State Merging with Quantifiers in Symbolic Execution

David Trabish[1]

Noam Rinetzky[1]

Sharon Shoham[1]

Vaibhav Sharma[2]

[1]Tel Aviv University, Israel

[2]University of Minnesota, USA

KLEE 2024

(ESEC/FSE 2023)

Symbolic Execution: State Merging

- Mitigates path explosion by joining exploration paths
- Often leads to:
  - Large disjunctive constraints
  - Costly constraint solving

Today's Talk

- State merging using **compact quantified** constraints
- Specialized solving procedure

$$(\ldots) \lor (\ldots) \lor \ldots\ldots\ldots \lor (\ldots)$$

$$\Downarrow$$

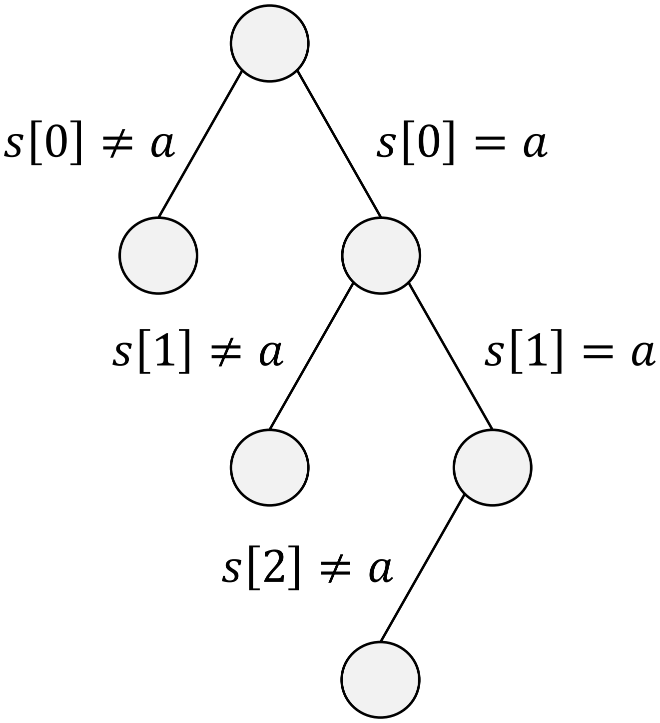
$$\forall x. (\ldots)$$

Example

```c
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, ‘a’);
int m = strspn(s + n, ‘b’);
...
```

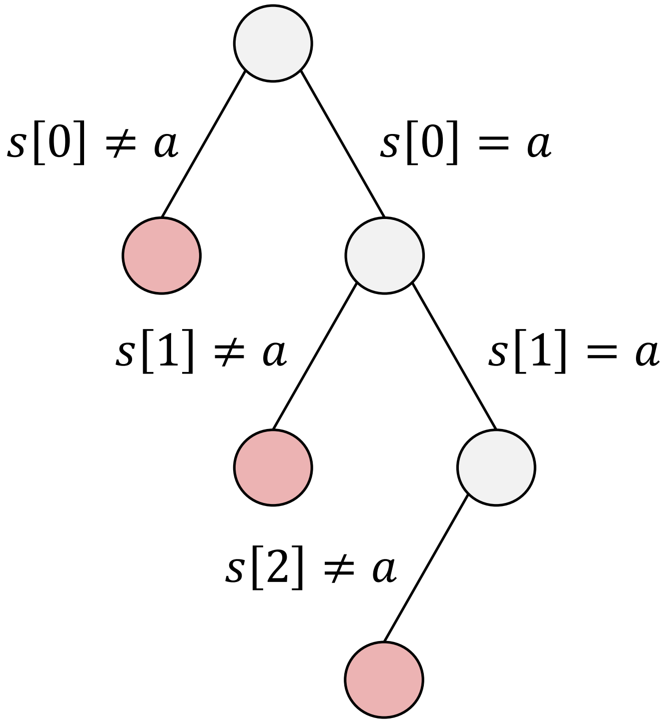
Example

```c
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, 'a');
int m = strspn(s + n, 'b');
...
```



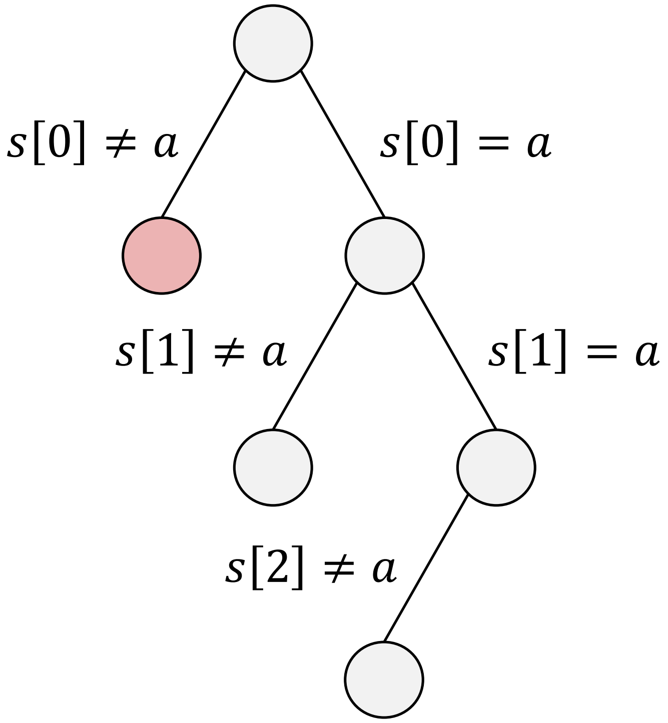
State Merging: Path Constraints

```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, 'a');
int m = strspn(s + n, 'b');
...
```



