A Bounded Symbolic-Size Model for Symbolic Execution

David Trabish

Shachar Itzhaky

Noam Rinetzky

Tel-Aviv University, Israel

Technion, Israel

ESEC/FSE 2021

Symbolic Execution: Introduction

- Systematic program analysis technique
  - Run program with symbolic inputs
- Many applications:
  - Test input generation
  - Bug finding
  - …
- Active research area
- Used in industry

SAMSUNG

TRAIL OF BITS

FUJITSU

Microsoft

KLEE

Motivation

- Size of input affects program behavior

Motivation

- Size of input affects program behavior

$|input| \geq 5$

```
int osip_via_parse(const char *hvalue) {
  if (hvalue == NULL) return OSIP_BADPARAMETER;
  const char *version = strchr(hvalue, '/');
  if (version == NULL) return OSIP_SYNTAXERROR;
  const char *protocol = strchr(version + 1, '/');
  if (protocol == NULL) return OSIP_SYNTAXERROR;
  if (protocol - version < 2) return OSIP_SYNTAXERROR;
  ...
  const char *host = strchr(protocol + 1, ' ');
  if (host == NULL) return OSIP_SYNTAXERROR;
  if (host == protocol + 1) {
    while (0 == strncmp(host, " ", 1)) {
      host++;
      if (strlen(host) == 1) return OSIP_SYNTAXERROR;
    }
    host = strchr(host + 1, ' ');
  }
  ...
```

BUG

$|input| = 1$

```
int osip_uri_parse_headers(const char *headers) {
  const char *equal = strchr(headers, '=');
  const char *_and = strchr(headers + 1, '&');
  ...
}
```

BUG

Motivation

- Size of input affects program behavior
- The problem: concrete-size model

$$|input| \geq 5$$

```
int osip_via_parse(const char *hvalue) {
  if (hvalue == NULL) return OSIP_BADPARAMETER;
  const char *version = strchr(hvalue, '/');
  if (version == NULL) return OSIP_SYNTAXERROR;
  const char *protocol = strchr(version + 1, '/');
  if (protocol == NULL) return OSIP_SYNTAXERROR;
  if (protocol - version < 2) return OSIP_SYNTAXERROR;
  ...
  const char *host = strchr(protocol + 1, ' ');
  if (host == NULL) return OSIP_SYNTAXERROR;
  if (host == protocol + 1) {
    while (0 == strncmp(host, " ", 1)) {
      host++;
      if (strlen(host) == 1) return OSIP_SYNTAXERROR;
    }
    host = strchr(host + 1, ' ');
  }
  ...
```

BUG

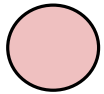
$$|input| = 1$$

```
int osip_uri_parse_headers(const char *headers) {
  const char *equal = strchr(headers, '=');
  const char *_and = strchr(headers + 1, '&');
  ...
}
```

BUG

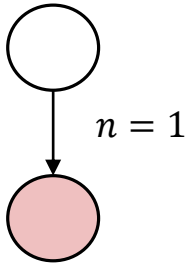
Concrete-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n);
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



Concrete-Size Model

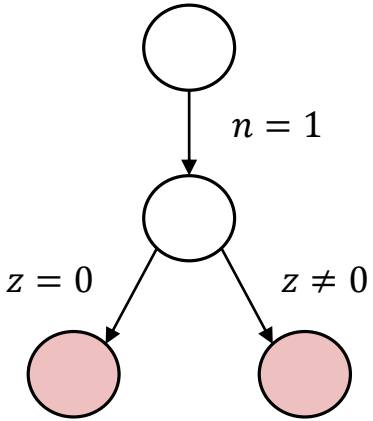
```
int n; // symbolic
int z; // symbolic

char *p = malloc(n);
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



concretize symbolic size n

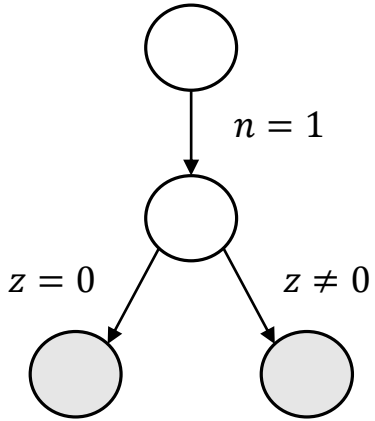
Concrete-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n);
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



Concrete-Size Model

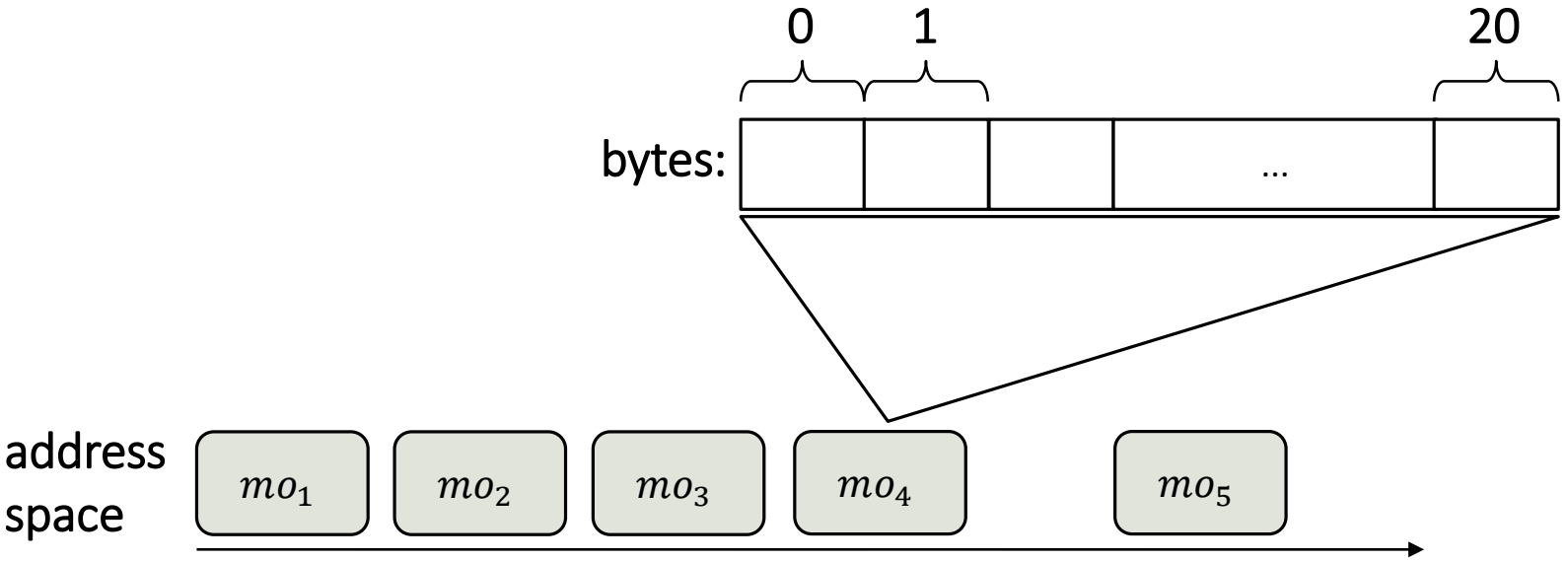
```
int n; // symbolic
int z; // symbolic

