

Semantic Program Repair Using a Reference Implementation

Sergey Mechtaev*

National University of Singapore
mechtaev@comp.nus.edu.sg

Manh-Dung Nguyen*

National University of Singapore
nguyenmanhdung1710@gmail.com

Yannic Noller

Humboldt University of Berlin
yannic.noller@informatik.hu-berlin.de

Lars Grunske

Humboldt University of Berlin
grunske@informatik.hu-berlin.de

Abhik Roychoudhury

National University of Singapore
abhik@comp.nus.edu.sg

International Conference on Software Engineering, Gothenburg, Sweden, May 27-June 3, 2018.





Introdução

Visão Geral

Implementação

Experimento

Resultado



- Correção de bugs realizada de forma manual
 - Tempo
 - Recurso
- Reparo automatizado
 - Redução dos esforços manuais
 - Correções de defeitos em software
 - Fornece patches de correção



Problema

- Baixa qualidade dos patches
- Não garantem a generalização
- Dificuldade de gerar uma especificação exata do comportamento pretendido
- Essa questão se torna uma barreira para a adoção dessa tecnologia pelos desenvolvedores.



- Técnicas para melhorar a qualidade dos patches gerados automaticamente
 - Priorização de patch
 - Anti-padrões
 - Geração de testes




Solução Proposta

- Gerar patches baseado em uma implementação de referência
 - Integre a noção de equivalência condicional
 - Uso de um algoritmo geração de patch escalável [Angelix]
- Funcionamento do sistema
 - recebe uma condição de entrada, ϕ
 - gera automaticamente um patch
 - equivalência condicional ao programa de referência.

