Informed Patch Selection via Automatic Decision Tree Construction

Nguyễn Gia Phong Jooyong Yi

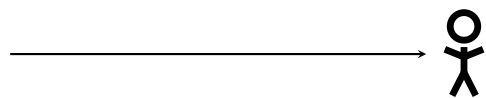
UNIST

2023-07-06

Motivation

Problem: APR tools give developer patch pool to choose from without explanation due to insufficient specification

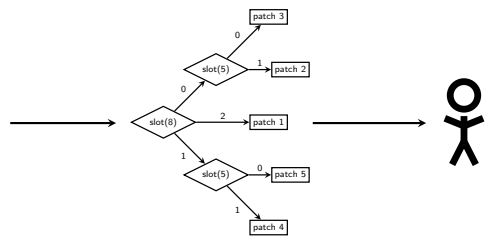
Patch	Rank
3	1
5	2
2	3
1	4
4	5



Motivation

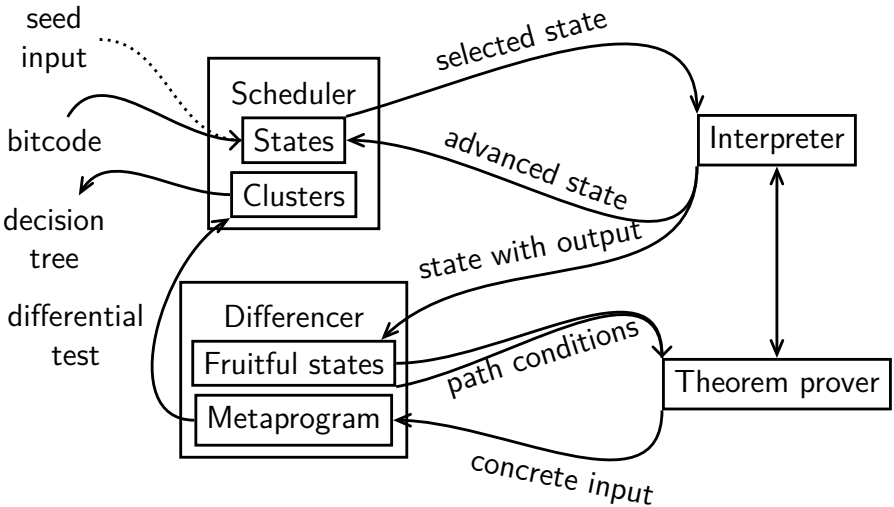
Problem: APR tools give developer patch pool to choose from without explanation due to insufficient specification

Patch	Rank
3	1
5	2
2	3
1	4
4	5



Solution: Generate decision tree of input–output pairs

Approach: Symbolic Execution



Challenges

- Troubling applicability of output value selection
- Inherent low scalability of symbolic execution
  - Even for common libraries, e.g. `printf`, `atoi`
- Symbolic input generation overhead
- Expensive exit state splitting $\Longrightarrow$ Caching
- SMT solver's low horizontal scalability: buffer size limit

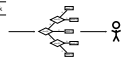
Informed Patch Selection via Automatic Decision Tree Construction

Nguyễn Gia Phong, Jooyong Yi
Lab of Software, UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Introduction

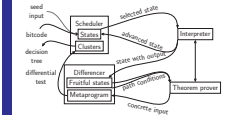
- Developers need to pick from pool of patch candidates
  - ▷ Large pool generated from automated program repair tools
  - ▷ Patch correctness determination not automatable due to incomplete specification
  - ▷ Probabilistic ranking provided instead of explanation
- Our tool generates decision tree based on input/output variables obtained from symbolic execution

Patch	Rank
3	1
5	2
2	3
1	4
4	5



Approach

Using symbolic execution:
- SMT solver can generate concrete program input.
- Existing approaches can already distinguish pair of program revisions.
- Scale for many revisions!



Output Selection

- Manual annotations by developer? Nontrivial!
- Symbolic buffer? Size limit!
- Propagated values from inputs? Path condition complexity!
  - ▷ Focus on function interface
  - ▷ Local to patch locations

All of the above! ⟹ "Fruitful" states!

Input Generation

- Name each output value occurrence
- (SMT) Solve for their difference

```
assert (f in!x out!a!a)
assert (f in!x out!a!b)
assert (g in!y out!a!a)
assert (h in!y out!a!b)
assert (distinct out!a!a out!a!b)
```

Metaprogram Example

```
int slot(int n) {
  int __choose1 = _choose(1);
  if      (__choose1 == 0) { /* original, noop */ }
  else if (__choose1 == 1) { __patch(1); n += 1;  }
  else if (__choose1 == 2) { __patch(2); n += 5;  }
  else if (__choose1 == 3) { __patch(3); n += 6;  }
  else if (__choose1 == 4) { __patch(4); n += 9;  }
  else if (__choose1 == 5) { __patch(5); n += 12; }
  ...
```

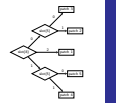
Scheduling Ideas

- Prioritise reaching patch location
- Favor corresponding output across revisions
- Prefer undistinguished revisions

Clustering

Differential test:
- Program input
- Output ↦ revision cluster

From clusters,
- Build decision tree: minimize height
- Feedback to scheduler



Evaluation

Preliminary:
- Buggy programs from IntroClass
- Patch generation: MSV

Work in progress:
- Real-world bugs: BugCspp7
- Patches: passing tests but semantically different
- Criteria: decision tree depth, efficiency

Challenges

- Troubling applicability of output value selection
- Inherent low scalability of symbolic execution
  - ▷ Even for common library, e.g. `printf`, `atoi`
- Symbolic input generation overhead
- Expensive exit state splitting ⟹ Caching
- SMT solver's low horizontal scalability: buffer size limit

Conclusion

- Interactive patch validation by example I/O
- I/O examples generated with symbolic & concrete execution
- Multiple scheduling heuristics possible
- Scaling challenges to be resolved

mcsinyx@unist.ac.kr

Please visit our poster for in-depth techniques and discussions!