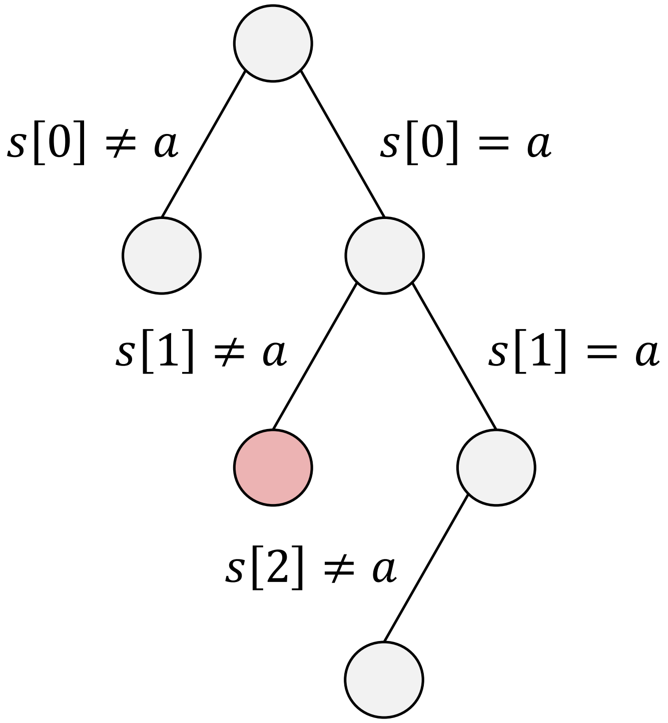
State Merging: Path Constraints

$(s[0] \neq a)$



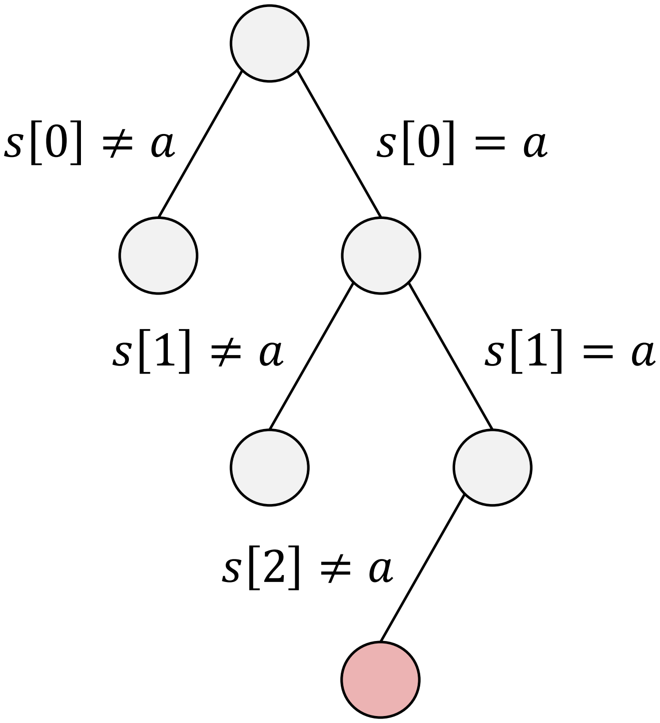
State Merging: Path Constraints

$(s[0] \neq a)$
$(s[0] = a \land s[1] \neq a)$



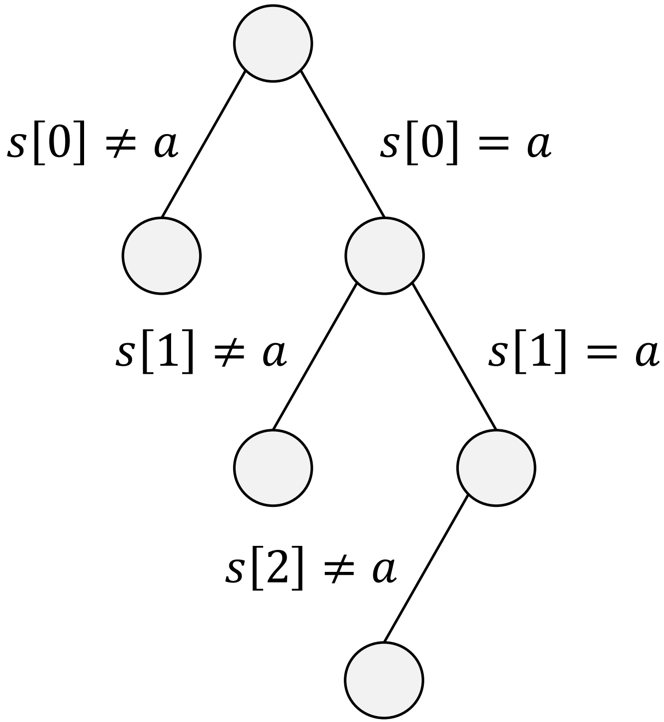
State Merging: Path Constraints

$(s[0] \neq a)$
$(s[0] = a \land s[1] \neq a)$
$(s[0] = a \land s[1] = a \land s[2] \neq a)$



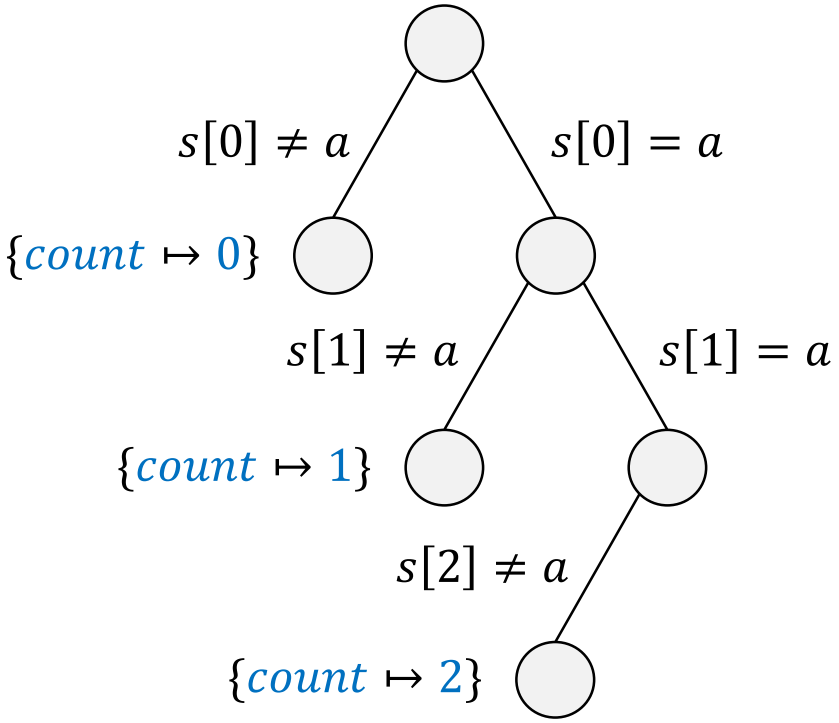
State Merging: Path Constraints

$(s[0] \neq a) \vee$
$(s[0] = a \wedge s[1] \neq a) \vee$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$



