What Are We Going To Do About Libraries? A Work in Non-Progress Talk

Martin Nyx Brain

University of Oxford

September 18, 2018
A Work in Non-Progress Talk

Don’t say I didn’t warn you...

- No answers; only problems.
- No results; only opinions.
“The Library Problem”

Parts of the program are not available or desirable to analyse

Because…

- Source unavailable
- External functionality
- Out of scope
- Platform independence
- Unspecified / imp. def.
- Too complex
- Program not finished
- This *is* the library
The Pyramid Model of Verification

- Automatic
- No Missed Bugs
- No False Alarms
The Pyramid Model of Verification

- Over-approximate: No Missed Bugs
- Automatic: No False Alarms
- Under-approximate: Human-assisted

Techniques:
- Abstract Interpretation
- Static Analysis
- Bug Patterns
- Model Checking
- Testing & Symbolic Execution
- Functional Verification
- Deductive Verification
The Pyramid Model of Verification

- **Over-approximate**:
  - Static Analysis
  - Abstract Interpretation

- **Automatic**:
  - Bug Patterns

- **Under-approximate**:
  - Testing & Symbolic Execution
  - Model Checking

- **Human-assisted**
  - Deductive Verification
  - Functional Verification

- **No Missed Bugs**
- **No False Alarms**
The Over-approximate Solution: Just Over-approximate

Non-det / Havoc
+ Simple in principle
- But what about...
- Correct OR precise

Bug Patterns

Testing & Symbolic Execution

Under-approximate

Model Checking

Automatic

No Missed Bugs

No False Alarms

Deductive Verification

Human-assisted

Functional Verification
size_t f00(void*, size_t, size_t, struct s *)
size_t fread(void*, size_t, size_t, FILE *)
The Under-approximate Solution: “Concolic”

- Non-det / Havoc
  + Simple in principle
  - But what about...
  - Correct OR precise

- Concolic
  + Works reasonably
  - If you can run the binary...
  - Fully stateful

- Automatic

- Human-assisted

Deductive Verification

Functional Verification
The Under-approximate Solution: "Concolic"

```c
ssize_t f01(int, const void*, size_t, int,
            const struct t*, size_t)
```
The Under-approximate Solution : “Concolic”

ssize_t sendto(int, const void*, size_t, int, const struct sockaddr*, socklen_t)
The Human-assisted Solution: Write Models

Non-det / Havoc
+ Simple in principle
- But what about...
- Correct OR precise

Concolic
+ Works reasonably
- If you can run the binary...
- Fully stateful

Automatic

Bug Patterns

No Missed Bugs

No False Alarms

Model
+ Use solver well
- Assuming docs are right...
- Validation

Deductive Verification
Functional Verification
void * realloc(void *ptr, size_t size)

Should we model...
  • When is size too much?
  • Return NULL?
  • Return NULL is sticky?
  • Alignment of result?
  • When does it return ptr?
  • errno set?
Isn’t this what game semantics is supposed to fix?
Possible Approaches

1. Isn’t this what game semantics is supposed to fix?
2. Lattice-based (formula) abstraction refinement

HAVOC!

- Havoc Global
- Term.
- Non-Term
- Havoc Args

Pure function

return 0;
Possible Approaches

1. Isn’t this what game semantics is supposed to fix?
2. Lattice-based (formula) abstraction refinement
3. What is “the answer” anyway?
Possible Approaches

1. Isn't this what game semantics is supposed to fix?
2. Lattice-based (formula) abstraction refinement
3. What is “the answer” anyway?
4. Opaque handles → automata?

FILE *fopen(const char *pathname, const char *mode);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
nsize_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
int feof(FILE *stream);
int ferror(FILE *stream);
int fclose(FILE *stream);
Possible Approaches

1. Isn’t this what game semantics is supposed to fix?
2. Lattice-based (formula) abstraction refinement
3. What is “the answer” anyway?
4. Opaque handles \(\rightarrow\) automata?
5. The spec is in the caller!

```c
struct dirent *d = readdir(root);
    do {
        struct dirent *d = readdir(tmp);
        if (strcmp(d->d_name, "vmlinuz") == 0) {
            ...}
    } else {
        perror("Directory empty");
        return errno;
    }
```

Possible Approaches

1. Isn’t this what game semantics is supposed to fix?
2. Lattice-based (formula) abstraction refinement
3. What is “the answer” anyway?
4. Opaque handles $\rightarrow$ automata?
5. The spec is in the caller!
6. Is modular symbolic execution impossible? Prove it!

Assuming independence is an (the only?) over-approximation...
Conclusions

1. The library problem is the pressing problem for practical application of verification tools (that can be solved by theoretical advances).
2. Current approaches are not practical / cost-effective.
3. Your solution here?
Conclusions

1. The library problem is the pressing problem for practical application of verification tools (that can be solved by theoretical advances).
2. Current approaches are not practical / cost-effective.
3. Your solution here?

Thank you for your time and attention.

Made using only Free Software