goals:

  - a more precise analysis for CTL properties in real code

  - automated repair when CTL violations occur



A [blue U red]



E [blue U red]

Example modified from “Reasoning about Nondeterminism in Programs” PLDI 2013

CTL Properties and Violations

```
1  x = 0;
2  while (true) {
3    y = *;
4    x = 1;
5    n = *;
6    while (n>0) {
7      n = n - y;
8    }
9    x = 0;
10 }
```

“Whenever x = 1, then eventually x = 0.”

$$EG(x=1 \Rightarrow AF(x=0))$$ ✗

If we restrict the nondeterministic choice at line 3

To be y >= 1, the the following holds as well.

$$AG(x=1 \Rightarrow AF(x=0))$$ ✓

Example modified from “Reasoning about Nondeterminism in Programs” PLDI 2013

CTL Properties and Violations

```
1  x = 0;
2  while (true) {
3    y = *;
4    x = 1;
5    n = *;
6    while (n>0) {
7      n = n - y;
8    }
9    x = 0;
10 }
```

“Whenever x = 1, then eventually x = 0.”

$$EG(x=1 \Rightarrow AF(x=0))$$ ✗

If we restrict the nondeterministic choice at line 3

To be y >= 1, the the following holds as well.

$$AG(x=1 \Rightarrow AF(x=0))$$ ✓

> **“Termination is a sub-problem of liveness properties. ”**
>
> --- *[POPL07, TACAS12, CAV2015, POPL18, PLDI19, PLDI21]*

Existing analyses for CTL

- **CTL model checking**:

  Recursively labeling the states of a finite state machine with the CTL sub-formula.

  Termination analysis: none

- **Faster temporal reasoning for infinite-state programs (T2 [PLDI 13, FMCAD 14]):**

  Iteratively synthesize preconditions asserting the satisfaction of CTL sub-formulas

  Termination analysis: counterexample-based ranking function synthesis

- **Abstract interpretation of CTL properties (Function [ESOP 17]):**

  Mixed usage of over-approximation (∀), and under-approximation for (∃).

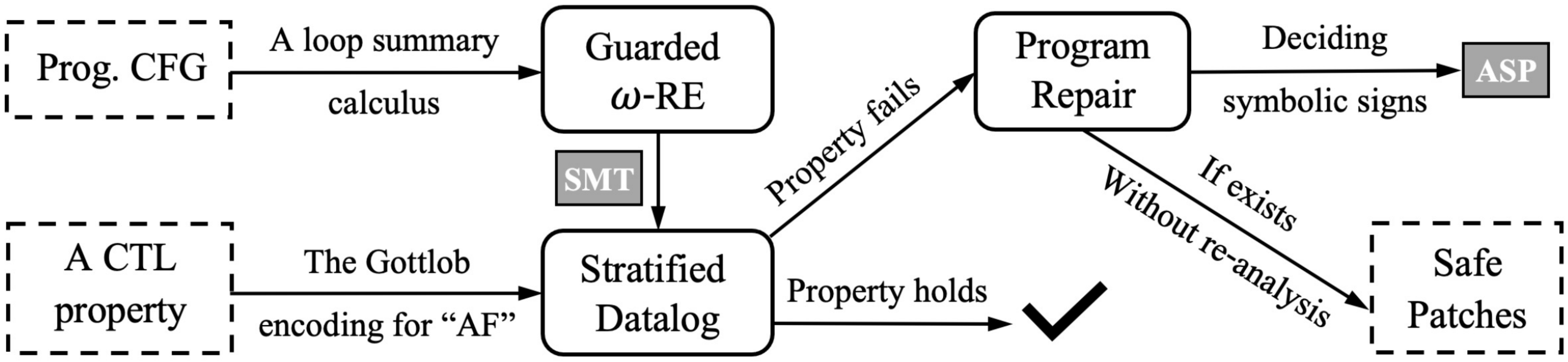
  Termination analysis: using widening and dual widening at loop heads

Table 1. Experimental results for CTL analysis, comparing with FUNCTION and T2. Here, "**Exp.**" marks the expected results, and "**Time**" records the execution times (in seconds). For each tool, we use "✓", "✗", and "?" to represent the proved, disproved, and unknown return results, respectively. Moreover, we use "-" when the tool cannot parse the formula or the input program, and "TO" represents a timeout with a 30-second limit.

