Patch Validation via Symbolic Canvassing

한승헌, 이주용 (UNIST)

1. Introduction

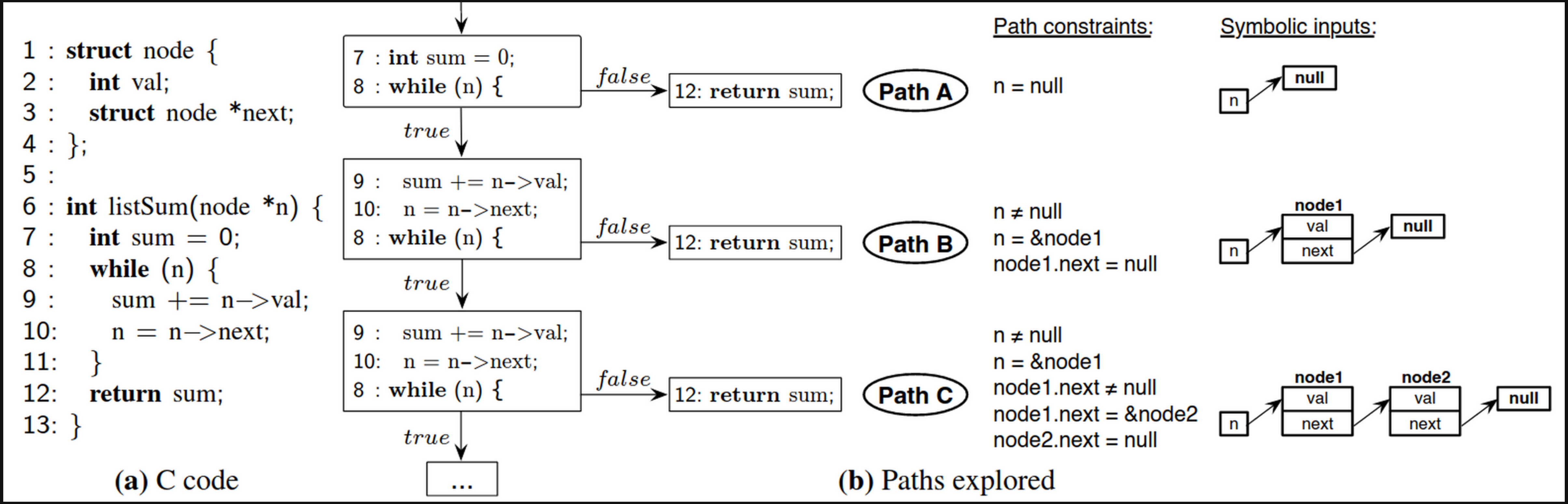
- Automated program repair tool generates a lot of patches
- Patch validation is important
    - No crash
    - No regression error

2. Symbolic Memory Manipulation

2.1. Dynamic analysis

- Run the code until crash location
- Snapshot at target function entry
- Analyze snapshotted memory state as a graph
- Select candidate nodes from accessed pointers and memory

2.2. K-Bounded Lazy Initialization



```
1 : struct node {
2 :   int val;
3 :   struct node *next;
4 : };
5 :
6 : int listSum(node *n) {
7 :   int sum = 0;
8 :   while (n) {
9 :     sum += n->val;
10:     n = n->next;
11:   }
12:   return sum;
13: }
```

(a) C code

(b) Paths explored

3. Validation

3.1. Patches

- All patches are instrumented as meta program
- Can apply same state to multiple patches
- Fork state at patch selection

3.2. Run with Same State

- Generate crashing memory state
- Record manipulated memory and constraints to the shadow memory
    - Expanded memory by lazy initialization
    - Constraints of each path
- Apply the shadow memory and constraints to the other patches
- Rerun the patched program with crashing memory state

4. Filter Out Incorrect Patches

- Given each memory state and constraints, classify them with crash or non-crash
- Determine correct patch