ProveNFix: Temporal Property guided Program Repair

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

17th July @ FSE 2024, Porto de Galinhas, Brazil
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.”

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>`

```c
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>`

```c
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()`. **Future-condition**

On failure, returns a null pointer.

```c
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

Entailment Checking

A collection of specifications

\[ \Phi \sqsubseteq [y^*/x^*]\Phi_{pre} \]

\[ \Phi'_{post} = [r/ret, y^*/x^*]\Phi_{post} \]

\[ \mathcal{E} \vdash \{ \Phi \cdot \Phi'_{post} \} e \{ \Phi_e \} \]

\[ \mathcal{E} \vdash \{ \Phi \} r = nm(y^*); e \{ \Phi'_{post} \cdot \Phi_e \} \]

[FR-Call]
Future-condition based modular analysis

Entailment Checking

\[ \Phi \subseteq [y^*/x^*] \Phi_{pre} \quad \Phi_{post}' = [r/ret, y^*/x^*] \Phi_{post} \]

\[ \mathcal{E} \vdash \{ \Phi \cdot \Phi_{post}' \} e \{ \Phi_e \} \]

\[ \mathcal{E} \vdash \{ \Phi \} r = nm(y^*); e \{ \Phi_{post}' \cdot \Phi_e \} \]

[FR-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

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

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

```c
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

```c
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

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

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

```c
int* wrap_malloc_IV ()
// future: true ∧ _*
{ int* ptr = malloc (4);
  + if (ptr != NULL) free(ptr); // a repair
  return NULL;}
```

Failed entailment: true ∧ Φ ⊏ ptr≠null ∧ Ψ (free(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.