char *p = malloc(n);
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



only 2 paths explored

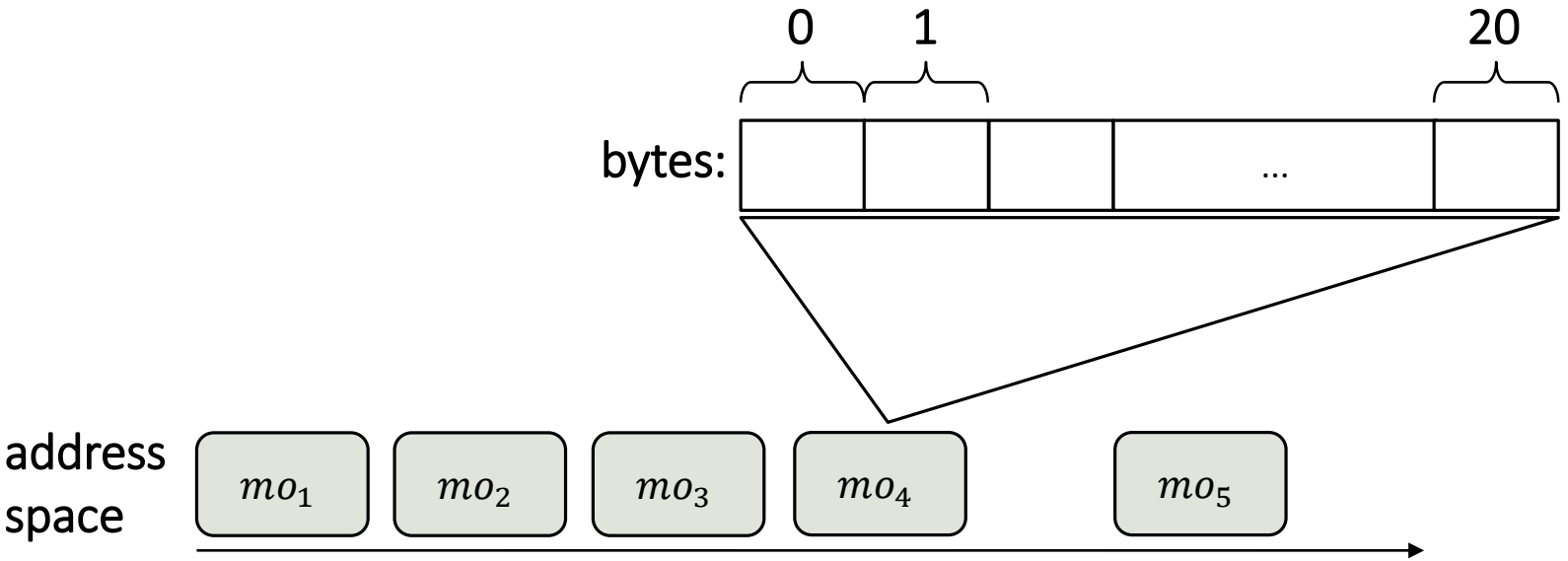
Concrete-Size Model

- Linear address space
- Explicit encoding *(QF_ABV)*



Fully Symbolic-Size Model

- Linear address space → impossible to avoid overlapping
- Explicit encoding *(QF_ABV)* → high memory consumption



Bounded Symbolic-Size Model

A memory object has:

- Fixed (concrete) capacity
- Symbolic size : 0, 1, … , capacity

Bounded Symbolic-Size Model

A memory object has:

- Fixed (concrete) capacity
- Symbolic size : 0, 1, … , capacity

- Works with a linear address space ✓

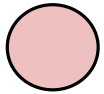
Bounded Symbolic-Size Model

A memory object has:

- Fixed (concrete) capacity
- Symbolic size : 0, 1, … , capacity

- Works with a linear address space ✓
- Controllable memory consumption (user-specified capacity) ✓

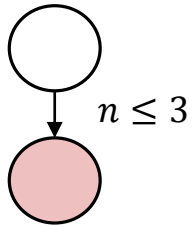
Symbolic-Size Model



```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```

Symbolic-Size Model

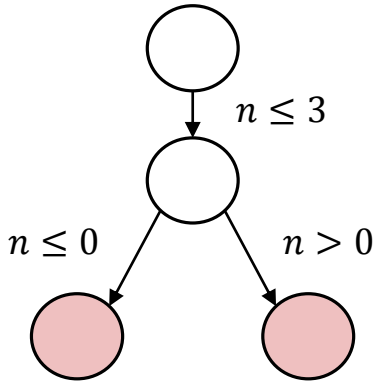
```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



add capacity constraint

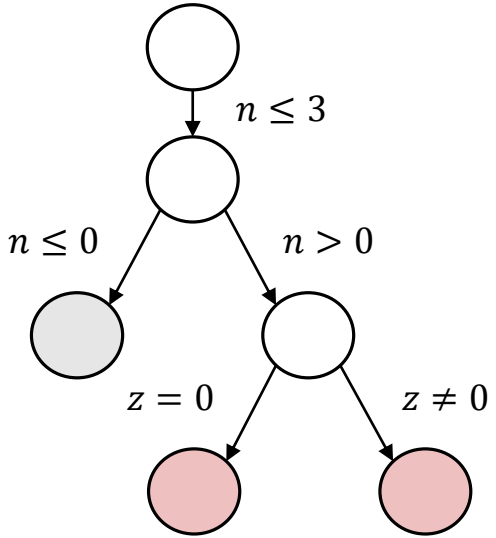
Symbolic-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



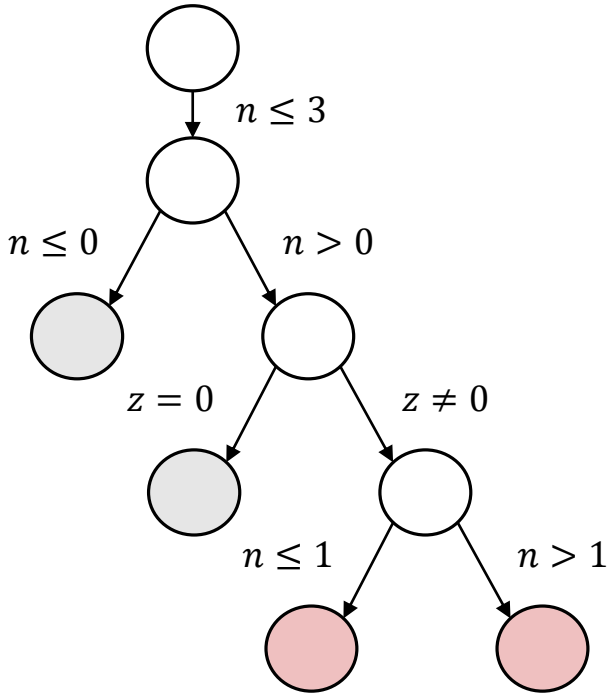
Symbolic-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