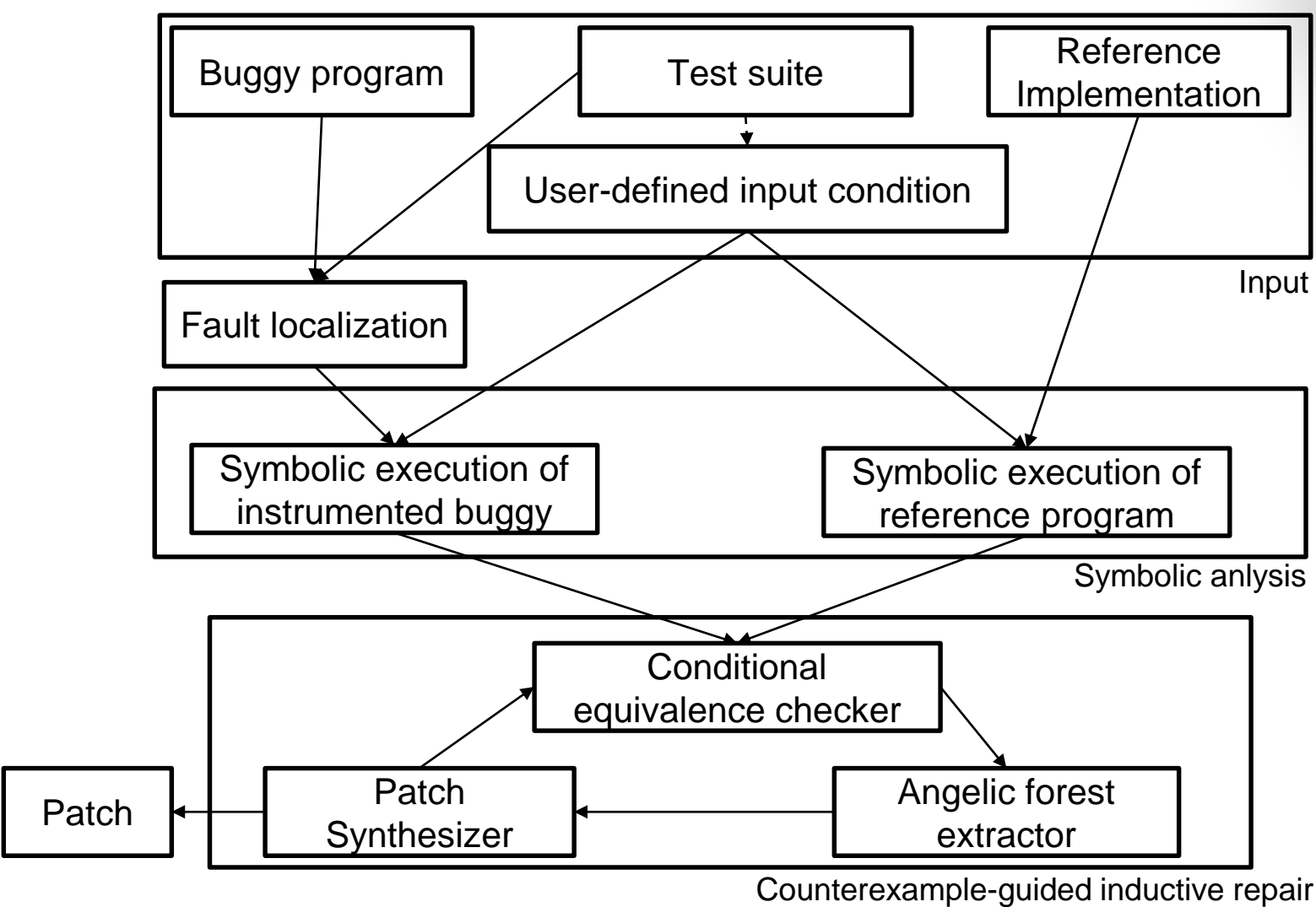


Contribuições

- Inferir uma especificação correta de uma implementação de referência e usá-la para orientar o reparo do programa, a fim de abordar o problema do overfitting de teste.
- Introduzir um algoritmo escalável de geração de patches com base em uma implementação de referência que garante a equivalência condicional dos programas patched e de referência w.r.t. uma condição de entrada definida pelo usuário.
- Avaliação sobre duas implementações de utilitários UNIX (GNU Coreutils e Busybox) que demonstram que tal metodologia aborda o problema de overfitting de teste de reparo de programas e escala para softwares do mundo real



Introdução
Visão Geral
Implementação
Experimento
Resultado





```
int search(int x, int a[], int length) {  
    int i;  
    for (i=0; i<length; i++) {  
        if (x == a[i])  
            return i;  
    }  
    return -1;  
}
```

(a) Reference program

```
int search(int x, int a[], int length) {  
    int L = 0;  
    int R = length - 1;  
    do {  
        int m = (L+R)/2;  
        if (x == a[m]) {  
            return m;  
        } else if (x < a[m]) { // bug fix: x > a[m]  
            L = m+1;  
        } else {  
            R = m-1;  
        }  
    } while (L <= R);  
    return -1;  
}
```

(c) Buggy program



Negative input	$x \mapsto 2$ $a \mapsto [2, 4, 6]$ $length \mapsto 3$
Expected output	0
Symbolic inputs	$x \mapsto \gamma$ $a \mapsto [\alpha_0, \alpha_1, \alpha_2]$ $length \mapsto \delta$
Input condition	$\phi := \alpha_0 < \alpha_1 < \alpha_2 \wedge \delta = 3$

(e) Test and input condition

```

int search(int x, int a[], int length) {
    int i;
    for (i=0; i<length; i++) {
        if (x == a[i])
            return i;
    }
    return -1;
}

```

(a) Reference program

ID	π^r	θ_{out}^r
r_1	$\gamma = \alpha_0$	0
r_2	$\gamma \neq \alpha_0 \wedge \gamma = \alpha_1$	1
r_3	$\gamma \neq \alpha_0 \wedge \gamma \neq \alpha_1 \wedge \gamma = \alpha_2$	2
r_4	$\gamma \neq \alpha_0 \wedge \gamma \neq \alpha_1 \wedge \gamma \neq \alpha_2$	-1