State Merging: Memory

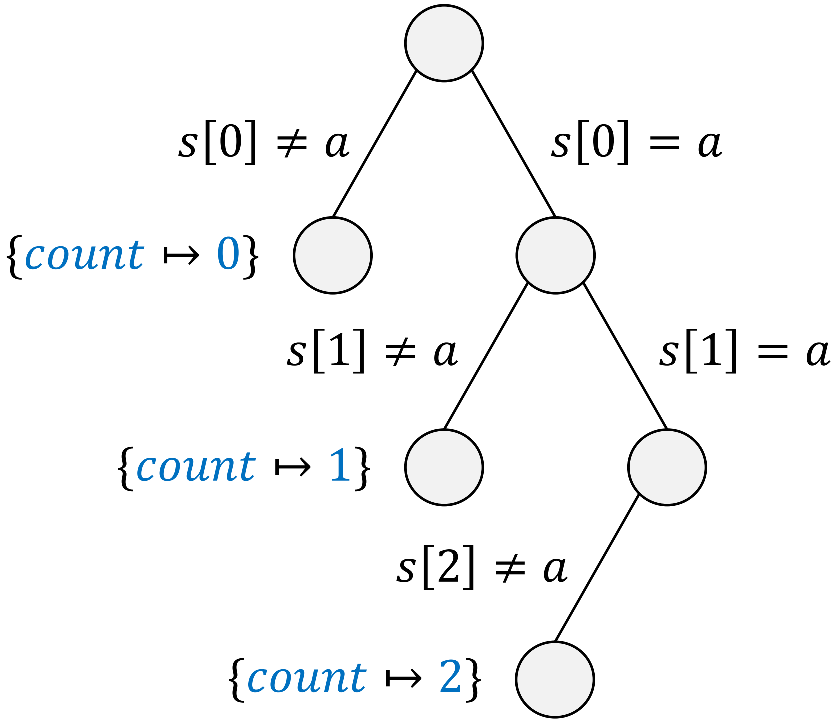
```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, 'a');
int m = strspn(s + n, 'b');
...
```



State Merging: Memory

$ite($
$\quad s[0] \neq a,$
$\quad 0,$
$\quad \ldots$
$)$

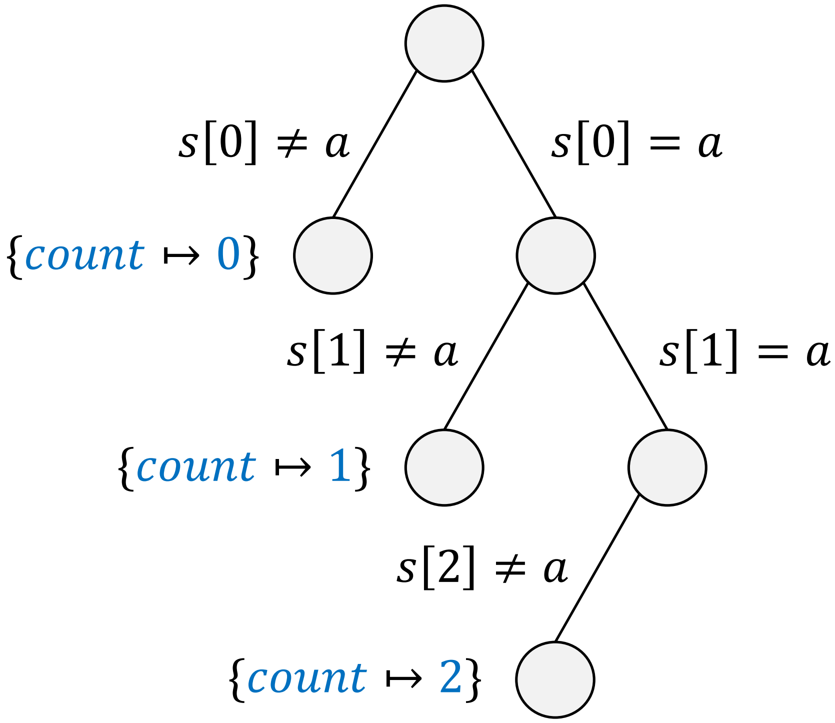
merged value of count



State Merging: Memory

$$
\begin{array}{l}
ite(\\
\quad s[0] \neq a, \\
\quad 0, \\
\quad ite(\\
\qquad s[0] = a \wedge s[1] \neq a, \\
\qquad 1, \\
\qquad \ldots \\
\quad) \\
)
\end{array}
$$

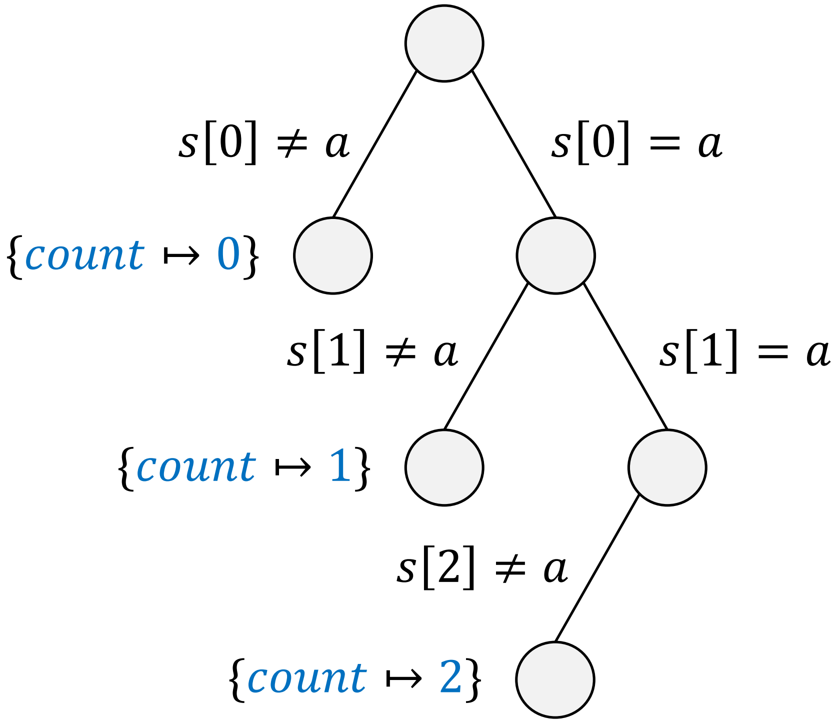
merged value of `count`



State Merging: Memory

$$
\begin{aligned}
& ite(\\
& \quad s[0] \neq a, \\
& \quad 0, \\
& \quad ite(\\
& \quad\quad s[0] = a \wedge s[1] \neq a, \\
& \quad\quad 1, \\
& \quad\quad 2 \\
& \quad) \\
&)
\end{aligned}
$$

merged value of `count`



State Merging

```c
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, ‘a’);
int m = strspn(s + n, ‘b’);
...
```

$$\left\{ ite(s[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1, 2)) \right.$$