	Program	**Loc**	**CTL Property**	**Exp.**	**FUNCTION**		**T2**		**CTLEXPERT**	
					Res.	**Time**	**Res.**	**Time**	**Res.**	**Time**
1	AF_terminate	25	AF(Exit())	✗	?	0.021	?	0.414		
2	toylin1 (Fig. 21)	32	EF(resp≥5)	✓	?	0.064	✗	0.294		
3	timer-simple …	26	AG((timer_1=0→AF(output_1=1)))	✓	?	1.739	✗	0.867		
4	AGAF…(Fig. 4)	16	AG((AF(t=1))∧(AF(t=0)))	✓	?	0.034	✗	0.597		
5	coolant_basi …	76	AU(init=0)(AU(init=1)(AG(init=3)))	✓	?	6.615	-	-		
6	AF_Bangalo …	22	AG((y<1)→AF(x<0))	✓	?	0.345	✗	0.249		
7	AFParity...(Fig. 2)	14	AF(y=1)	✓	?	0.012	?	0.362		
8	Nested... (Fig. 15)	20	AF(Exit())	✓	?	0.196	?	0.553	???	
9	acqrel.c	42	AG((A=1)→AF(R=0))	✓	✓	0.040	✗	0.786		
10	test_existent…	23	EF(r=1)	✓	?	0.022	✗	0.283		
11	test_global.c	14	AF(AG(y>0))	✓	?	0.219	✓	0.694		
12	test_until.c	13	AU(x>y)(x≤y)	✗	✓	0.033	-	-		
13	next.c	7	AX(AX(x=0))	✗	?	0.005	TO	-		
14	multiChoice.c	39	AF((x=4)∨(x=-4))	✓	✓	0.077	✓	0.409		
15	multiChoice.c	39	EF(x=4) ∧ EF(x=-4)	✓	?	0.086	✓	0.296		
	Total	408			13.3%	9.509	20%	5.804		

We propose "CTLexpert"

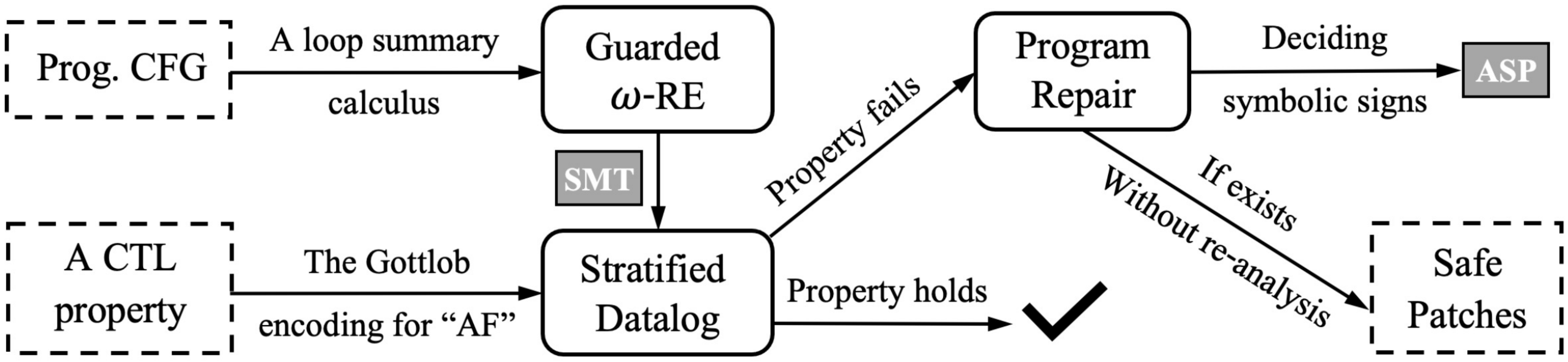
$$(\textit{Guarded } \omega\textit{-RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$