(b) Summary of reference program

```

int search(int x, int a[], int length) {
    int L = 0;
    int R = length - 1;
    do {
        int m = (L+R)/2;
        if (x == a[m]) {
            return m;
        } else if (x < a[m]) { // bug fix: x > a[m]
            L = m+1;
        } else {
            R = m-1;
        }
    } while (L <= R);
    return -1;
}

```

ID	π^b	θ_c	θ_{out}^b
b_1	$\gamma = \alpha_1$	-	1
b_2	$\gamma \neq \alpha_1 \wedge \beta^0 \wedge \gamma = \alpha_2$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$	2
b_3	$\gamma \neq \alpha_1 \wedge \beta^0 \wedge \gamma \neq \alpha_2 \wedge \beta^1$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$ $\beta^1 : \{x \mapsto \gamma, a[m] \mapsto \alpha_2\}$	-1
b_4	$\gamma \neq \alpha_1 \wedge \beta^0 \wedge \gamma \neq \alpha_2 \wedge \neg \beta^1$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$ $\beta^1 : \{x \mapsto \gamma, a[m] \mapsto \alpha_2\}$	-1
b_5	$\gamma \neq \alpha_1 \wedge \neg \beta^0 \wedge \gamma = \alpha_0$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$	0
b_6	$\gamma \neq \alpha_1 \wedge \neg \beta^0 \wedge \gamma \neq \alpha_0 \wedge \beta^1$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$ $\beta^1 : \{x \mapsto \gamma, a[m] \mapsto \alpha_0\}$	-1
b_7	$\gamma \neq \alpha_1 \wedge \neg \beta^0 \wedge \gamma \neq \alpha_0 \wedge \neg \beta^1$	$\beta^0 : \{x \mapsto \gamma, a[m] \mapsto \alpha_1\}$ $\beta^1 : \{x \mapsto \gamma, a[m] \mapsto \alpha_0\}$	-1

(d) Specification of buggy program





$$VC = \forall \alpha_0 \forall \alpha_1 \forall \alpha_2 \forall \gamma \bigwedge_{(\pi^r, \theta_{out}^r)} \bigwedge_{(\pi^b, \theta_{out}^b)} \pi^r \wedge \pi^b \wedge (\beta = e[\![\theta_c]\!]) \Rightarrow \theta_{out}^r = \theta_{out}^b$$

$$\equiv \forall \alpha_0 \forall \alpha_1 \forall \alpha_2 \forall \gamma ($$

$$(r_1, b_3) \quad \neg(\gamma = \alpha_0 \wedge \gamma \neq \alpha_1 \wedge \beta^0 \wedge \beta^0 = \gamma < \alpha_1 \wedge \gamma \neq \alpha_2 \wedge \beta^1 \wedge \beta^1 = \gamma < \alpha_2)$$

$$(r_1, b_4) \quad \wedge \neg(\gamma = \alpha_0 \wedge \gamma \neq \alpha_1 \wedge \beta^0 \wedge \beta^0 = \gamma < \alpha_1 \wedge \gamma \neq \alpha_2 \wedge \neg \beta^1 \wedge \beta^1 = \gamma < \alpha_2)$$

$$(r_3, b_6) \quad \wedge \neg(\gamma = \alpha_2 \wedge \gamma \neq \alpha_1 \wedge \neg \beta^0 \wedge \beta^0 = \gamma < \alpha_1 \wedge \gamma \neq \alpha_0 \wedge \beta^1 \wedge \beta^1 = \gamma < \alpha_0)$$

$$(r_3, b_7) \quad \wedge \neg(\gamma = \alpha_2 \wedge \gamma \neq \alpha_1 \wedge \neg \beta^0 \wedge \beta^0 = \gamma < \alpha_1 \wedge \gamma \neq \alpha_0 \wedge \neg \beta^1 \wedge \beta^1 = \gamma < \alpha_0))$$

(f) Verification condition

- Se ambas as execuções no programa de referência e de bugs seguem o mesmo caminho, suas saídas devem ser as mesmas
- A verification condition também codificará os valores das variáveis visíveis na expressão ($x < a[m]$) computada no contexto simbólico θ_c



- A fim de verificar a validade da verification condition (VC), foi verificado a insatisfatibilidade de sua negação.
- A VC negado será resolvido com um solver SMT.
- Após negar o VC, o solver SMT gera uma entrada de contraexemplo $\{(\beta^0, c, \sigma)\}$, given that $c := False, \sigma := \{x \mapsto -1, a[m] \mapsto 0\}$
- Dada essa entrada, o sintetizador retorna um patch plausível ($x == a[m]$).