State Merging

```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, ‘a’);
int m = strspn(s + n, ‘b’);
...
```

$$
\left\{
\begin{array}{l}
ite(\\
\quad s[0] \neq a, \\
\quad 0, \\
\quad ite(\\
\quad\quad s[0] = a \wedge s[1] \neq a, \\
\quad\quad 1, \\
\quad\quad 2 \\
\quad) \\
)
\end{array}
\right.
$$

State Merging

```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, 'a');
int m = strspn(s + n, 'b');
...
```

$$
\left\{
\begin{array}{l}
ite(\\
\quad s[0] \neq a, \\
\quad 0, \\
\quad ite(\\
\quad\quad s[0] = a \wedge s[1] \neq a, \\
\quad\quad 1, \\
\quad\quad 2 \\
\quad) \\
)
\end{array}
\right.
$$

State Merging

Path constraints

$\big((s[0] \neq a) \vee (s[0] = a \wedge s[1] \neq a) \vee (s[0] = a \wedge s[1] = a \wedge s[2] \neq a)\big) \wedge$
$($
$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] \neq a) \vee$
$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] \neq a) \vee$
$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 2] \neq a)$
$)$

Value of `m`

$ite($
$\mathrm{s}[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] \neq a,$
$0,$
$ite($
$s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] \neq a,$
$1,$
2
$)$
$)$

State Merging with Quantifiers

Merging the path constraints

```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, ‘a’);
int m = strspn(s + n, ‘b’);
...
```

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a) \vee$$
$$(s[0] = a \wedge s[1] \neq a) \vee$$
$$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$$

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a)$$
$$(s[0] = a \wedge s[1] \neq a)$$
$$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$$

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a)$$
$$(s[0] = a \land s[1] \neq a)$$
$$(s[0] = a \land s[1] = a \land s[2] \neq a)$$

$$s[0] = a \land \cdots \land s[i-1] = a \land s[i] \neq a$$

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a)$$
$$(s[0] = a \land s[1] \neq a)$$
$$(s[0] = a \land s[1] = a \land s[2] \neq a)$$

$$s[0] = a \land \cdots \land s[i-1] = a \land s[i] \neq a$$

$i = 0, 1, 2$

$$\Updownarrow$$

$$(\forall x.\, 1 \leq x \leq i \rightarrow s[x-1] = a) \land s[i] \neq a$$

bound variable

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a) \lor$$
$$(s[0] = a \land s[1] \neq a) \lor$$
$$(s[0] = a \land s[1] = a \land s[2] \neq a)$$

$$\Updownarrow$$

$$\big((\forall x. 1 \leq x \leq 0 \rightarrow s[x-1] = a) \land s[0] \neq a\big) \lor$$
$$\big((\forall x. 1 \leq x \leq 1 \rightarrow s[x-1] = a) \land s[1] \neq a\big) \lor$$
$$\big((\forall x. 1 \leq x \leq 2 \rightarrow s[x-1] = a) \land s[2] \neq a\big)$$

State Merging with Quantifiers

Merging the path constraints

$$(s[0] \neq a) \lor$$
$$(s[0] = a \land s[1] \neq a) \lor$$
$$(s[0] = a \land s[1] = a \land s[2] \neq a)$$

$$\Updownarrow$$

$$\big((\forall x. 1 \leq x \leq 0 \rightarrow s[x-1] = a) \land s[0] \neq a\big) \lor$$
$$\big((\forall x. 1 \leq x \leq 1 \rightarrow s[x-1] = a) \land s[1] \neq a\big) \lor$$
$$\big((\forall x. 1 \leq x \leq 2 \rightarrow s[x-1] = a) \land s[2] \neq a\big)$$

$$\Updownarrow$$

$$0 \leq i \leq 2 \land (\forall x. 1 \leq x \leq i \rightarrow s[x-1] = a) \land s[i] \neq a$$

fresh free variable

State Merging with Quantifiers

Merging memory

$pc \stackrel{\text{def}}{=} 0 \leq i \leq 2 \wedge (\forall x. 1 \leq x \leq i \rightarrow s[x-1] = a) \wedge s[i] \neq a$

merged value of n $\left\{ \begin{array}{l} ite(\\ \quad \mathrm{s}[0] \neq a, \\ \quad 0, \\ \quad ite(\\ \quad\quad s[0] = a \wedge s[1] \neq a, \\ \quad\quad 1, \\ \quad\quad 2 \\ \quad) \\ ) \end{array} \right.$

State Merging with Quantifiers

Merging memory

$pc \stackrel{\text{def}}{=} 0 \leq i \leq 2 \wedge (\forall x. 1 \leq x \leq i \rightarrow s[x-1] = a) \wedge s[i] \neq a$

merged value of n

$$\begin{cases} ite(\\ \quad s[0] \neq a, \\ \quad 0, \\ \quad ite(\\ \quad\quad s[0] = a \wedge s[1] \neq a, \\ \quad\quad 1, \\ \quad\quad 2 \\ \quad) \\ ) \end{cases}$$

State Merging with Quantifiers

Merging memory

$$pc \stackrel{\text{def}}{=} 0 \le i \le 2 \land (\forall x.\, 1 \le x \le i \rightarrow s[x-1] = a) \land s[i] \ne a$$

merged value of n

$$\left\{ \begin{array}{l} ite(\\ \quad s[0] \ne a, \\ \quad 0, \\ \quad ite(\\ \quad\quad s[0] = a \land s[1] \ne a, \\ \quad\quad 1, \\ \quad\quad 2 \\ \quad) \\ ) \end{array} \right. \Longrightarrow \quad i$$

State Merging with Quantifiers

Merging the path constraints

```
int strspn(char *s, char c) {
  int count = 0;
  while (s[count] == c) {
    count++;
  }
  return count;
}

// symbolic, null-terminated
char s[3];
int n = strspn(s, ‘a’);
int m = strspn(s + n, ‘b’);
...
```

