Verified Static Analysis of Low-Level Languages

Vincent Laporte

Advisors: Sandrine Blazy and David Pichardie

Université Rennes 1 – Celtique team

ÉJCP — 2013-05-25

Motivation

Possible outcomes when running a program

- result

Motivation

Possible outcomes when running a program

- result
- error message

Motivation

Possible outcomes when running a program

- result
- error message
- segmentation fault

Motivation

Possible outcomes when running a program

- result
- error message
- segmentation fault
- machine gets compromized

Motivation

Possible outcomes when running a program

- result
- error message
- segmentation fault
- machine gets compromized
- machine catches fire

Motivation

Possible outcomes when running a program

- result
- error message
- segmentation fault
- machine gets compromized
- machine catches fire

Problem

Can we decide, without running the program whether the behavior will be safe or not?

Let's use static analysis.

Motivation (cont.)

Possible outcomes when running a static analyzer

- "don't know" $\rightarrow$ seek the bug

Motivation (cont.)

Possible outcomes when running a static analyzer

- "don't know" $\rightarrow$ seek the bug
- "safe" $\rightarrow$ run the program $\rightarrow$ witness a safe behavior

Motivation (cont.)

Possible outcomes when running a static analyzer

- "don't know" $\rightarrow$ seek the bug
- "safe" $\rightarrow$ run the program $\rightarrow$ witness a safe behavior
- "safe" $\rightarrow$ run the program $\rightarrow$ witness an *unsafe* behavior

Motivation (cont.)

Possible outcomes when running a static analyzer

- "don't know" $\rightarrow$ seek the bug
- "safe" $\rightarrow$ run the program $\rightarrow$ witness a safe behavior
- "safe" $\rightarrow$ run the program $\rightarrow$ witness an *unsafe* behavior

Problem

How to trust a static analyzer?

Let's formally prove its correctness.

What is “safety”?

C Programs Have “Undefined Behaviors”

- out-of-bounds array access
- signed integer overflow
- null pointer dereference
- read from not initialized memory
- ...

What is “safety”?

C Programs Have “Undefined Behaviors”

- out-of-bounds array access
- signed integer overflow
- null pointer dereference
- read from not initialized memory
- ...

Safe Binary Program?

Execution stays within a given (code) segment.

Requires the ability to predict all jump targets.

What is “safety”?

C Programs Have “Undefined Behaviors”

- out-of-bounds array access
- signed integer overflow
- null pointer dereference
- read from not initialized memory
- ...

Safe Binary Program?

Execution stays within a given (code) segment.

Requires the ability to predict all jump targets.

Both cases require *value* analysis

Architecture of a Static Analyzer

Programs

- Abstract Interpreter
- Numerical Abstract Domains
- Abstract Memory Model

Proofs

- Semantics of the analyzed language (CompCert)
- Prove the abstract domains w.r.t. machine integers/floats
- Prove the memory model w.r.t. the language one
- Don't prove the interpreter (too hard): program and prove a validator instead

Conclusion

So Far

Static analyzers of

- an intermediate language of CompCert
- a toy binary language

Future Work

- Improve precision of the memory model
- Handle realistic binary (x86)