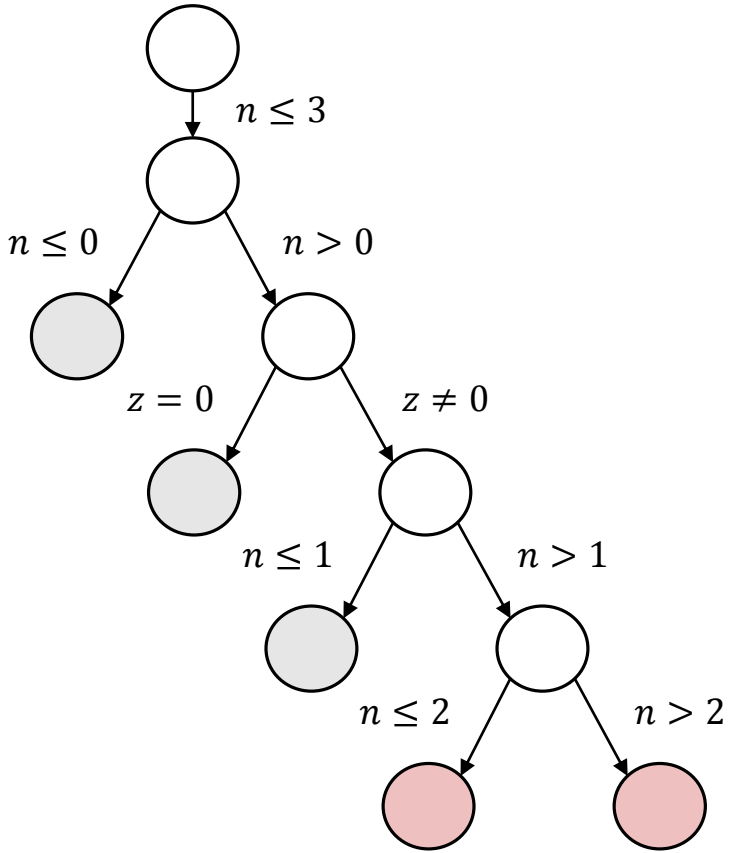
Symbolic-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



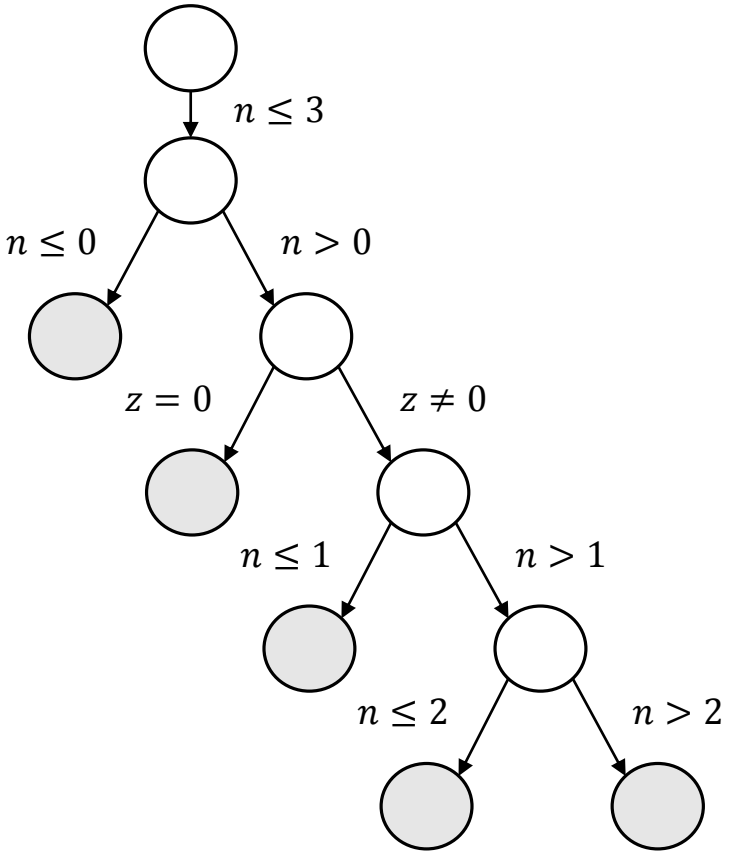
Symbolic-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



Symbolic-Size Model

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```



5 paths explored

Arising Challenges

- Additional symbolic-size expressions
- Amplifies path explosion
  - Especially with **size-dependent loops**

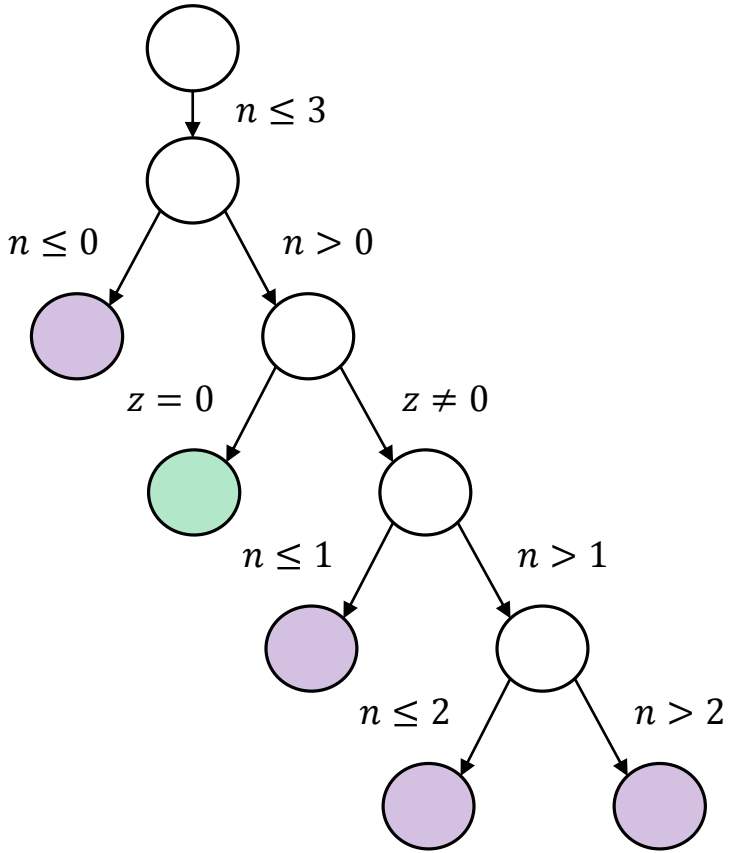
Merging Approach

- Detect **symbolic-size dependent** loops
- Execute the loop till **full exploration**
- **Merge** the resulting states

