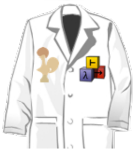
Introduction to Verification of System Software

Karl Palmskog
palmskog@kth.se — https://setoid.com
Researcher, KTH Royal Institute of Technology, Sweden



System Software

"[S]oftware designed to provide a platform for other software."

Examples

- compilers
- operating system kernels
- file systems

Properties

- mostly well understood
- regularly evaluated by researchers
- important to businesses, society, ...

Problem: Trustworthiness of System Software

- programmers and users rely on interfaces
- crashes and failures affect many clients
- "building blocks" of important services
- how to make guarantees about functionality & security?

Program Bugs, Defects, Infections, and Failures

From Zeller's *Why Programs Fail*:

Defect incorrect program code (code bug)

Infection incorrect program state (state bug)

Failure observable incorrect program behavior (behavior bug)

Methods to Avoid and Remove Bugs

Code type checking, static analysis, ...

Runtime debugging, testing, monitoring, ...

The Limits of Testing

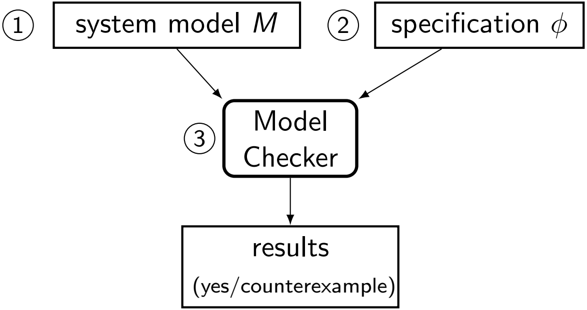
"Testing shows the presence, not the absence of bugs."

—E.W. Dijkstra, NATO SE Conference, 1969

Two Approaches To Program Verification

1. show absence of infections (state bugs)
2. show absence of defects (code bugs)

Classic Method: (Software) Model Checking



Classic Method: Deductive Verification

```
/*@ requires
  @   a != null && 0 <= i < a.length && 0 <= j < a.length;
  @ assigns
  @   a[i], a[j];
  @ ensures
  @   swapped{Old,Here}(a, i, j);
  @*/
public static void swap(int[] a, int i, int j) {
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}
```

Problem 1: Surprises

Surprise property or behavior of a program that can't be classified as feature/problem, for lack of specification

Problem 2: Code Level vs. Algorithm/Design Level

- benefits of verification of algorithms/design documented ...
- ... but guarantees do not extend to systems (machine code)

Algorithm/Design Level

```
binarySearch(a, key):
  low := 0;
  high := size(a) − 1;
  while (low <= high) {
    mid := (low + high) / 2;
    midVal := a[mid];
    if (midVal < key)
      low := mid + 1;
    else if (midVal > key)
      high := mid − 1;
    else
      return mid;
  }
  return −low − 1;
```

Code-Level Specification

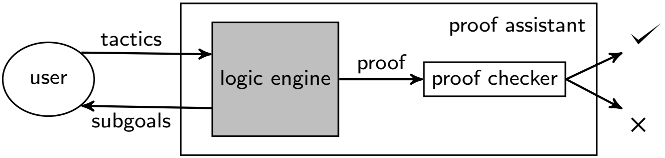
```
/*@ requires a != null && sorted(a, 0, a.length - 1);
  @
  @ behavior has_key:
  @ assumes \exists integer i;
  @  0 <= i <= a.length - 1 && a[i] == key;
  @ ensures a[\result] == key;
  @
  @ behavior has_not_key:
  @ assumes \forall integer i;
  @  0 <= i <= a.length - 1 ==> a[i] != key;
  @ ensures insertion_point(-\result - 1, key, a);
  @*/
public static int binarySearch(int[] a, int key)
```

Verified Code

```
public static int binarySearch(int[] a, int key) {
  int low = 0;
  int high = a.length - 1;
  while (low <= high) {
    int mid = low + ((high - low) / 2);
    int midVal = a[mid];
    if (midVal < key) {
      low = mid + 1;
    } else if (midVal > key) {
      high = mid - 1;
    } else {
      return mid;
    }
  }
  return -low - 1;
}
```

Verification Using Proof Assistants

1. encode definitions/programs in (higher-order) formalism
2. prove propositions **interactively** using powerful **tactics**
3. check soundness of every low-level step



examples: Coq, HOL4, HOL Light, Isabelle/HOL, Lean, Nuprl, ...

