# Proving C program correct using C light operational semantics

# Outline

1. Formal verification - quick intro (high-level)
2. Coq mini intro
3. Approach
   - Particular approach we consider: reasoning about C programs in Coq
   - Base PL concepts mini intro: syntax, AST, semantics.
4. Toy example: strlen Informal specification (man page)
   - Formal specification of strlen (relational)
   - Simple implementation in C
   - From C program to AST using clightgen
   - Semantics of C program semantics and its equivalence to specification
   - Undefined behaviours in C and guarding against them
5. Conclusions

CompCert example

# Coq intro

Explain what was done before: disadvantages and advantages of purely functional approach (Illya)

What we try now and why.

- reason about the actual implementation
- parse C code into an abstract syntax tree using C light generator of CompCert (not verified)
- reason about the C light program using operational semantics

# C light syntax

types

# C light semantics

Operational semantics: bigstep

# Informal spec

... DESCRIPTION
The strlen() function calculates the length of the string pointed to by s, excluding the terminating null byte.

RETURN VALUE
The strlen() function returns the number of bytes in the string pointed to by s.

CONFORMING TO
POSIX.1-2001, POSIX.1-2008, C89, C99, C11, SVr4, 4.3BSD.

To formalize the spec we need a formal model of C integers, pointers and memory model

# Int and Pointer offset types

Formalizations of machine integers modulo $2^N$ defined as a module type in CompCert `lib/Integers.v`.

A machine integer (type int) is represented as a Coq arbitrary-precision integer (type Z) plus a proof that it is in the range 0 (included) to modulus (excluded).

```
Record int: Type :=
mkint { intval: Z; intrange: −1 < intval < modulus }.
```

8, 32, 64-bit integers are supported, as well as 32 and 64-bit pointer offsets

# Memory model

defined in CompCert `common/Memory.v`

a type [mem] of memory states, the following 4 basic operations over memory states, and their properties:

load : read a memory chunk at a given address;

store : store a memory chunk at a given address;

alloc : allocate a fresh memory block;

free : invalidate a memory block.

# Formal spec

```
Inductive strlen (m : mem) (b : block) (ofs : ptrofs) : int →
Prop :=
| LengthZero: load m [b,ofs] = Some 0 → strlen m b ofs 0
| LengthSucc: ∀ (n : int) (c : char),
    strlen m b ofs + 1 n →
    load m [b,ofs] = Vint c →
    c <> Int.zero →
    strlen m b ofs n + 1.
```

# From C program to AST using clightgen

```
#include <stddef.h>

size_t strlen(const unsigned char *s)
{
   size_t i = 0;

   while(*s++)
        i++;

   return i;
}
```

# C light AST (loop of strlen)

```
Definition f_strlen_loop := {|
fn_params := ((_s, (tptr tuchar)) :: nil);
fn_temps := ((_i, tuint) :: (_t1, (tptr tuchar)) :: (_t2, tuchar) :
fn_body :=
(Sloop
(Ssequence
(Ssequence
(Ssequence
  (Sset _t1 (Etempvar _s (tptr tuchar)))
  (Sset _s
    (Ebinop Oadd (Etempvar _t1 (tptr tuchar))
      (Econst_int (Int.repr 1) tint) (tptr tuchar))))
(Ssequence
  (Sset _t2 (Ederef (Etempvar _t1 (tptr tuchar)) tuchar))
  (Sifthenelse (Etempvar _t2 tuchar) Sskip Sbreak)))
(Sset _i
(Ebinop Oadd (Etempvar _i tuint) (Econst_int (Int.repr 1)
 tint)
  tuint)))
Sskip) |}.
```

Formal verification - quick intro

Approach

Toy example: length of a C string