1. CTL property $\Rightarrow$ Stratified Datalog rules
2. Target program (CFG) $\Rightarrow$ Guarded ω-regular expression $\Rightarrow$ Datalog facts/rules
3. The Datalog execution checks CTL properties precisely
4. When buggy, Datalog based repair comes in

We propose "CTLexpert"

$$(\textit{Guarded } \omega\textit{-RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$



1. CTL property ⇒ Stratified Datalog rules
2. Target program (CFG) ⇒ Guarded ω-regular expression ⇒ Datalog facts/rules
3. The Datalog execution checks CTL properties precisely
4. When buggy, Datalog based repair comes in

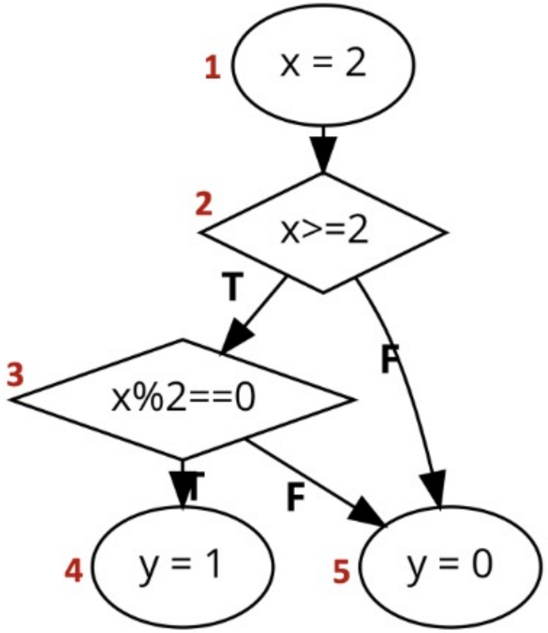
Goals/Benefits:

1. Precise loop summaries
2. Find all the repair solutions

CFG to Datalog

```
//AF(y=1)
x = 2;
if (x>=2 && x%2==0){
  y = 1;
} else {
  y = 0;
}
```

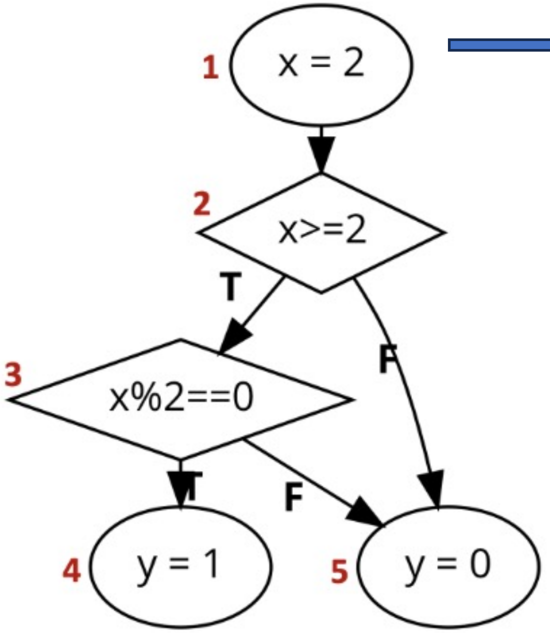
```
y_eq_1(S)           :- State(S), Eq("y",1,S).
AF_y_eq_1_T(S,S1)   :- !y_eq_1(S), flow(S,S1).
AF_y_eq_1_T(S,S1)   :- AF_y_eq_1_T(S,S2), !y_eq_1(S2), flow(S2,S1).
AF_y_eq_1_S(S)      :- AF_y_eq_1_T(S,S).
AF_y_eq_1_S(S)      :- !y_eq_1(S), flow(S,S1), AF_y_eq_1_S(S1).
AF_y_eq_1(S)        :- State(S), !AF_y_eq_1_S(S).
```



CFG to Datalog

```
//AF(y=1)
x = 2;
if (x>=2 && x%2==0){
  y = 1;
} else {
  y = 0;
}
```

```
y_eq_1(S)            :- State(S), Eq("y",1,S).
AF_y_eq_1_T(S,S1)    :- !y_eq_1(S), flow(S,S1).
AF_y_eq_1_T(S,S1)    :- AF_y_eq_1_T(S,S2), !y_eq_1(S2), flow(S2,S1).
AF_y_eq_1_S(S)       :- AF_y_eq_1_T(S,S).
AF_y_eq_1_S(S)       :- !y_eq_1(S), flow(S,S1), AF_y_eq_1_S(S1).
AF_y_eq_1(S)         :- State(S), !AF_y_eq_1_S(S).
```