State Merging with Quantifiers

Path constraints

$$0 \leq i \leq 2 \wedge (\forall x. 1 \leq x \leq i \rightarrow s[x-1] = a) \wedge s[i] \neq a \wedge$$
$$0 \leq j \leq 2 \wedge (\forall x. 1 \leq x \leq j \rightarrow s[i+x-1] = b) \wedge s[i+j] \neq b$$

Value of `m`

j

Comparison: Path Constraints

standard

$$\big((s[0] \neq a) \vee (s[0] = a \wedge s[1] \neq a) \vee (s[0] = a \wedge s[1] = a \wedge s[2] \neq a)\big) \wedge$$
$$($$
$$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] \neq a) \vee$$
$$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] \neq a) \vee$$
$$(s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] = a \wedge s[ite(\mathrm{s}[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 2] \neq a)$$
$$)$$

---

with quantifiers

$$0 \leq i \leq 2 \wedge (\forall x. 1 \leq x \leq i \rightarrow s[x-1] = a) \wedge s[i] \neq a \wedge$$
$$0 \leq j \leq 2 \wedge (\forall x. 1 \leq x \leq j \rightarrow s[i+x-1] = b) \wedge s[i+j] \neq b$$

Comparison: Memory

standard

$$
\begin{array}{l}
ite(\\
\quad s[ite(s[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] \neq a, \\
\quad 0, \\
\quad ite(\\
\quad\quad s[ite(s[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 0] = a \wedge s[ite(s[0] \neq a, 0, ite(s[0] = a \wedge s[1] \neq a, 1,2)) + 1] \neq a, \\
\quad\quad 1, \\
\quad\quad 2 \\
\quad) \\
)
\end{array}
$$

---

with quantifiers

j

Questions

- How to **automatically synthesize** the quantified constraints?
- How to **efficiently solve** the quantified constraints?
- Does it improve the **performance** of symbolic execution?

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a)$
$(s[0] = a \wedge s[1] \neq a)$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a)$
$(s[0] = a \wedge s[1] \neq a)$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

$\Downarrow$

abstraction

β
$\alpha\beta$
$\alpha\alpha\beta$

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a)$
$(s[0] = a \wedge s[1] \neq a)$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

$\Downarrow$

abstraction

$\beta \qquad \alpha^0\beta$
$\alpha\beta \qquad \alpha^1\beta$
$\alpha\alpha\beta \qquad \alpha^2\beta$

$\left.\begin{array}{l}\alpha^0\beta\\ \alpha^1\beta\\ \alpha^2\beta\end{array}\right\} \alpha^*\beta$

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a)$
$(s[0] = a \wedge s[1] \neq a)$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

$\Downarrow$

abstraction

$\beta \qquad \alpha^0\beta$
$\alpha\beta \qquad \alpha^1\beta$
$\alpha\alpha\beta \qquad \alpha^2\beta$

$\left.\begin{matrix}\alpha^0\beta\\ \alpha^1\beta\\ \alpha^2\beta\end{matrix}\right\} \alpha^*\beta \Rightarrow$

synthesis constraints

$\varphi_\alpha(1) \stackrel{\text{def}}{=} s[0] = a$
$\varphi_\alpha(2) \stackrel{\text{def}}{=} s[1] = a$
$\Rightarrow \varphi_\alpha(x) \stackrel{\text{def}}{=} s[x-1] = a$

$\varphi_\beta(0) \stackrel{\text{def}}{=} s[0] \neq a$
$\varphi_\beta(1) \stackrel{\text{def}}{=} s[1] \neq a$
$\varphi_\beta(2) \stackrel{\text{def}}{=} s[2] \neq a$
$\Rightarrow \varphi_\beta(x) \stackrel{\text{def}}{=} s[x] \neq a$

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a)$
$(s[0] = a \wedge s[1] \neq a)$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

$\Downarrow$

abstraction

β	$\alpha^0\beta$	
$\alpha\beta$	$\alpha^1\beta$	$\alpha^*\beta$
$\alpha\alpha\beta$	$\alpha^2\beta$	

$\Rightarrow$

quantified path constraints

$0 \leq i \leq 2 \wedge (\forall x. 1 \leq x \leq i \rightarrow \varphi_\alpha[x]) \wedge \varphi_\beta[i]$

$\Uparrow$

synthesis constraints

$\varphi_\alpha(1) \overset{\text{def}}{=} s[0] = a$
$\varphi_\alpha(2) \overset{\text{def}}{=} s[1] = a$
$\Rightarrow \varphi_\alpha(x) \overset{\text{def}}{=} s[x-1] = a$

$\varphi_\beta(0) \overset{\text{def}}{=} s[0] \neq a$
$\varphi_\beta(1) \overset{\text{def}}{=} s[1] \neq a$
$\varphi_\beta(2) \overset{\text{def}}{=} s[2] \neq a$
$\Rightarrow \varphi_\beta(x) \overset{\text{def}}{=} s[x] \neq a$

Synthesizing Quantified Constraints

path constrains

$(s[0] \neq a) \vee$
$(s[0] = a \wedge s[1] \neq a) \vee$
$(s[0] = a \wedge s[1] = a \wedge s[2] \neq a)$

$\Longleftrightarrow$

quantified path constraints

$0 \leq i \leq 2 \wedge (\forall x.\, 1 \leq x \leq i \rightarrow \varphi_\alpha[x]) \wedge \varphi_\beta[i]$

abstraction

$\beta \quad \alpha^0\beta$
$\alpha\beta \quad \alpha^1\beta$
$\alpha\alpha\beta \quad \alpha^2\beta$
$\} \ \alpha^*\beta$

$\Rightarrow$

synthesis constraints

$\varphi_\alpha(1) \stackrel{\text{def}}{=} s[0] = a$
$\varphi_\alpha(2) \stackrel{\text{def}}{=} s[1] = a$
$\Rightarrow \varphi_\alpha(x) \stackrel{\text{def}}{=} s[x-1] = a$

$\varphi_\beta(0) \stackrel{\text{def}}{=} s[0] \neq a$
$\varphi_\beta(1) \stackrel{\text{def}}{=} s[1] \neq a$
$\varphi_\beta(2) \stackrel{\text{def}}{=} s[2] \neq a$
$\Rightarrow \varphi_\beta(x) \stackrel{\text{def}}{=} s[x] \neq a$

