tcs Research

Inventing for impact

VeriFuzz 1.4: Checking for (Non-)termination

Ravindra Metta[1,2] *r.metta@tcs.com*

Prasanth yeduru[1]

Hrishikesh Karmarkar[1]

Raveendra Medicherla[1]

[1] tcs Research

[2] Technische Universität München

April 23, 2023

Non-Termination

Listing 1.1. Program P

```
1  i=1; j=nondet();
2  while(j!=1) {
3     i=i+1;
4     if (i==nondet()) exit(0);
5     j=nondet(); }
```

Suppose nondet() = 129

Suppose nondet() = 1

Suppose nondet() = 5

Non-Termination: Abstract Interpretation imprecise

Listing 1.1. Program P

```
1  i=1; j=nondet();
2  while(j!=1) {
3     i=i+1;
4     if (i==nondet()) exit(0);
5     j=nondet(); }
```

j : [INT_MIN, INT_MAX]

i : <2, INT_MAX]

j: [INT_MIN, INT_MAX]

Loop head and exit are both reachable

Non-Termination via concretization ICSME'22

Listing 1.1. Program P

```
1  i=1; j=nondet();
2  while(j!=1) {
3     i=i+1;
4     if (i==nondet()) exit(0);
5     j=nondet(); }
```

Listing 1.2. Program P'

```
1  i=1; j=129;
2  while(j!=1) {
3     i=i+1;
4     if (i==1) exit(0);
5     j=5; }
```

j : [129,129]

i : <2. INT_MAX]

j : [5,5]

False

Loop head reachable, but not loop exit

How to concretize: via a test generator (VeriFuzz 1,2)

Bounded Termination

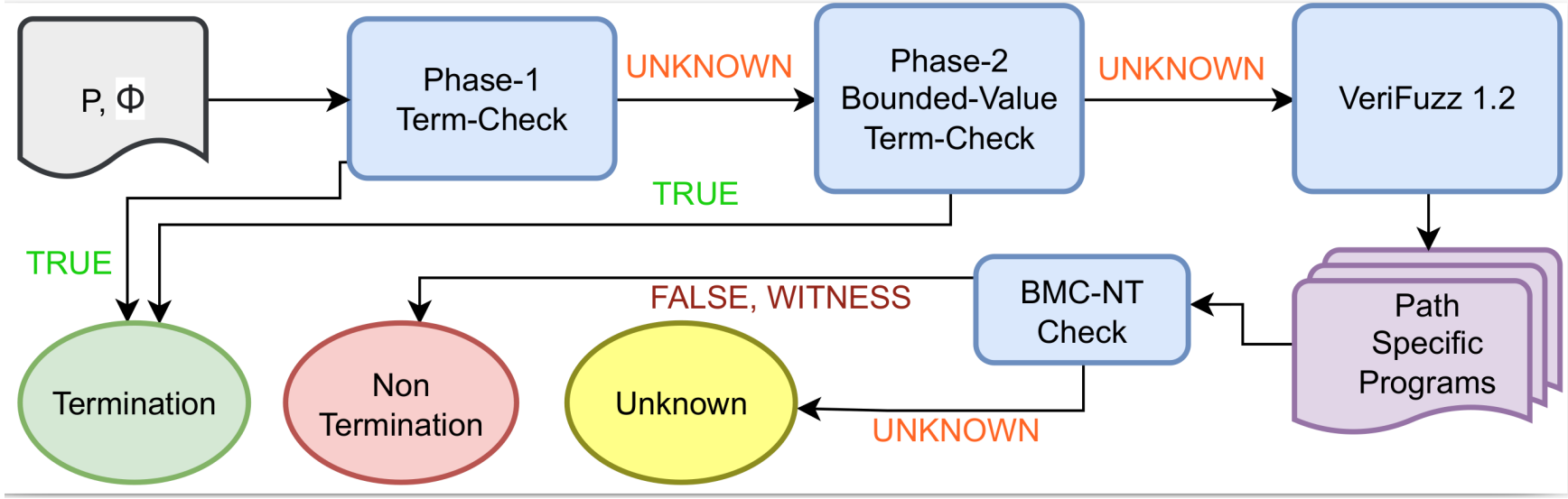
```
int main(){

  int in_len = __VERIFIER_nondet_int();
  if(in_len < 1){return 1;}
  char* in = alloca(in_len);
  for(int i=0; i<in_len-1; i++)
  {
    in[i] = __VERIFIER_nondet_char();
  }
  in[in_len-1]=0;    ensures termination

  return atoi(in);
}
```

```
int atoi(const char* s) {
  long int v=0;
  int sign=1;
  while ( *s == ' ' ) s++;
  switch (*s) {
  case '-': sign=-1;
  case '+': ++s;
  }
  while ( (*s - '0') > 0 && (*s - '0') < 10) {
    v=v*10+*s-'0'; ++s;
  }
  return sign==-1?-v:v;
}
```

- Observation: if the loop terminates for any n > 1, it terminates for all n
- Bounded check: *learn* a sufficient range for n, and if P terminates within this range, guess it is terminating
  - For e.g. restrict n to [-16, 15]
  - BMC works; but result *unsound* due to range restriction
  - Hence, can lead to false negatives

VeriFuzz 1.4 : Tool Architecture

- **VeriFuzz 1.2 for test generation for NT, CBMC for Bounded T-Check**



VeriFuzz 1.2 Results

- **Correctly identified: T 865/1043, NT 351/766**
- **3 False Negatives**
- **0 False Positives, but PSPs need improvement**

Future

- Recurrent set based proofs for NT via directed tests
- Ranking functions with small model properties

tcs Research

Inventing for impact

Thank you