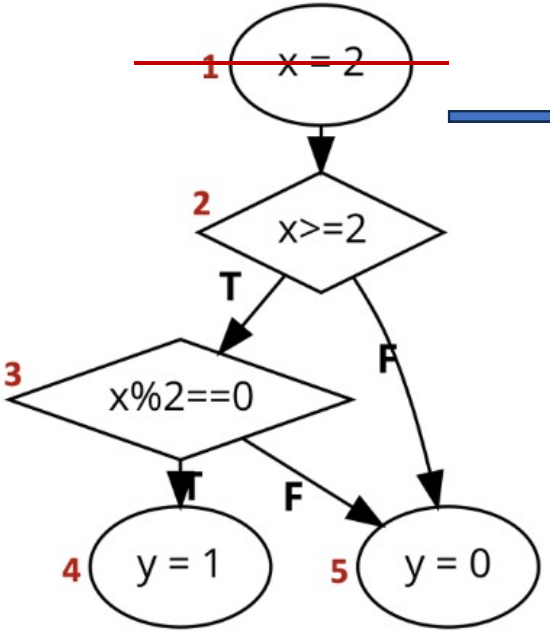

```
Even("x", 1). GtEq("x", 2, 1). // generated from "x=2"

Eq("y", 1, 4). flow (1, 2). flow (4, 4). flow (5, 5).
flow (2, 3):- GtEq("x", 2, 2).
flow (3, 4):- Even("x", 2).
flow (2, 5):- Lt ("x", 2, 2).
flow (3, 5):- Odd ("x", 2).
```

Disabled transitions!

CFG to Datalog

```
//AF(y=1)
x = 2;
if (x>=2 && x%2==0){
  y = 1;
} else {
  y = 0;
}
```

```
y_eq_1(S)             :- State(S), Eq("y",1,S).
AF_y_eq_1_T(S,S1) :- !y_eq_1(S), flow(S,S1).
AF_y_eq_1_T(S,S1) :- AF_y_eq_1_T(S,S2), !y_eq_1(S2), flow(S2,S1).
AF_y_eq_1_S(S)      :- AF_y_eq_1_T(S,S).
AF_y_eq_1_S(S)      :- !y_eq_1(S), flow(S,S1), AF_y_eq_1_S(S1).
AF_y_eq_1(S)         :- State(S), !AF_y_eq_1_S(S).
```




```
Even("x", 1). GtEq("x", 2, 1). // generated from "x=2"
Even("x", 1). GtEq("x", 2, 1). Odd("x", 1). Lt("x", 2, 1).
Eq("y", 1, 4). flow (1, 2). flow (4, 4). flow (5, 5).
flow (2, 3):- GtEq("x", 2, 2).
flow (3, 4):- Even("x", 2).
flow (2, 5):- Lt  ("x", 2, 2).
flow (3, 5):- Odd ("x", 2).
```

Patches: (1) deleting the newly added "Odd" and "Lt" facts

(2) adding a predicate "Eq("y",1, 5)"

Loops to Guarded ω-RE

$$(\textit{Guarded } \omega\textit{-RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$

```
void main () { //AF(Exit())
  int m,n; int step=8;
  while (1) {
    m = 0;
    while (m < step){
      if (n < 0) return;
      else {
        m = m + 1;
        n = n - 1; }}}}
```

$$\begin{cases} [m \geq step] \cdot \epsilon \ \vee & (1) \\ [m < step \wedge n < 0] \cdot Exit() \ \vee & (2) \\ ([m < step \wedge n \geq 0] \cdot (m' = m+1) \cdot (n' = n\text{-}1))^\star & (3) \end{cases}$$