Solving Quantified Constraints

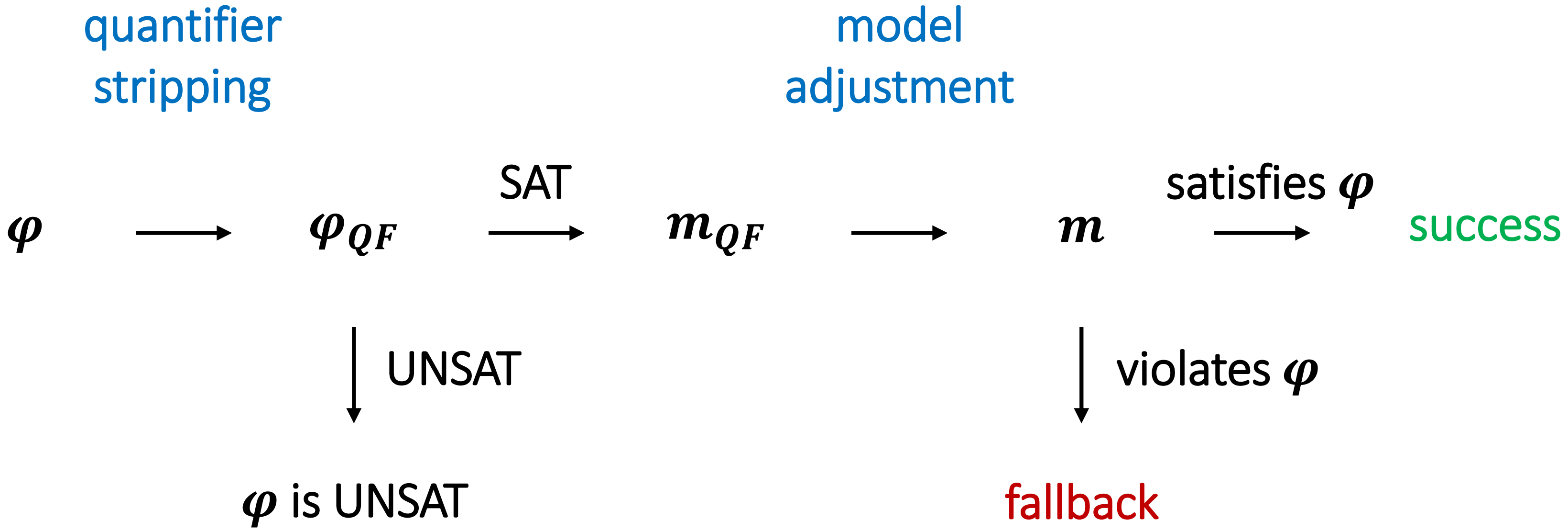
Problem:

- Quantified constraints are **challenging** for SMT solvers
- **Slower** compared to standard (quantifier-free) state merging

Our solution:

- Exploit the **specific structure** of the quantified constraints
- Add a **solving layer** on top of the SMT solver

Solving Procedure



Quantifier Stripping

Input: $\boldsymbol{\varphi}$

$$\forall x.\ 1 \leq x \leq i \rightarrow \psi(x)$$

$$\Downarrow$$

$$\underbrace{1 \leq x \leq i \rightarrow \psi(x)\ [1/x]}_{\text{instantiation}} \wedge \underbrace{\neg(1 \leq t_1 \leq i) \wedge \ \ldots \wedge \neg(1 \leq t_m \leq i)}_{\varphi \Rightarrow \neg\psi[t_j\ /\ x]}$$

Quantifier Stripping

$$\boldsymbol{\varphi} \begin{cases} (\boldsymbol{s}[\boldsymbol{n}] = \mathbf{0}) \wedge (\mathbf{1} \leq \boldsymbol{i} \leq \mathbf{10}) \wedge (\boldsymbol{s}[\boldsymbol{i} - \mathbf{1}] = \mathbf{8}) \wedge \\ (\forall \boldsymbol{x}.\ \mathbf{1} \leq \boldsymbol{x} \leq \boldsymbol{i} \rightarrow \boldsymbol{s}[\boldsymbol{x} - \mathbf{1}] \neq \mathbf{0}) \end{cases}$$

Quantifier Stripping

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$$

$$\varphi_{QF} \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (1 \leq i \rightarrow s[0] \neq 0) \wedge \neg(1 \leq n+1 \leq i) \end{array} \right.$$

Quantifier Stripping

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

$$\varphi_{QF} \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (1 \leq i \rightarrow s[0] \neq 0) \wedge \neg(1 \leq n+1 \leq i) \end{cases}$$

Quantifier Stripping

$$\varphi \begin{cases} (s[n]=0) \wedge (1 \le i \le 10) \wedge (s[i-1]=8) \wedge \\ (\forall x.\ 1 \le x \le i \rightarrow s[x-1] \ne 0) \end{cases}$$

$$\varphi_{QF} \begin{cases} (s[n]=0) \wedge (1 \le i \le 10) \wedge (s[i-1]=8) \wedge \\ (1 \le i \rightarrow s[0] \ne 0) \wedge \neg(1 \le n+1 \le i) \end{cases}$$

$\Downarrow$ SMT

satisfies φ_{QF} $\Big\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1,0,0,0,0,0,8,0]$

Quantifier Stripping

$$\boldsymbol{\varphi} \left\{ \begin{array}{c} (\boldsymbol{s}[\boldsymbol{n}] = \mathbf{0}) \wedge (\mathbf{1} \leq \boldsymbol{i} \leq \mathbf{10}) \wedge (\boldsymbol{s}[\boldsymbol{i} - \mathbf{1}] = \mathbf{8}) \wedge \\ (\forall \boldsymbol{x}.\ \mathbf{1} \leq \boldsymbol{x} \leq \boldsymbol{i} \rightarrow \boldsymbol{s}[\boldsymbol{x} - \mathbf{1}] \neq \mathbf{0}) \end{array} \right.$$

$$\boldsymbol{\varphi}_{\boldsymbol{QF}} \left\{ \begin{array}{c} (\boldsymbol{s}[\boldsymbol{n}] = \mathbf{0}) \wedge (\mathbf{1} \leq \boldsymbol{i} \leq \mathbf{10}) \wedge (\boldsymbol{s}[\boldsymbol{i} - \mathbf{1}] = \mathbf{8}) \wedge \\ (\mathbf{1} \leq \boldsymbol{i} \rightarrow \boldsymbol{s}[\mathbf{0}] \neq \mathbf{0}) \wedge \neg(\mathbf{1} \leq \boldsymbol{n} + \mathbf{1} \leq \boldsymbol{i}) \end{array} \right.$$