Merging Approach

```
int n; // symbolic
int z; // symbolic

char *p = malloc(n); // capacity = 3
for (unsigned i = 0; i < n; i++) {
  if (z == 0) {
    break;
  }
  p[i] = i;
}
```

group states by loop exit



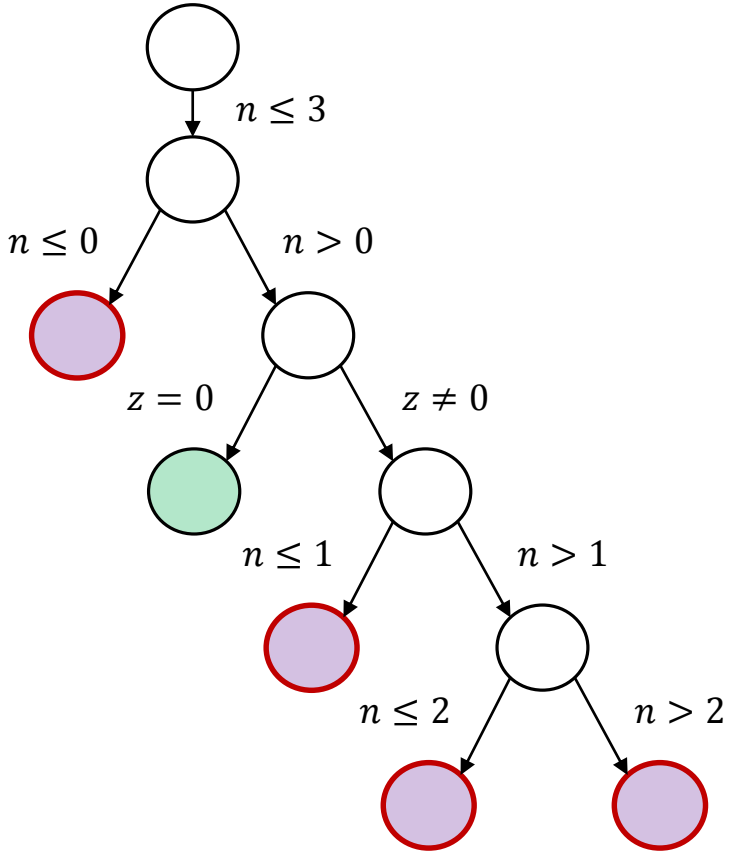
Merging Approach

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint

$(n > 0 \wedge z = 0)$

merged constraint



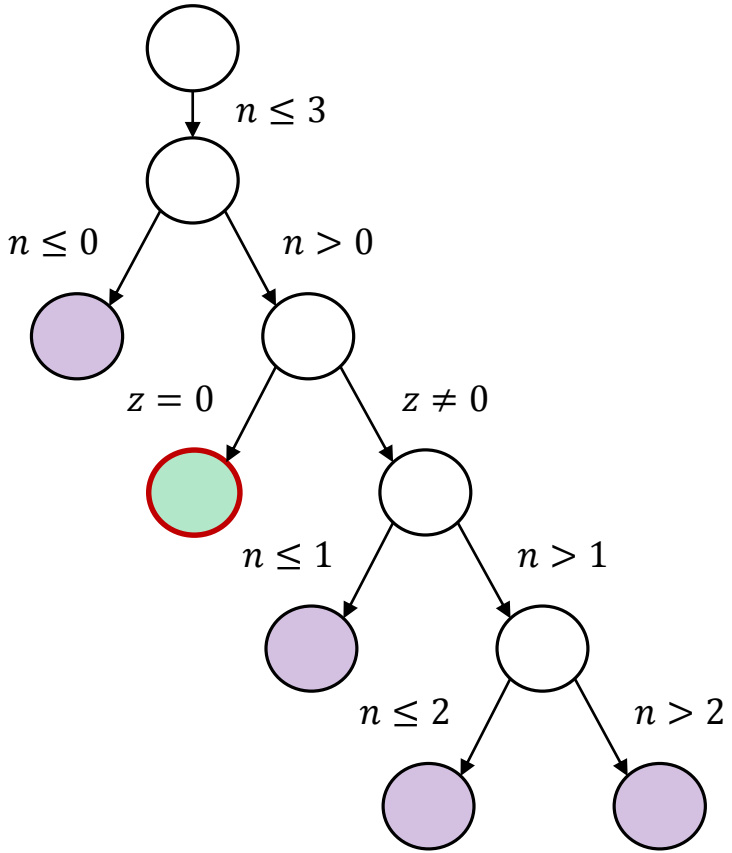
Merging Approach

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint

$(n > 0 \wedge z = 0)$

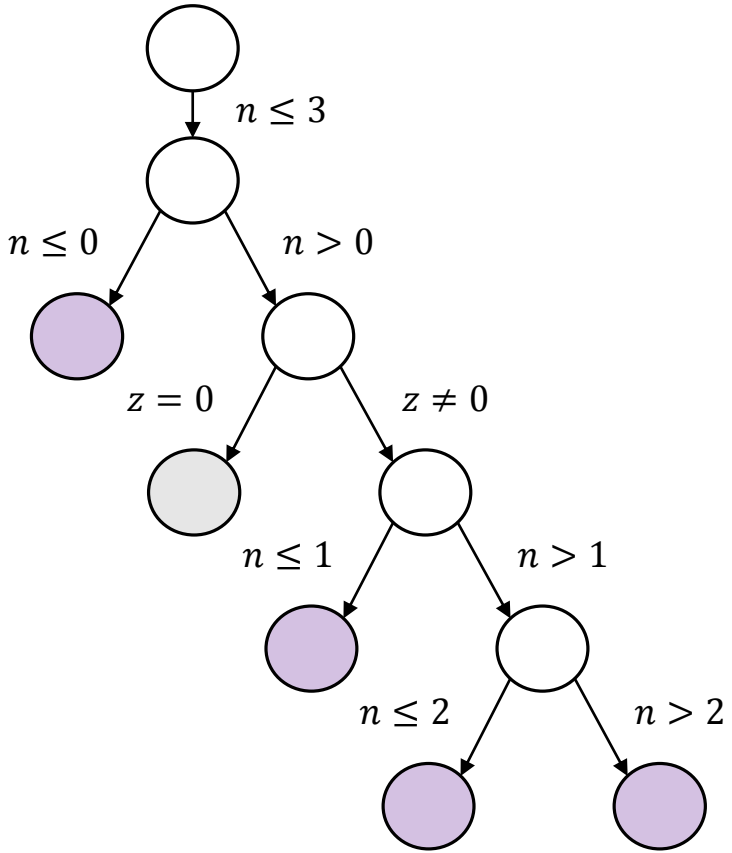
merged constraint



Merging Optimization

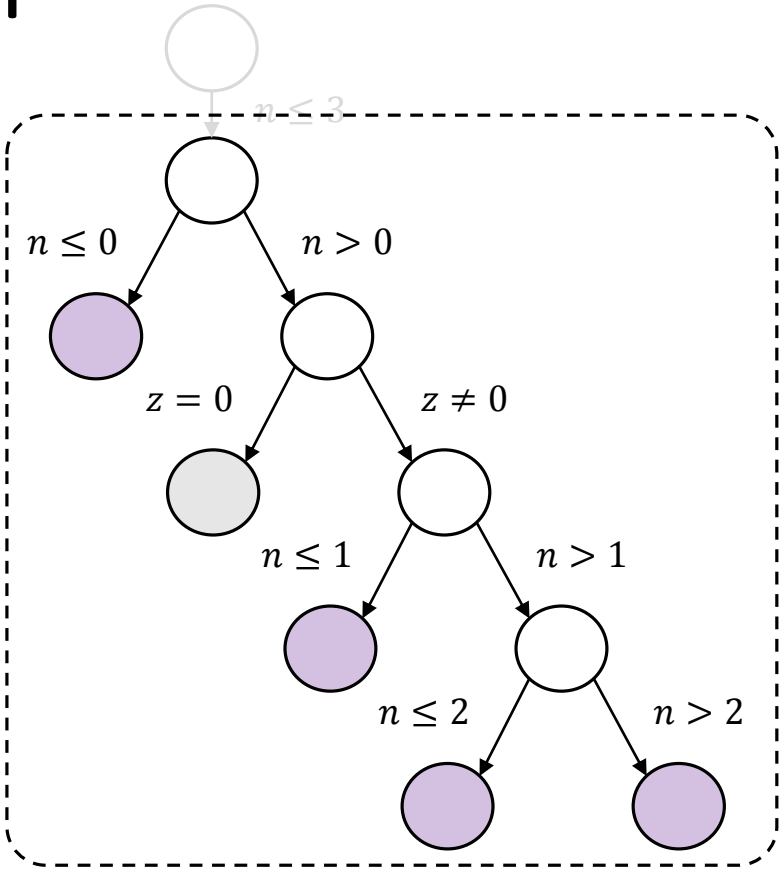
$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