Loops to Guarded ω-RE

$$(\textit{Guarded } \omega\text{-}\textit{RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$

```
void main () { //AF(Exit())
  int m,n; int step=8;
  while (1) {
    m = 0;
    while (m < step){
      if (n < 0) return;
      else {
        m = m + 1;
        n = n - 1; }}}}
```

- Inner loop: RF = {step-m-1, n}

$$\Phi_{inner} \equiv \begin{cases} [(step\text{-}m\text{-}1) \geq n] \cdot (n' < 0) \cdot Exit() \;\; \vee \\ [(step\text{-}m\text{-}1) < n] \cdot (m' \geq step) \cdot (n' = n\text{-}(step\text{-}m)) \end{cases}$$

$$\begin{cases} [m \geq step] \cdot \epsilon \;\; \vee & (1) \\ [m < step \wedge n < 0] \cdot Exit() \;\; \vee & (2) \\ ([m < step \wedge n \geq 0] \cdot (m' = m+1) \cdot (n' = n\text{-}1))^{\star} & (3) \end{cases}$$

Ranking function: when RF >= 0, stays in the loop, and when RF < 0, exits the loop.

Loops to Guarded ω-RE

$$(\textit{Guarded } \omega\textit{-RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$

```
void main () { //AF(Exit())
  int m,n; int step=8;
  while (1) {
    m = 0;

                 rn;

                 }}
```

Φ_{inner}

- Inner loop: RF = {step-m-1, n}

$$\Phi_{inner} \equiv \begin{cases} [(step\text{-}m\text{-}1) \geq n] \cdot (n' < 0) \cdot Exit() \ \vee \\ [(step\text{-}m\text{-}1) < n] \cdot (m' \geq step) \cdot (n' = n\text{-}(step\text{-}m)) \end{cases}$$

- Outer loop body, Φ_{inner} [0/m] :

$$\begin{cases} [(step\text{-}1) \geq n] \cdot (n' < 0) \cdot Exit() \ \vee & (4) \\ ([(step\text{-}1) < n] \cdot (m' \geq step) \cdot (n' = n\text{-}step))^\star & (5) \end{cases}$$

Ranking function: when RF >= 0, stays in the loop, and when RF < 0, exits the loop.

Loops to Guarded ω-RE

$$(\textit{Guarded } \omega\textit{-RE}) \quad \Phi \quad ::= \quad \bot \mid \epsilon \mid \pi_s \mid [\pi_s] \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi^\omega$$

```
void main () { //AF(Exit())
  int m,n; int step=8;
  while (1) {



                         rn;


                         }}
```

Φ_{inner} [0/m]

- Inner loop: RF = {step-m-1, n}

$$\Phi_{inner} \equiv \begin{cases} [(step\text{-}m\text{-}1) \geq n] \cdot (n' < 0) \cdot Exit() \;\; \vee \\ [(step\text{-}m\text{-}1) < n] \cdot (m' \geq step) \cdot (n' = n\text{-}(step\text{-}m)) \end{cases}$$

- Outer loop body, Φ_{inner} [0/m] :

$$\begin{cases} [(step\text{-}1) \geq n] \cdot (n' < 0) \cdot Exit() \;\; \vee & (4) \\ ([(step\text{-}1) < n] \cdot (m' \geq step) \cdot (n' = n\text{-}step))^\star & (5) \end{cases}$$

- Outer loop: RF = {n-step}

$$\Phi_{outer} \equiv [step \geq 1] \cdot (n' < 0) \cdot Exit() \;\; \vee \;\; ([step < 1] \cdot (m' \geq step))^\omega$$

Since step=8, we have proved termination !

Ranking function: when RF >= 0, stays in the loop, and when RF < 0, exits the loop.