$\Downarrow$ SMT

violates $\boldsymbol{\varphi}$ $\left\{ \quad \boldsymbol{n} \mapsto \mathbf{7}, \boldsymbol{i} \mapsto \mathbf{7}, \boldsymbol{s} \mapsto [\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{8}, \mathbf{0}] \right.$

Model Adjustment: Duplication

$$\boldsymbol{\varphi} \left\{ \begin{array}{c} (\boldsymbol{s}[\boldsymbol{n}] = \mathbf{0}) \wedge (\mathbf{1} \leq \boldsymbol{i} \leq \mathbf{10}) \wedge (\boldsymbol{s}[\boldsymbol{i} - \mathbf{1}] = \mathbf{8}) \wedge \\ (\forall \boldsymbol{x}.\ \mathbf{1} \leq \boldsymbol{x} \leq \boldsymbol{i} \rightarrow \boldsymbol{s}[\boldsymbol{x} - \mathbf{1}] \neq \mathbf{0}) \end{array} \right.$$

violates $\boldsymbol{\varphi}$ $\left\{ \quad \boldsymbol{n} \mapsto \mathbf{7}, \boldsymbol{i} \mapsto \mathbf{7}, \boldsymbol{s} \mapsto [\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{8}, \mathbf{0}] \right.$

Model Adjustment: Duplication

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \land (1 \leq i \leq 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$$

evaluate

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

$$1 \leq x \leq 7$$

Model Adjustment: Duplication

φ $\left\{ \begin{array}{l} (s[n] = 0) \land (1 \leq i \leq 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$

evaluate

violates φ $\left\{ \quad n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0] \right.$

$1 \leq x \leq 7$

$s[0], s[1], \ldots s[5], s[6]$

Model Adjustment: Duplication

φ $\begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$

evaluate

violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots s[5], s[6]$$

Model Adjustment: Duplication

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots s[5], s[6]$$

transform

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 1, 0]$$

Model Adjustment: Duplication

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots s[5], s[6]$$

transform

$n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 1, 0]$

Model Adjustment: Duplication

φ $\left\{ \begin{array}{c} (s[n] = 0) \land (1 \leq i \leq 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$

violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$1 \leq x \leq 7$

$s[0], s[1], \ldots s[5], s[6]$

transform

still violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 1, 0]$

Model Adjustment: Repair

$$\varphi \begin{cases} (s[n] = 0) \land (1 \leq i \leq 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

$$\text{violates } \varphi \begin{cases} n \mapsto 7, i \mapsto 7, s \mapsto [1,1,1,1,1,1,1,0] \end{cases}$$

Model Adjustment: Repair

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \le i \le 10) \wedge \color{red}{(s[i-1] = 8)} \wedge \\ (\forall x.\ 1 \le x \le i \to s[x-1] \ne 0) \end{cases}$$

evaluate

violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 1, 0]$

$$s[6]$$

$$s[i-1] = 8 \qquad (s[x-1] \ne 0)[7 / x]$$

Model Adjustment: Repair

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \land (1 \le i \le 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \le x \le i \to s[x-1] \ne 0) \end{array} \right.$$

evaluate

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1,1,1,1,1,1,1,0]$$

$$\varphi'_{QF} \leftarrow \varphi_{QF} \land (s[x-1] \ne 0)[7 / x]$$

Model Adjustment: Repair

φ: $(s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0)$

violates φ: $n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 1, 0]$

evaluate

$$\varphi'_{QF} \leftarrow \varphi_{QF} \wedge (s[6] \neq 0)$$

Model Adjustment: Repair

$$\varphi'_{QF} \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (1 \leq i \rightarrow s[0] \neq 0) \wedge \neg(1 \leq n+1 \leq i) \wedge (s[6] \neq 0) \end{array} \right.$$

$\Downarrow$ SMT

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

Model Adjustment: Repair

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

violates φ $\Big\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

Model Adjustment: Repair

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$$

evaluate

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

$$1 \leq x \leq 7$$

Model Adjustment: Repair

$$\varphi \begin{cases} (s[n] = 0) \land (1 \leq i \leq 10) \land (s[i-1] = 8) \land \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

evaluate

violates φ

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots, s[5], s[6]$$

Model Adjustment: Repair

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$$

violates φ $\Big\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots, s[5], \sout{s[6]}$$

conflict

Model Adjustment: Repair

$$\varphi \left\{ \begin{array}{c} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{array} \right.$$

violates φ $\{$

$$n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$$

$$1 \leq x \leq 7$$

$$s[0], s[1], \ldots, s[5]$$

Model Adjustment: Repair

$$\varphi \begin{cases} (s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge \\ (\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0) \end{cases}$$

violates φ $\{$ $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$1 \leq x \leq 7$

$s[0], s[1], \ldots, s[5]$

transform

$n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 8, 0]$

Model Adjustment: Repair

$\boldsymbol{\varphi}$ $\left\{ \begin{array}{l} (\boldsymbol{s}[\boldsymbol{n}] = \mathbf{0}) \land (\mathbf{1} \leq \boldsymbol{i} \leq \mathbf{10}) \land (\boldsymbol{s}[\boldsymbol{i} - \mathbf{1}] = \mathbf{8}) \land \\ (\forall \boldsymbol{x}.\ \mathbf{1} \leq \boldsymbol{x} \leq \boldsymbol{i} \rightarrow \boldsymbol{s}[\boldsymbol{x} - \mathbf{1}] \neq \mathbf{0}) \end{array} \right.$

violates $\boldsymbol{\varphi}$ $\{$ $\boldsymbol{n} \mapsto \mathbf{7}, \boldsymbol{i} \mapsto \mathbf{7}, \boldsymbol{s} \mapsto [\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{8}, \mathbf{0}]$

$\mathbf{1} \leq \boldsymbol{x} \leq \mathbf{7}$

$\boldsymbol{s}[\mathbf{0}], \boldsymbol{s}[\mathbf{1}], \ldots, \boldsymbol{s}[\mathbf{5}]$

transform

$\boldsymbol{n} \mapsto \mathbf{7}, \boldsymbol{i} \mapsto \mathbf{7}, \boldsymbol{s} \mapsto [\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{8}, \mathbf{0}]$