Merging Optimization

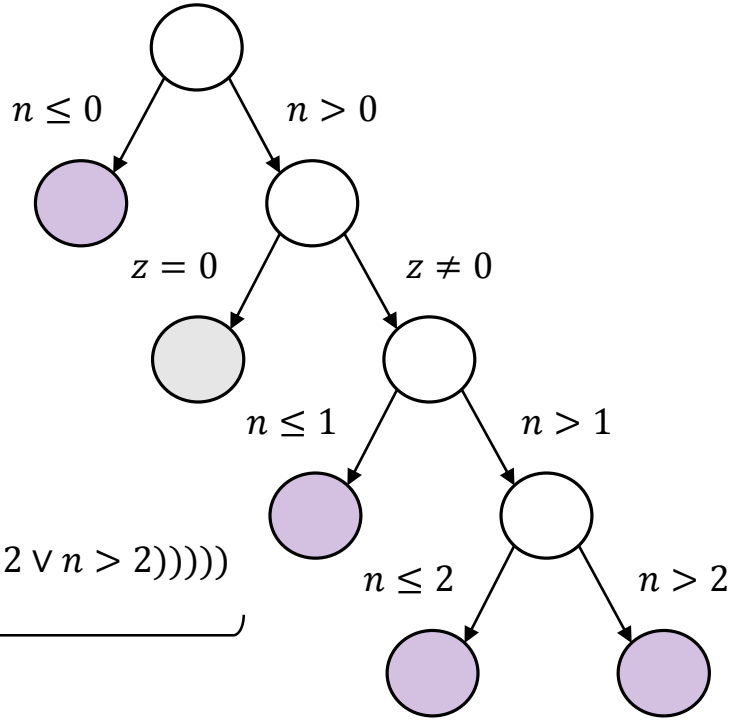
$$\underbrace{\begin{array}{l}(n \leq 0) \vee \\ (n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee \\ (n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee \\ (n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)\end{array}}_{\text{merged constraint}}$$



Merging Optimization

$$(n \leq 0) \vee$$
$$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$$
$$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$$
$$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$$

merged constraint



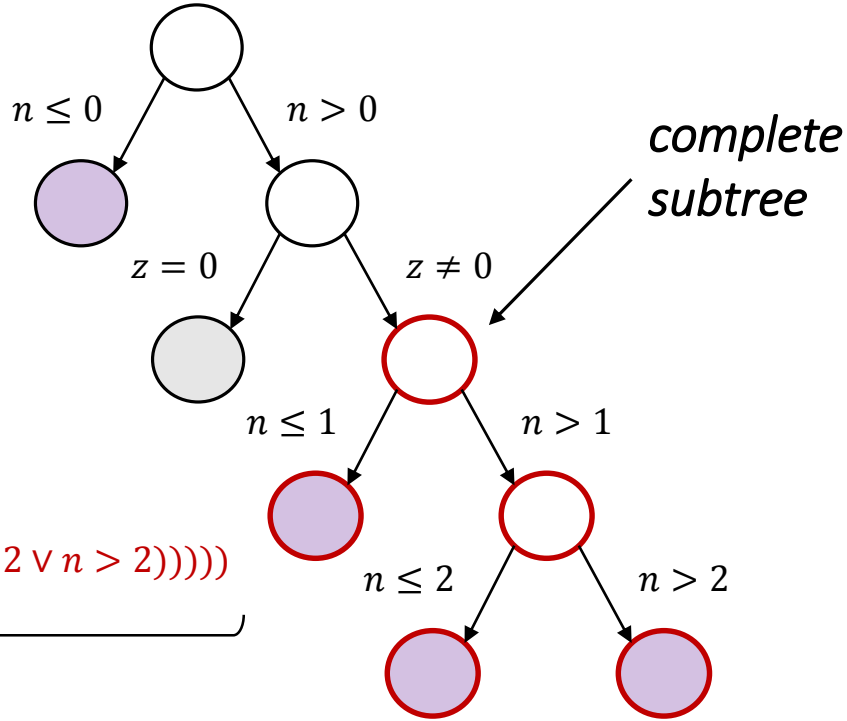
$$(n \leq 0 \vee (n > 0 \wedge z \neq 0 \wedge (n \leq 1 \vee (n > 1 \wedge (n \leq 2 \vee n > 2)))))$$