Some Large-Scale Verification Projects

Project	**Year**	**Assistant**	**LOC**
4-Color Theorem	2005	Coq	60k
Odd Order Theorem	2012	Coq	150k
Kepler Conjecture	2015	HOL Light	500k
CompCert	2009	Coq	40k
seL4	2009	Isabelle/HOL	200k
Verdi Raft	2016	Coq	50k

Effort in Large-Scale Projects

Project	**Domain**	**Effort**
CompCert	C compiler	$>$ 8 person years
seL4	OS kernel	$>$ 22 person years
Verdi Raft	distributed systems	$>$ 2 person years

A Verified Software Development Workflow (in Coq)

1. Write purely functional program
2. Write specification and prove program correct
3. Extract program to practical language (OCaml, Haskell, ...)
4. Link extracted program to libraries for I/O, communication, ...

```
Fixpoint alternate l1 l2 :=
match l1 with
| []  ⇒ l2
| h1 :: t1 ⇒
 match l2 with
 | [] ⇒ h1 :: t1
 | h2 :: t2 ⇒
   h1 :: h2 :: alternate t1 t2
 end
end.
```

1. Coq program

```
Inductive alt :=
| alt_nil : forall l,
  alt [] l l
| alt_step : forall a l t1 t2,
  alt l t1 t2 →
  alt (a :: t1) l (a :: t2).

Lemma alt_alternate :
  forall l1 l2 l3,
  alt l1 l2 l3 →
  alternate l1 l2 = l3.
Proof.
(* omitted proof script ... *)
Qed.
```

2. Coq spec/proof

```
let rec alternate l1 l2 =
 match l1 with
 | [] → l2
 | h1 :: t1 →
  (match l2 with
   | [] → h1 :: t1
   | h2 :: t2 →
     h1 :: (h2 ::
     (alternate t1 t2)))
```

3. OCaml program

Case Study: CompCert

- (almost) end-to-end verified compiler for C subset in Coq
- compiled code performance comparable to `gcc -O1`
- target: PowerPC, ARM, and x86(-64) assembly
- sold commercially
- applied in embedded systems development

Unsupported by CompCert

- unstructured `switch` (e.g., "Duff's device")
- `long double` type
- variable-length arrays

CompCert Correctness: Semantic Preservation

```
Theorem transf_c_program_correct:
 forall (p: Csyntax.program) (tp: Asm.program) (b: behavior),
   transf_c_program p = OK tp →
   program_behaves (Asm.semantics tp) b →
   exists b', program_behaves (Csem.semantics p) b'
     ∧ behavior_improves b' b.
```

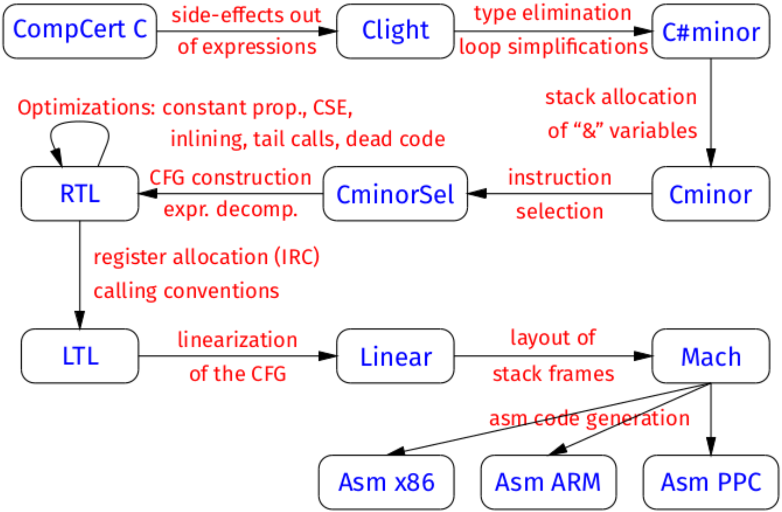
Leveraging CompCert for Program Verification

How to obtain guarantees about specific C programs?

1. encode C program in Coq using `clightgen`
2. specify properties of program in Coq
3. prove that program satisfies specification in Coq
4. guarantees hold for CompCert-generated assembly

`https://vst.cs.princeton.edu`

CompCert Passes



CompCert's Trusted Computing Base

- Coq's proof checker
- C semantics reflects meaning of C99 specification
- OCaml extraction preserves meaning of Coq's language
- OCaml compiler, runtime, libraries, OS, hardware

Earlier version of CompCert tested differentially against gcc and clang; no wrong-code errors found.

Try out CompCert (for Research Purposes)

- https://compcert.inria.fr
- easiest to install via OPAM: https://opam.ocaml.org

```
opam repo add coq-rl https://coq.inria.fr/opam/released
opam install coq-compcert

ccomp -c src1.c
ccomp -c src2.c
ccomp -o exec src1.o src2.o
```