RQ 1: verifying CTL properties

	Program	Loc	CTL Property	Exp.	FUNCTION		T2		
					Res.	Time	Res.	Time	
1	AF_terminate	25	AF(Exit())	✗	?	0.021	?	0.414	???
2	toylin1 (Fig. 21)	32	EF(resp≥5)	✓	?	0.064	✗	0.294	
3	timer-simple …	26	AG((timer_1=0→AF(output_1=1)))	✓	?	1.739	✗	0.867	
4	AGAF…(Fig. 4)	16	AG((AF(t=1))∧(AF(t=0)))	✓	?	0.034	✗	0.597	
5	coolant_basi …	76	AU(init=0)(AU(init=1)(AG(init=3)))	✓	?	6.615	-	-	
6	AF_Bangalo …	22	AG((y<1)→AF(x<0))	✓	?	0.345	✗	0.249	
7	AFParity...(Fig. 2)	14	AF(y=1)	✓	?	0.012	?	0.362	
8	Nested... (Fig. 15)	20	AF(Exit())	✓	?	0.196	?	0.553	
9	acqrel.c	42	AG((A=1)→AF(R=0))	✓	✓	0.040	✗	0.786	
10	test_existent…	23	EF(r=1)	✓	?	0.022	✗	0.283	
11	test_global.c	14	AF(AG(y>0))	✓	?	0.219	✓	0.694	
12	test_until.c	13	AU(x>y)(x≤y)	✗	✓	0.033	-	-	
13	next.c	7	AX(AX(x=0))	✗	?	0.005	TO	-	
14	multiChoice.c	39	AF((x=4)∨(x=-4))	✓	✓	0.077	✓	0.409	
15	multiChoice.c	39	EF(x=4) ∧ EF(x=-4)	✓	?	0.086	✓	0.296	
	Total	408			13.3%	9.509	20%	5.804	

RQ 1: verifying CTL properties

	Program	Loc	CTL Property	Exp.	FUNCTION		T2		CTLEXPERT	
					Res.	Time	Res.	Time	Res.	Time
1	AF_terminate	25	AF(Exit())	✗	?	0.021	?	0.414	✗	0.31
2	toylin1 (Fig. 21)	32	EF(resp≥5)	✓	?	0.064	✗	0.294	✗	0.456
3	timer-simple …	26	AG((timer_1=0⟶AF(output_1=1)))	✓	?	1.739	✗	0.867	✓	0.406
4	AGAF…(Fig. 4)	16	AG((AF(t=1))∧(AF(t=0)))	✓	?	0.034	✗	0.597	✓	0.135
5	coolant_basi …	76	AU(init=0)(AU(init=1)(AG(init=3)))	✓	?	6.615	-	-	✗	0.678
6	AF_Bangalo …	22	AG((y<1)⟶AF(x<0))	✓	?	0.345	✗	0.249	✓	0.228
7	AFParity...(Fig. 2)	14	AF(y=1)	✓	?	0.012	?	0.362	✓	0.248
8	Nested... (Fig. 15)	20	AF(Exit())	✓	?	0.196	?	0.553	✓	0.665
9	acqrel.c	42	AG((A=1)⟶AF(R=0))	✓	✓	0.040	✗	0.786	✓	0.6
10	test_existent…	23	EF(r=1)	✓	?	0.022	✗	0.283	✓	0.277
11	test_global.c	14	AF(AG(y>0))	✓	?	0.219	✓	0.694	✓	0.367
12	test_until.c	13	AU(x>y)(x≤y)	✗	✓	0.033	-	-	✗	0.185
13	next.c	7	AX(AX(x=0))	✗	?	0.005	TO	-	✗	0.299
14	multiChoice.c	39	AF((x=4)∨(x=-4))	✓	✓	0.077	✓	0.409	✓	1.365
15	multiChoice.c	39	EF(x=4) ∧ EF(x=-4)	✓	?	0.086	✓	0.296	✓	1.421
	Total	408			13.3%	9.509	20%	5.804	86.7%	7.64

Limitation 1: limited abilities when there are nondeterministic choices for the branching.