merged constraint

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



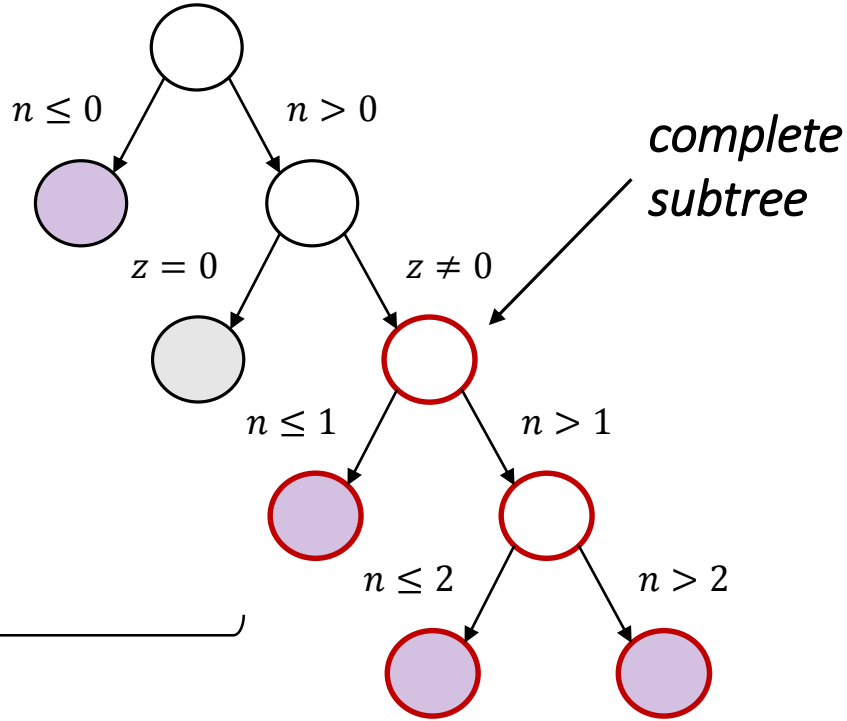
$(n \leq 0 \vee (n > 0 \wedge z \neq 0 \wedge (n \leq 1 \vee (n > 1 \wedge (n \leq 2 \vee n > 2)))))$