```
(IntRE) \Phi ::= \sqrt(\pi \land \theta)
(Traces) \theta ::= \bot \mid \epsilon \mid I \mid \theta_1 \cdot \theta_2 \mid \theta_1 \lor \theta_2 \mid \theta^*
(Events) I ::= A(v) \mid A(\_\_) \mid \neg A(v) \mid \neg(\_\_) \mid \_ \mid I_1 \land I_2
(Pure) \pi ::= T \mid F \mid bop(t_1, t_2) \mid \pi_1 \land \pi_2 \mid \pi_1 \lor \pi_2 \mid \neg \pi \mid \exists x. \pi
(Terms) t ::= v \mid t_1 + t_2 \mid t_1 - t_2
(Values) v ::= c \mid x \mid null
```

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:

\[ x > 2 \land E \subseteq x > 1 \land (E \lor F) \]

\[ x > 0 \land E \not\subseteq x > 1 \land (E \lor F) \]

\[ \text{true} \land E \not\subseteq \text{true} \land (E \cdot F) \]

\[ (a \lor b) \ast \subseteq (a \lor b \lor bb) \ast \quad \text{[Reoccur]} \]

\[ \epsilon \cdot (a \lor b) \ast \subseteq \epsilon \cdot (a \lor b \lor bb) \ast \quad \text{[Reoccur]} \]

\[ a \cdot (a \lor b) \ast \subseteq (a \lor b \lor bb) \ast \]

\[ b \cdot (a \lor b) \ast \subseteq \ldots \]

\[ (a \lor b) \ast \subseteq (a \lor b \lor bb) \ast \]
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”
Can temporal property analysis be modular? Can!

“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”
Experiment 1: detecting bugs

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

<table>
<thead>
<tr>
<th>Primitive APIs</th>
<th>Pre</th>
<th>Post</th>
<th>Future</th>
<th>Targeted Bug Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open/socket/fopen/fdopen/opendir</code></td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>Resource Leak</td>
</tr>
<tr>
<td><code>close/fclose/endmntent/fflush/closedir</code></td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td><code>malloc/realloc/calloc/localtime</code></td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>Null Pointer Dereference</td>
</tr>
<tr>
<td><code>malloc/realloc/localtime</code></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Memory Usage (Leak, Use-After-Free, Double Free)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>kLoC</th>
<th>#NPD</th>
<th>#ML</th>
<th>#RL</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swoole(a4256e4)</td>
<td>44.5</td>
<td>30+7</td>
<td>16+4</td>
<td>13+1</td>
<td>2m 50s</td>
</tr>
<tr>
<td>lxc(72cc48f)</td>
<td>63.3</td>
<td>7+9</td>
<td>11+6</td>
<td>5+1</td>
<td>55.62s</td>
</tr>
<tr>
<td>WavPack(22977b2)</td>
<td>36</td>
<td>23+7</td>
<td>3+9</td>
<td>0+2</td>
<td>27.99s</td>
</tr>
<tr>
<td>flex(d3de49f)</td>
<td>23.9</td>
<td>14+4</td>
<td>3</td>
<td>0</td>
<td>32.25s</td>
</tr>
<tr>
<td>p11-kit</td>
<td>76.2</td>
<td>3+5</td>
<td>13+3</td>
<td>5</td>
<td>1m 57s</td>
</tr>
<tr>
<td>x264(d4099dd)</td>
<td>67.7</td>
<td>0</td>
<td>12</td>
<td>2</td>
<td>2m 33s</td>
</tr>
<tr>
<td>re-cutils-1.8</td>
<td>81.9</td>
<td>25</td>
<td>13+10</td>
<td>1</td>
<td>9m 10s</td>
</tr>
<tr>
<td>inetutils-1.9.4</td>
<td>117.2</td>
<td>7+4</td>
<td>9+3</td>
<td>1</td>
<td>30.26s</td>
</tr>
<tr>
<td>snort-2.9.13</td>
<td>378.2</td>
<td>44+12</td>
<td>26+4</td>
<td>1+2</td>
<td>8m 49s</td>
</tr>
<tr>
<td>grub(c6b9a0a)</td>
<td>331.1</td>
<td>13+12</td>
<td>1</td>
<td>0+3</td>
<td>3m 27s</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1,220.00</td>
<td>166+60</td>
<td>107+30</td>
<td>26+9</td>
<td>31m 12s</td>
</tr>
</tbody>
</table>
Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

Require: \( \mathcal{E}, (\pi \land \theta_{\text{target}}) \)
Ensure: An expression \( e_R \) such that \( \mathcal{E} \vdash \{T \land \epsilon\} e_R \{\pi \land \theta_{\text{target}}\} \)

1: \( e_{\text{acc}} = () \)
2: for each \( nm(x^*) \mapsto [\Phi_{\text{pre}}, \Phi_{\text{post}}, \Phi_{\text{future}}] \in \mathcal{E} \) do
3: if \( \theta_{\text{target}} = \epsilon \) then return if \( \pi \) then \( e_{\text{acc}} \) else ()
4: else
5: // there exist a set of program variables \( y^* \)
6: \( \theta'_{\text{target}} = (\pi \land [y^*/x^*] \Phi_{\text{post}})^{-1} \theta_{\text{target}} \)
7: \( e_{\text{acc}} = e_{\text{acc}} \cup nm(y^*) \)
8: end if
9: end for
10: return without any suitable patches

Example: \( \text{true} \land \mathcal{E} \not\subseteq \text{ptr} \neq \text{null} \land _{\land^*}.(\text{free(ptr)}) \)

\[ \Rightarrow \text{synthesis(} \text{ptr} \neq \text{null} \land _{_{\land^*}.}(\text{free(ptr)})\text{)} \Rightarrow \text{if (ptr} \neq \text{NULL) free(ptr);} \]
Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

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

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

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

$\Rightarrow$ $\text{synthesis} (\text{ptr} \neq \text{null} \land \_\_\_\^\*. (\text{free} (\text{ptr}))) \Rightarrow \text{if (ptr !\= NULL) free(ptr);}$
Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

Require: $E, (\pi \land \theta_{target})$
Ensure: An expression $e_{R}$ such that $E \vdash \{T \land \epsilon\} e_{R} \{\pi \land \theta_{target}\}$

1: $e_{acc} = ()$
2: for each $nm(x^*) \leftrightarrow [\Phi_{pre}, \Phi_{post}, \Phi_{future}] \in E$ do
3: if $\theta_{target} = \epsilon$ then return if $\pi$ then $e_{acc}$ else ()
4: else
5: // there exist a set of program variables $y^*$
6: $\theta'_{target} = (\pi \land [y^*/x^*]\Phi_{post})^{-1}\theta_{target}$
7: $e_{acc} = e_{acc}; nm(y^*)$
8: end if
9: end for
10: return without any suitable patches

Example: $true \land E \nexists ptr\neq null \land _\wedge^*.(free(ptr))$

$\Rightarrow$ synthesis$(ptr\neq null \land _\wedge^*.(free(ptr))) \Rightarrow$ if (ptr != NULL) free(ptr);
Automated repair via deductive synthesis

Algorithm 1 Algorithm for the Deductive Synthesis

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

1: $e_{\text{acc}} = ()$
2: for each $nm(x^*) \leftrightarrow [\Phi_{\text{pre}}, \Phi_{\text{post}}, \Phi_{\text{future}}] \in \mathcal{E}$ do
3: if $\theta_{\text{target}} = \epsilon$ then return if $\pi$ then $e_{\text{acc}}$ else ()
4: else
5: // there exist a set of program variables $y^*$
6: $\theta'_{\text{target}} = (\pi \land [y^*/x^*]\Phi_{\text{post}})^{-1}\theta_{\text{target}}$
7: $e_{\text{acc}} = e_{\text{acc}} \cup nm(y^*)$
8: end if
9: end for
10: return without any suitable patches

Example: $\text{true} \land \mathcal{E} \not\subseteq \text{ptr}\neq\text{null} \land _{\text{^*}}^\text{.}(\text{free(ptr)})$

$\Rightarrow \text{synthesis}(\text{ptr}\neq\text{null} \land _{\text{^*}}^\text{.}(\text{free(ptr)})) \Rightarrow \text{if (ptr != NULL) free(ptr);}$
## Experiment 2: Repairing bugs

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

- 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))

<table>
<thead>
<tr>
<th>OpenSSL Applications</th>
<th>kLoC</th>
<th>Issue ID</th>
<th>Target API</th>
<th>Github Status</th>
<th>ProveNFix</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>keepalive(843ff80)</td>
<td>59.1</td>
<td>1003</td>
<td>SSL_CTX_new</td>
<td>✓</td>
<td>✓</td>
<td>5.62s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1004</td>
<td>SSL_new</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thc-ipv6(011376c)</td>
<td>30.9</td>
<td>28</td>
<td>BN_new</td>
<td>✓</td>
<td>✓</td>
<td>3.32s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>BN_set_word</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeRADIUS(94149dc)</td>
<td>258.9</td>
<td>2309</td>
<td>BIO_new</td>
<td>✓</td>
<td>✓</td>
<td>38.89s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2310</td>
<td>i2a_ASN1_OBJECT</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trafficserver(5ee6a5f)</td>
<td>34.1</td>
<td>4292</td>
<td>SSL_CTX_new</td>
<td>✓</td>
<td>✓</td>
<td>21.55s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4293</td>
<td>SSL_new</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4294</td>
<td>SSL_write</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>sslsplit(19a16bd)</td>
<td>18.7</td>
<td>224</td>
<td>SSL_CTX_use_certificate</td>
<td>✓</td>
<td>✓</td>
<td>2.69s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>225</td>
<td>SSL_use_PrivateKey</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>proxytunnel(f7831a2)</td>
<td>3.1</td>
<td>36</td>
<td>SSL_connect</td>
<td>✓</td>
<td>✓</td>
<td>0.62s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37</td>
<td>SSL_new</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Compositional static analyzer via temporal properties.
- Specified 17 APIs; found 515 vulnerabilities from 1 million LOC; with a 90% fix rate.
- Specification inference via bi-abduction.
- The inferred spec can be used to analysis protocol applications, e.g., OpenSSL.