RQ 2: Finding real code CTL bugs

	Program	Loc	ULTIMATE		T2		CTLEXPERT	
			Res.	Time	Res.	Time	Res.	Time
16 ✗	libvncserver(c311535)	25	✗	2.845	?	0.747	✗	0.855
16 ✓		27	✓	3.743	✓	0.403	✓	0.476
17 ✗	Ffmpeg(a6cba06)	40	✗	15.254	?	1.223	✗	0.606
17 ✓		44	✓	40.176	?	0.96	✓	0.397
18 ✗	cmus(d5396e4)	87	✗	6.904	?	2.717	✗	0.579
18 ✓		86	✓	33.572	?	4.826	✓	0.986
19 ✗	e2fsprogs(caa6003)	58	✗	5.952	?	2.518	✗	0.923
19 ✓		63	✓	4.533	?	16.441	✓	0.842
20 ✗	csound-android(7a611ab)	43	✗	3.654	-	-	✗	0.782
20 ✓		45	TO	-	-	-	✓	0.648
21 ✗	fontconfig(fa741cd)	25	✗	3.856	?	0.499	✗	0.769
21 ✓		25	Exception	-	?	0.51	✓	0.651
22 ✗	asterisk(3322180)	22	?	12.687	?	0.512	✗	0.196
22 ✓		25	?	11.325	?	0.563	✗	0.34
23 ✗	dpdk(cd64eeac)	45	✗	3.712	?	0.657	✗	0.447
23 ✓		45	✓	2.97	?	0.693	✗	0.481
24 ✗	xorg-server(930b9a06)	19	✗	3.111	?	0.551	✗	0.581
24 ✓		20	✓	3.101	?	0.57	✓	0.409
25 ✗	pure-ftpd(37ad222) (Fig. 5)	42	✓	2.555	?	0.452	✗	0.933
25 ✓		49	?	2.286	?	0.385	✓	0.383
	Total	786	70%	152.316	5%	34.842	90%	11.901

- Benchmark: Shi et al. [FSE 22]
- Extracted main segments of the bugs into smaller programs (~100 Loc)
- Maintained features, data structures, pointer arithmetic, etc.

RQ 2: Finding real code CTL bugs

	Program	Loc	Ultimate Res.	Ultimate Time	T2 Res.	T2 Time	CTLexpert Res.	CTLexpert Time
16 ✗	libvncserver(c311535)	25	✗	2.845	?	0.747	✗	0.855
16 ✓		27	✓	3.743	✓	0.403	✓	0.476
17 ✗	Ffmpeg(a6cba06)	40	✗	15.254	?	1.223	✗	0.606
17 ✓		44	✓	40.176	?	0.96	✓	0.397
18 ✗	cmus(d5396e4)	87	✗	6.904	?	2.717	✗	0.579
18 ✓		86	✓	33.572	?	4.826	✓	0.986
19 ✗	e2fsprogs(caa6003)	58	✗	5.952	?	2.518	✗	0.923
19 ✓		63	✓	4.533	?	16.441	✓	0.842
20 ✗	csound-android(7a611ab)	43	✗	3.654	-	-	✗	0.782
20 ✓		45	TO	-	-	-	✓	0.648
21 ✗	fontconfig(fa741cd)	25	✗	3.856	?	0.499	✗	0.769
21 ✓		25	Exception	-	?	0.51	✓	0.651
22 ✗	asterisk(3322180)	22	?	12.687	?	0.512	✗	0.196
22 ✓		25	?	11.325	?	0.563	✗	0.34
23 ✗	dpdk(cd64eeac)	45	✗	3.712	?	0.657	✗	0.447
23 ✓		45	✓	2.97	?	0.693	✗	0.481
24 ✗	xorg-server(930b9a06)	19	✗	3.111	?	0.551	✗	0.581
24 ✓		20	✓	3.101	?	0.57	✓	0.409
25 ✗	pure-ftpd(37ad222) (Fig. 5)	42	✓	2.555	?	0.452	✗	0.933
25 ✓		49	?	2.286	?	0.385	✓	0.383
	Total	786	70%	152.316	5%	34.842	90%	11.901

- Benchmark:
  Shi et al. [FSE 22]
- Extracted main segments of the bugs into smaller programs (~100 Loc)
- Maintained features, data structures, pointer arithmetic, etc.
- **Limitation 2**: semantically decreasing return values, e.g., the "read" function.

RQ 3: Repairing CTL bugs

Table 3. Experimental results for repairing CTL bugs. Column"**Symbols**" presents the numbers of symbolic constants + symbolic signs, while "**Facts**" presents the number of facts allowed to be removed + added. Apart from the total repair time, we record the the time spent by the ASP solver, in the column "**ASP Time**".