merged constraint

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



$(n \leq 0 \vee (n > 0 \wedge z \neq 0))$

merged constraint

Evaluation

API Testing

- GNU libtasn1 *(17 API's)*
- libpng *(13 API's)*
- GNU oSIP *(48 API's)*

Whole-program testing

- GNU Coreutils *(99 programs)*

Implementation

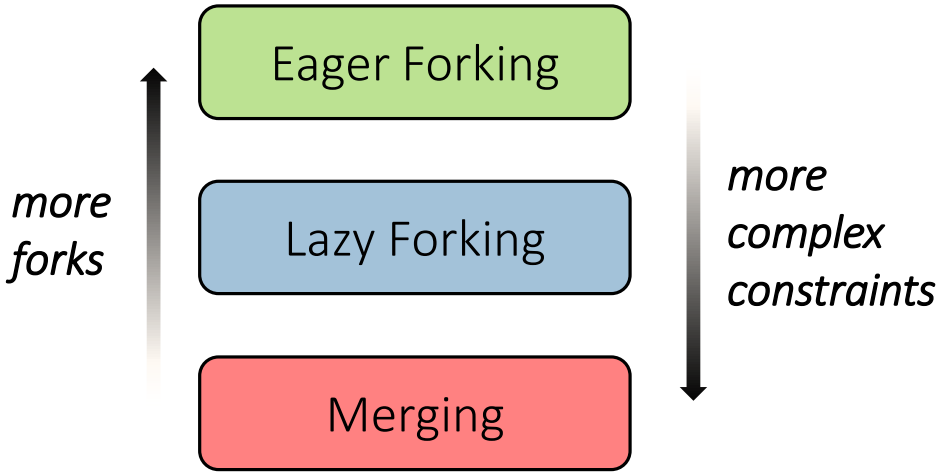
- On top of ***KLEE***

Evaluation: Approaches

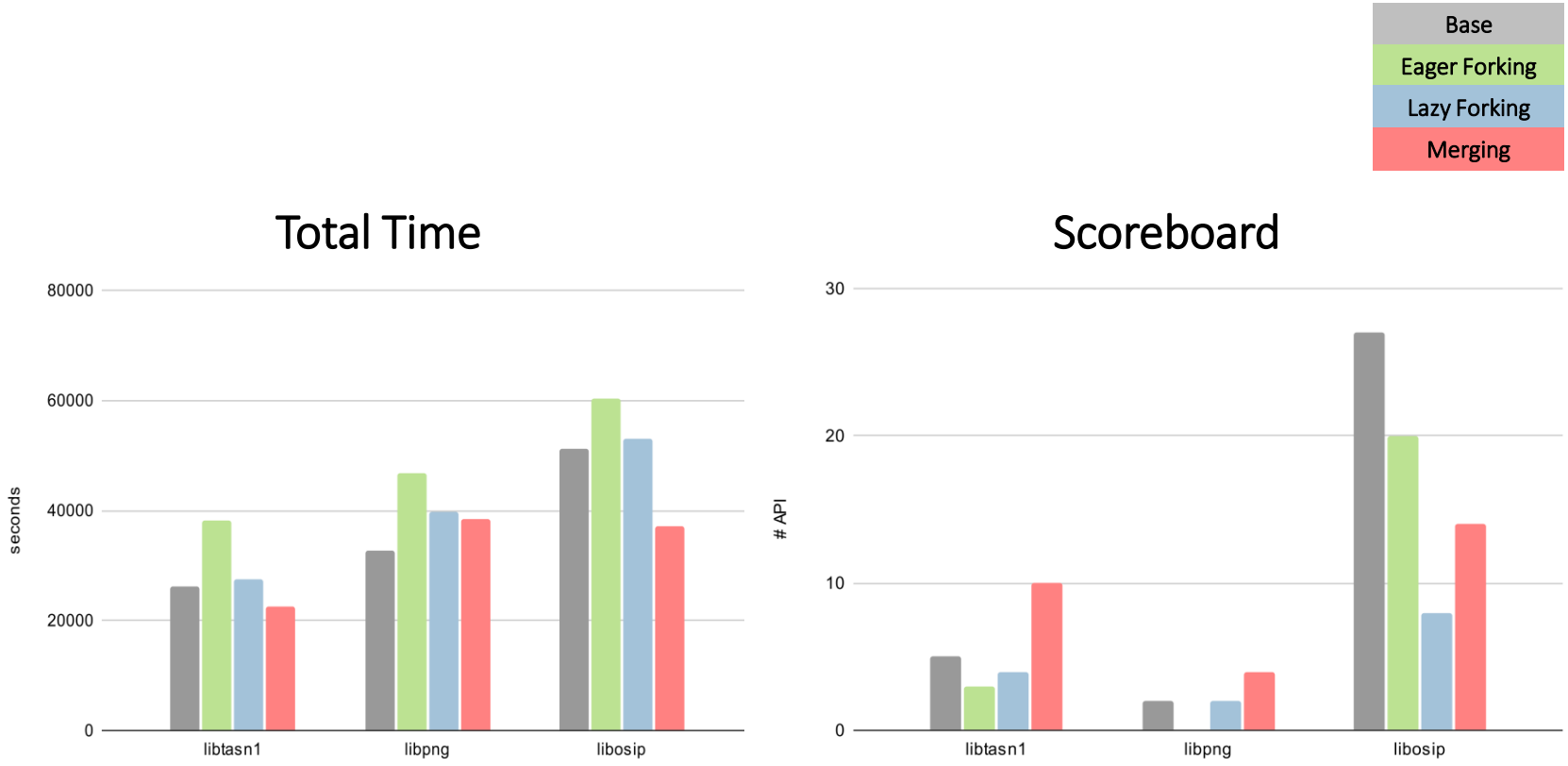
Concrete-size Model

Base

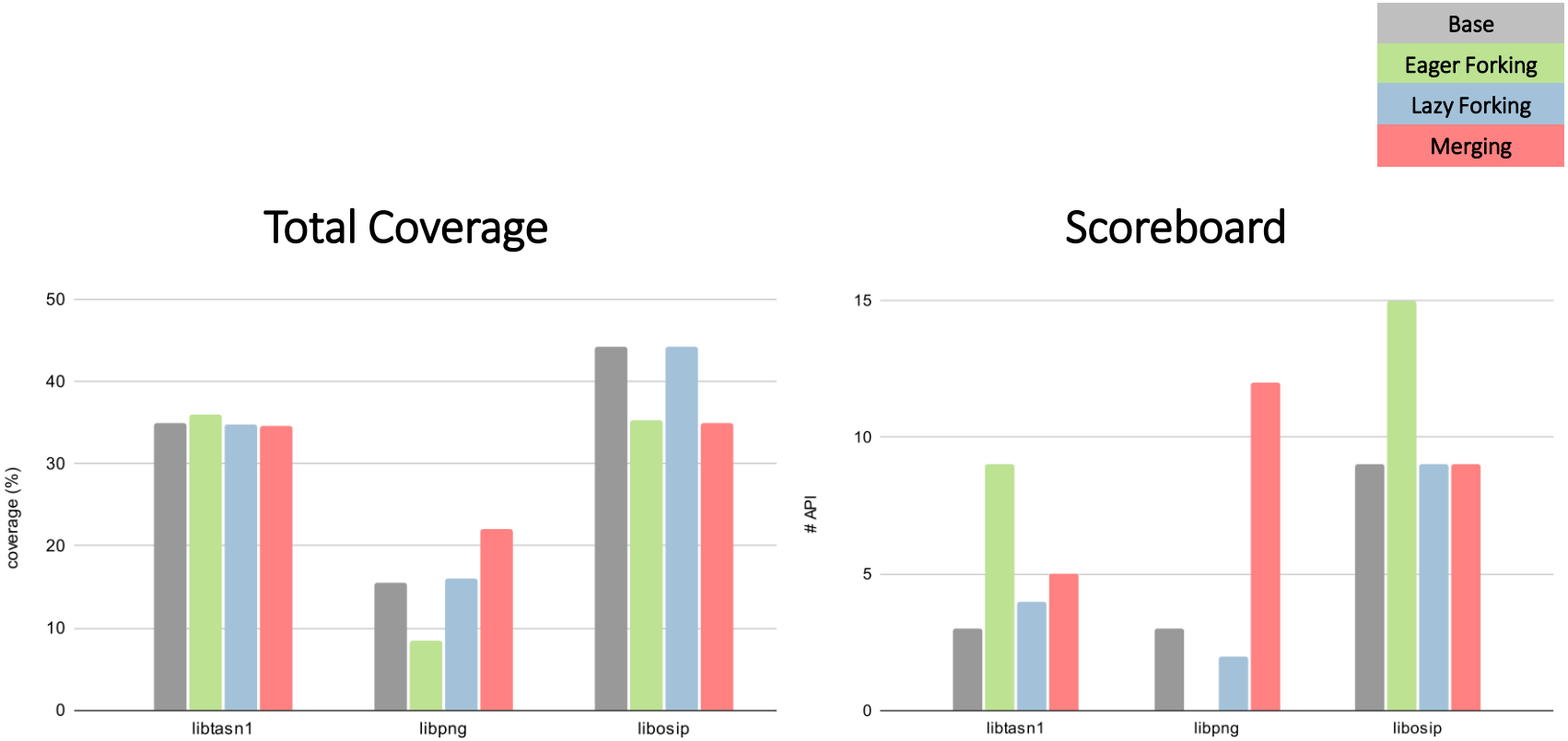
Bounded Symbolic-Size Model



API Testing: Analysis Time



API Testing: Coverage



API Testing: Found Bugs

- GNU libtasn1
  - one *out-of-bound-read*
- GNU oSIP
  - three *out-of-bound-read's*
  - one *integer-underflow*

Evaluation: GNU Coreutils

- In 94 programs, all approaches timeout:
  - Compare coverage
- In the rest 5 programs:
  - Merging approach is faster

Summary

- Bounded modeling of variable-size inputs
- Evaluated in API testing and whole-program testing
- Found previously unknown bugs

Future Work

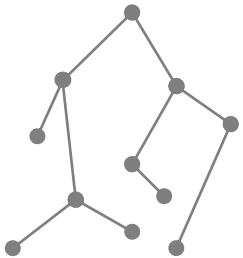
- Applying in other domains (patch testing, program repair, …)
- Better encoding in state merging

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

Backup

Main Challenges

Path Explosion



Constraint Solving

$$x = 1 \wedge z > 1 \wedge select(a_2, 7) = 1$$

$$y > 10 \wedge z > 1 \wedge z + y < 77$$

$$a > b + 23 \wedge c\ - a > 56$$

$$w > s * 6 \wedge t\ > w$$

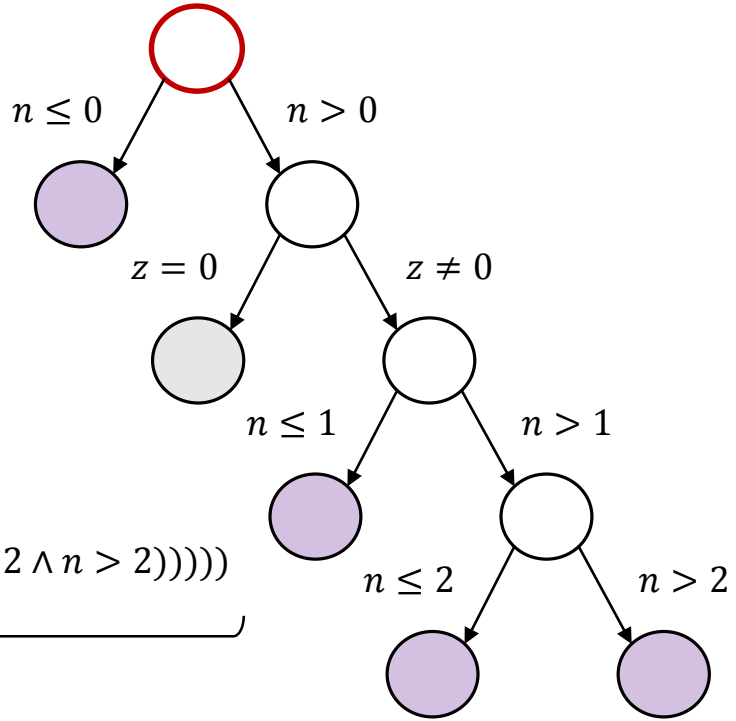
Concrete-Size Model

- A memory object has a concrete size
- Leads to concretizations (on allocations)
  - Less coverage
  - Missed bugs