Model Adjustment: Repair

φ { $(s[n] = 0) \wedge (1 \leq i \leq 10) \wedge (s[i-1] = 8) \wedge$
$(\forall x.\ 1 \leq x \leq i \rightarrow s[x-1] \neq 0)$

violates φ { $n \mapsto 7, i \mapsto 7, s \mapsto [1, 0, 0, 0, 0, 0, 8, 0]$

$1 \leq x \leq 7$

$s[0], s[1], \ldots, s[5]$

transform

satisfies φ { $n \mapsto 7, i \mapsto 7, s \mapsto [1, 1, 1, 1, 1, 1, 8, 0]$

Additional Contributions

Incremental state merging

- Merging on-the-fly
  - Instead of merging after the exploration of the code fragment
- Handling complex loops (exponential execution trees)

More details in the paper…

Implementation: KLEE

Main changes:

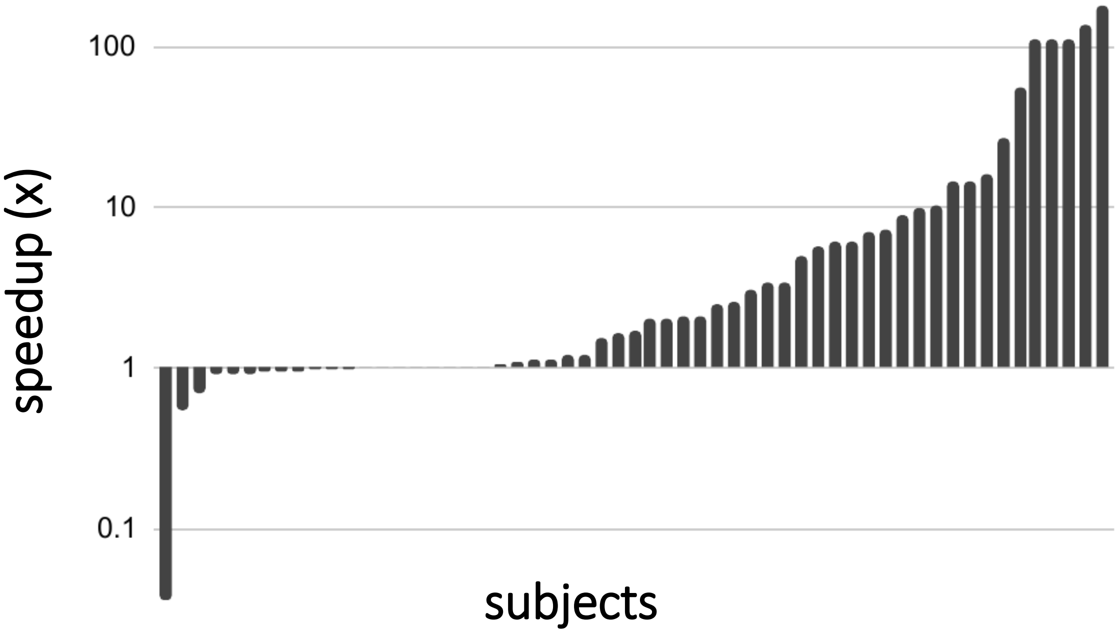
- Extended the expression language
- Extended state merging capabilities
- Extended the solver chain for solving quantified queries

Evaluation

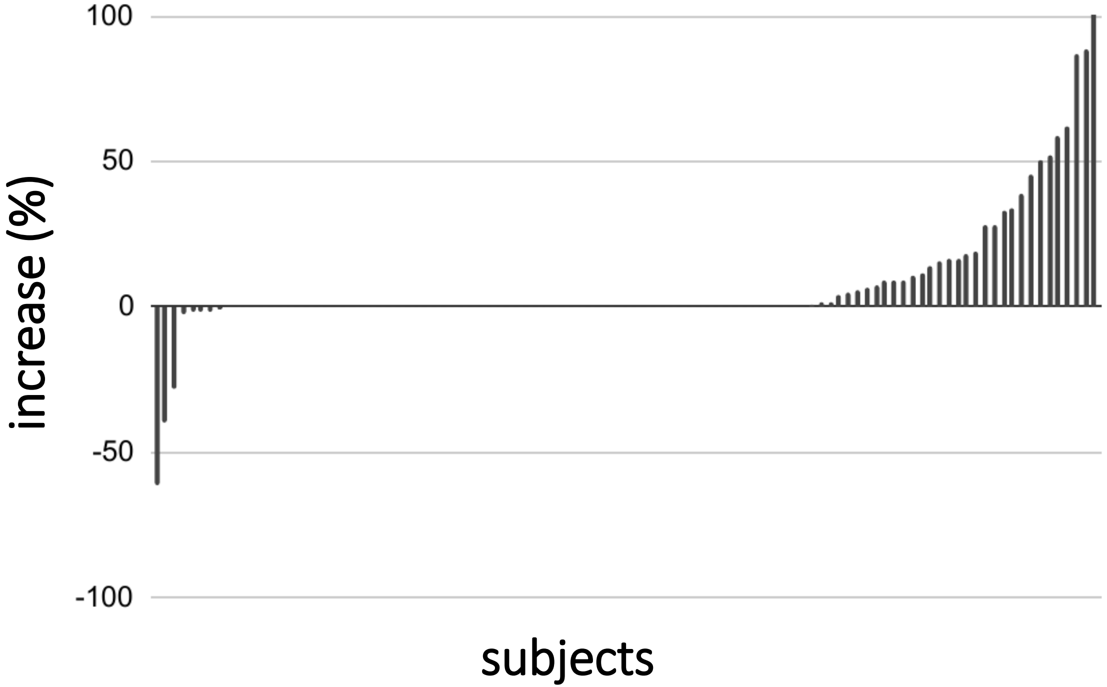
Benchmarks

- GNU oSIP *(35 subjects)*
- wget *(31 subjects)*
- GNU libtasn1 *(13 subjects)*
- libpng *(12 subjects)*
- APR (Apache Portable Runtime) *(20 subjects)*
- json-c *(5 subjects)*
- busybox *(30 subjects)*

Evaluation: Analysis Time



Evaluation: Coverage



Found Bugs

Detected bugs in ***klee-uclibc*** in the experiments with ***busybox***

- Two *memory out-of-bound's*
- Confirmed and fixed

Summary

- State merging using quantified constraints
- Specialized solving procedure for quantified constraints
- Evaluated on real-world benchmarks
- Found bugs

https://github.com/davidtr1037/klee-quantifiers