- Depois de inserir essa expressão no VC e negá-la, o solucionador SMT gera uma segunda entrada de contraexemplo

$$\{x \rightarrow 1, a \rightarrow [-1, 0, 1]\}.$$


$$\{(\beta^0, False, \{x \mapsto -1, a[m] \mapsto 0\}),$$

$$(\beta^0, True, \{x \mapsto 1, a[m] \mapsto 0\}) \}.$$

- Dada esta entrada, o sintetizador retorna o patch $(x \geq a[m])$.
- Depois de inserir esta expressão no VC e negá-lo, o SMT retorna insatisfatório, ou seja, o patch sintetizado atende a todos os requisitos.



- Para verificar a eficiência dessa nova abordagem foi realizado uma comparação com uma técnica de reparo guiada por teste (Angelix), utilizando o mesmo exemplo.
- Para esse problema em questão o Angelix produziu apenas patch plausível ($a [m] < a [m]$). Sendo necessário incluir mais casos de teste para gerar um patch correto



Introdução
Visão Geral
Implementação
Exeperimento
Resultados

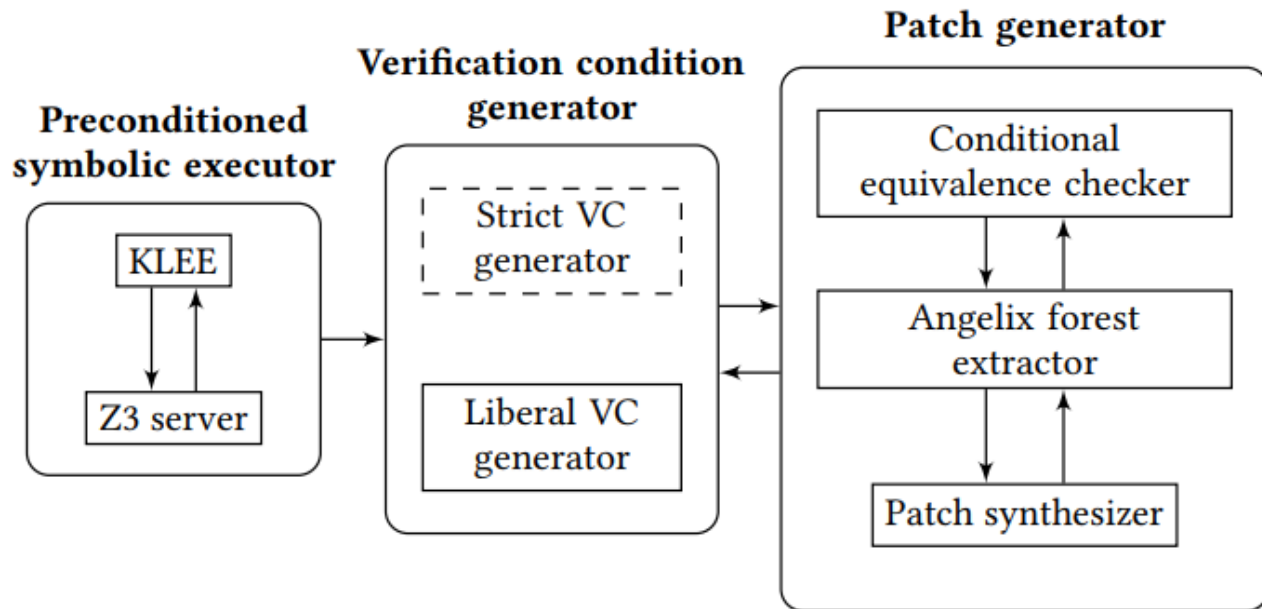



Figure 4: Architecture of SEMGRAFT.