	Program	Loc(Datalog)	Configuration		Fixed	ASP Time	Total Time
			Symbols	Facts			
1	AF_terminate	101	0+7	2+0	✓	0.053	1.019
12	test_until.c	72	0+3	1+0	✓	0.023	0.498
13	next.c	67	0+4	1+0	✓	0.023	0.472
16	libvncserver	97	0+6	1+0	✓	0.049	1.081
17	Ffmpeg	182	0+12	1+0	✓	0.11	1.989
18	cmus	160	0+12	1+0	✓	0.098	2.052
19	e2fsprogs	144	0+8	1+0	✓	0.075	1.515
20	csound-android	142	0+8	1+0	✓	0.076	1.613
21	fontconfig	146	0+11	1+0	✓	0.098	2.507
23	dpdk	175	0+12	1+0	✓	0.091	2.006
24	xorg-server	78	0+2	1+0	✓	0.026	0.605
25	pure-ftpd	216	3+18	2+1	✓	3.992	11.248
	Total	1580				4.714	26.605

Limitation 3: to preserve the completeness, we haven't deployed much of the space pruning techniques.

Summary

Thank you for your attention!

- Showing the feasibility of finding/repairing real-world bugs using CTL specs.
- Analysing/repairing both safety and liveness properties.
- Allow input ranking functions via annotations or ranking function synthesis tools, which can help the analyser perform better when needed.

Future Work

1) Large scale termination/non-terminating prover
2) Liveness checking for protocols: Termination + Safety checking + Fairness Assumption.

References

[ICFEM 2020] Yahui Song and Wei-Ngan Chin. Automated temporal verification of integrated dependent effects. In Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony, editors, Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, Singapore, Singapore, March 1-3, 2021, Proceedings, volume 12531 of Lecture Notes in Computer Science, pages 73–90. Springer, 2020. doi: 10.1007/978-3-030-63406-3_5. URL https://doi.org/10.1007/ 978-3-030-63406-3_5.

[VMCAI 2021] Yahui Song and Wei-Ngan Chin. A synchronous effects logic for temporal verification of pure esterel. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, Copenhagen, Denmark, January 17-19, 2021, Proceedings, volume 12597 of Lecture Notes in Computer Science, pages 417–440. Springer, 2021. doi: 10.1007/978-3-030-67067-2_19. URL https://doi.org/10.1007/ 978-3-030-67067-2_19.

[APLAS 2022] Yahui Song, Darius Foo, and Wei-Ngan Chin. Automated temporal verification for algebraic effects. In Ilya Sergey, editor, Programming Languages and Systems - 20th Asian Symposium, Auckland, New Zealand, December 5, 2022, Proceedings, volume 13658 of Lecture Notes in Computer Science, pages 88–109. Springer, 2022. doi: 10.1007/978-3-031-21037-2_5. URL https://doi.org/10.1007/ 978-3-031-21037-2_5.

References

[TACAS 2023] Yahui Song and Wei-Ngan Chin. Automated verification for real-time systems - via implicit clocks and an extended antimirov algorithm. In Sriram Sankaranarayanan and Natasha Sharygina, editors, Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I, volume 13993 of Lecture Notes in Computer Science, pages 569–587. Springer, 2023. doi: 10.1007/978-3-031-30823-9_29. URL https://doi.org/10.1007/978-3-031-30823-9_29.

[FSE 2024] Yahui Song, Xiang Gao, Wenhua Li, Wei-Ngan Chin, and Abhik Roychoudhury. Provenfix: Temporal property-guided program repair. Proceedings of the ACM on Software Engineering, 1(FSE):226–248, 2024b.

[FM 2024] Darius Foo, Yahui Song, and Wei-Ngan Chin. Staged specifications for automated verification of higher-order imperative programs. CoRR, abs/2308.00988, 2023. doi: 10.48550/ARXIV.2308.00988. URL https://doi.org/10.48550/arXiv.2308. 00988.

[ICFP 2024] Yahui Song, Darius Foo, and Wei-Ngan Chin. Specification and verification for unrestricted algebraic effects and handling. Proc. ACM Program. Lang., 8(ICFP), aug 2024a. doi: 10.1145/3674656. URL https://doi.org/10.1145/3674656.