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



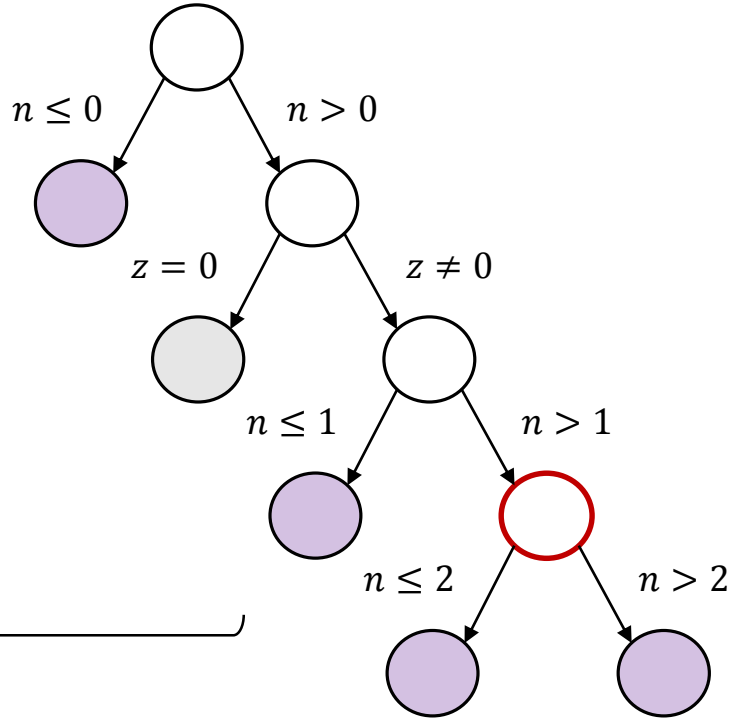
$(n \leq 0 \vee (n > 0 \wedge z \neq 0 \wedge (n \leq 1 \vee (n > 1 \wedge (n \leq 2 \wedge n > 2)))))$

merged constraint

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



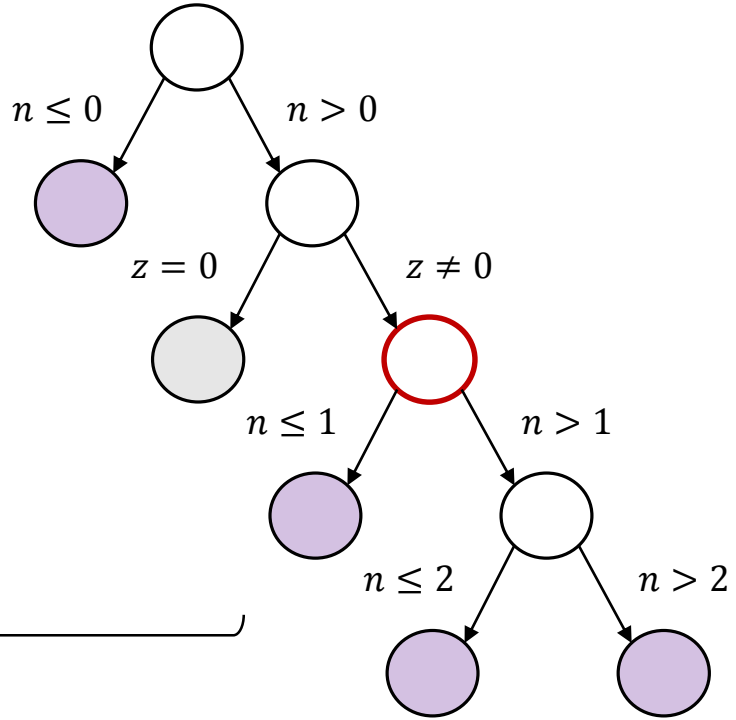
$(n \leq 2 \vee n > 2)$

merged constraint

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



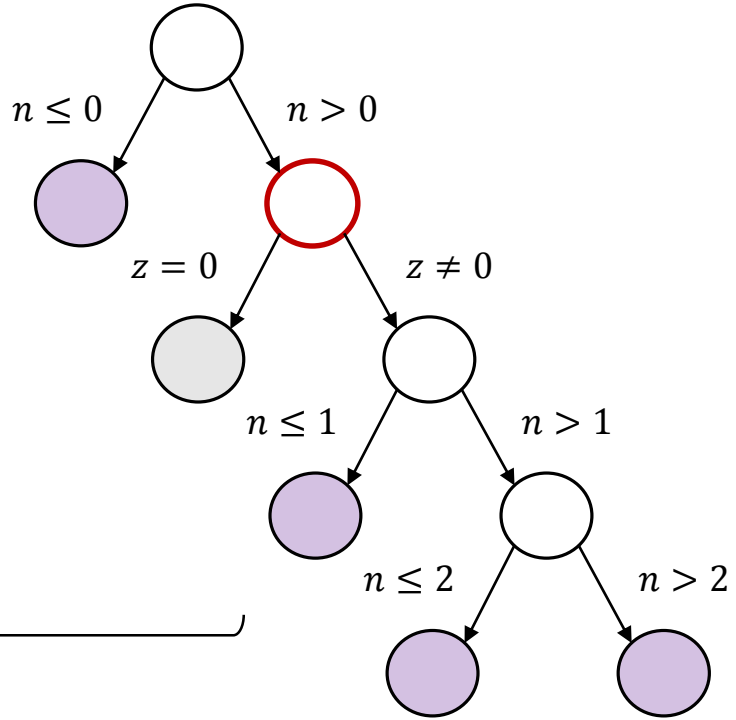
$(n \leq 1 \vee (n > 1 \wedge (n \leq 2 \wedge n > 2)))$

merged constraint

Merging Optimization

$(n \leq 0) \vee$
$(n > 0 \wedge z \neq 0 \wedge n \leq 1) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n \leq 2) \vee$
$(n > 0 \wedge z \neq 0 \wedge n > 1 \wedge n > 2)$

merged constraint



$(z \neq 0 \wedge (n \leq 1 \vee (n > 1 \wedge (n \leq 2 \wedge n > 2))))$

merged constraint

Evaluation

Compare different modes:

- Concrete-size model
  - *Base*
    - Concretize to max value
- *Bounded symbolic-size model*
  - *Eager Forking*
    - Fork at allocation time for each possible value
  - *Lazy Forking*
    - Fork on-demand (standard)
  - *Merging*

Summary & Future Work

- Bounded modeling of variable-size inputs
- Evaluated in API testing the whole-program testing
- Found previously unknown bugs

Future research directions:

- Applying in other domains (patch testing, program repair, …)
- Better encoding in state merging

Available on GitHub: https://github.com/davidtr1037/klee-symsize