Introdução
Visão Geral
Implementação
Experimento
Resultados



- Questões de pesquisa
 - (1) O SemGraft pode gerar reparos para softwares do mundo real?
 - (2) O SemGraft pode aliviar o problema de overftting de técnicas existentes de reparo de programas baseados em implementação de referência?




- Para abordar as questões de pesquisa descritas, foram adotados quatro critérios a seguir.
 - (1) Os assuntos são softwares do mundo real que são amplamente utilizados.
 - (2) Existem programas de referência que processam as mesmas entradas que os programas com bugs, mas exibem o comportamento correto.
 - (3) O buggy e o programa de referência são substancialmente diferentes em sua estrutura.
 - (4) As correções do desenvolvedor estão dentro dos espaços de busca de implementação.



- Foram utilizados 12 erros reais de software de dois projetos C de código aberto Busybox e GNU Coreutils
- Os erros foram extraídos de logs de commit, relatórios de bug e pesquisas anteriores.
- Ambas implementações fornecem utilitários UNIX com mesma funcionalidade, mas o Busybox foi implementado com otimização de tamanho, recursos limitados.



- Foi empregado a ferramenta SemGraft para reparar o Busybox Linux embarcado com ferramentas GNU como o Coreutils como referência e vice-versa.
- A nível de comparação foi empregado também uma abordagem de reparo de programa orientada a testes de ponta, Angelix [26].



Introdução
Visão Geral
Implementação
Experimento
Resultados

Table 1: Busybox subject programs

Buggy Prog.	Buggy Commit	Ref. Prog.	Ref. Prog. Version	Failure Description	ANGELIX'	SEMGRAFT
sed	c35545a	sed of GNU sed	version 3.01	Failed to handle zero-length match	Correct	Correct
seq	f7d1c59	seq of Coreutils	version 6.10	Wrong output when 2 input arguments are equal	Correct	Correct
sed	7666fa1	sed of GNU sed	version 3.01	Wrong output when handling <code>s///NUM</code>	Incorrect	Correct
sort	d1ed3e6	sort of Coreutils	version 8.27	Wrong output when handling <code>-kSTART,N.ENDCHAR</code>	Incorrect	Correct
seq	d86d20b	seq of Coreutils	version 8.27	seq no longer accepts 0 value as increment argument	Incorrect	Correct
sed	3a9365e	sed of GNU sed	version 3.01	Failed to handle <code>s///</code> which has empty matches	Incorrect	Correct

Table 2: Coreutils subject programs

Buggy Prog.	Buggy Commit	Ref. Prog.	Ref. Prog. Version	Failure Description	ANGELIX'	SEMGRAFT
mkdir	f7d1c59	mkdir of Busybox	version 1.27.2	Segmentation fault	Incorrect	Correct
mkfifo	cdb1682	mkfifo of Busybox	version 1.27.2	Segmentation fault	Incorrect	Correct
mknod	cdb1682	mknod of Busybox	version 1.27.2	Segmentation fault	Incorrect	Correct
copy	f3653f0	copy of Busybox	version 1.27.2	Failed to copy a file	Correct	Correct
md5sum	739cf4e	md5sum of Busybox	version 1.27.2	Segmentation fault	Correct	Correct
cut	6f374d7	cut of Busybox	version 1.27.2	Failed to handle <code>-b 2-, 3- like -b 2-</code>	Incorrect	Correct



Conclusões

- A técnica aborda o problema de overfitting, fornecendo garantias adicionais de correção.
- Especificamente, sintetiza um patch de imposição de equivalência condicional w.r.t. uma condição de entrada definida pelo usuário.
- O método se adapta a programas do mundo real, como GNU Coreutils e Busybox, e ajuda a gerar reparos mais corretos.

Referências



- ❖ Mehtaev, S., Nguyen, M. D., Noller, Y., Grunske, L., & Roychoudhury, A. (2018). Semantic Program Repair Using a Reference Implementation. In *Proceedings of ICSE*.