Approaches to C in seL4, Sydney and Isabelle

Thomas Sewell

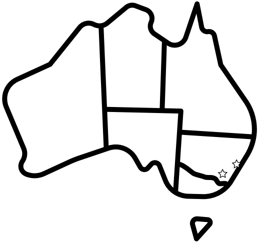
Cambridge

August 3, 2022

This is about C verification approaches from the L4.verified project.

The central feature is the “C-to-Isabelle“ parser

- or L4.verified C parser
- or NICTA C parser
- or Tuch/Norrish C semantics



C Proof Chain

Obviously, the parser converts C $\rightarrow$ Isabelle.

- $\rightarrow$ a *deeply encoded* syntax.
- annoying limitation: only one C file.

Then, prove something about it:

- e.g. with a Hoare/Floyd VCG on deep encoding
- via simulation/refinement to a monadic spec
  - L4.verified/seL4 proof approach
- via auto-refinement to a generated shallow spec
  - Auto-Corres approach
  - Cogent project
- via translation validation down to binary code

Example 1

For example ...

```
struct list_node *
list_rev_and_inc (struct list_node *p) {
  struct list_node *rev, *tmp;

  for (rev = NULL; p; ) {
    p->v ++;
    tmp = p->next;
    p->next = rev;
    rev = p;
    p = tmp;
  }

  return rev;
}
```

Example 2

```
demo_global_addresses.list_rev_and_inc_body ≡
TRY
 lvar_nondet_init rev_' rev_'_update;;
 lvar_nondet_init tmp_' tmp_'_update;;
 ´rev :== PTR_COERCE(unit → list_node_C) (PTR(unit) (SCAST(32 signed → 64) 0));;
 WHILE ´p ≠ NULL DO
   Guard C_Guard {|c_guard ´p|}
    (Guard SignedArithmetic
      {|- 2147483648 ≤ sint (h_val (hrs_mem ´t_hrs) (PTR(32 signed word) &(´p→["v_C"]))) + sint 1 ∧
       sint (h_val (hrs_mem ´t_hrs) (PTR(32 signed word) &(´p→["v_C"]))) + sint 1 ≤ 2147483647|}
      (Guard C_Guard {|c_guard ´p|}
        (´globals :==
           t_hrs_'_update
            (hrs_mem_update
              (heap_update (PTR(32 signed word) &(´p→["v_C"]))
                (h_val (hrs_mem ´t_hrs) (PTR(32 signed word) &(´p→["v_C"])) + 1))))));;
   Guard C_Guard {|c_guard ´p|}
    (´tmp :== h_val (hrs_mem ´t_hrs) (PTR(list_node_C ptr) &(´p→["next_C"])));;
   Guard C_Guard {|c_guard ´p|}
    (´globals :==
       t_hrs_'_update (hrs_mem_update (heap_update (PTR(list_node_C ptr) &(´p→["next_C"])) ´rev)));;
   ´rev :== ´p;;
   ´p :== ´tmp;;
   SKIP
 OD;;
 creturn global_exn_var_'_update ret__ptr_to_struct_list_node_C_'_update rev_';;
 Guard DontReach {} SKIP
CATCH SKIP
END
```

C Semantics

The C semantics $\equiv$ Simpl
+ "Tetris" memory model
+ parser elaboration.

Simpl: Simple Imperative Language

- by Norbert Schirmer & Verisoft project
- deeply embedded statement syntax (shallow expressions)
- big & small step semantics and a Floyd/Hoare VCG

```
type_synonym 's bexp = "'s set"
type_synonym 's assn = "'s set"

datatype (dead 's, 'p, 'f) com =
    Skip
  | Basic "'s ⇒ 's"
  | Spec "('s × 's) set"
  | Seq "('s ,'p, 'f) com" "('s,'p, 'f) com"
  | Cond "'s bexp" "('s,'p,'f) com"  "('s,'p,'f) com"
  | While "'s bexp" "('s,'p,'f) com"
  | Call "'p"
  | DynCom "'s ⇒ ('s,'p,'f) com"
  | Guard "'f" "'s bexp" "('s,'p,'f) com"
  | Throw
  | Catch "('s,'p,'f) com" "('s,'p,'f) com"
```

Tuch Memory Model

The memory model comes from Harvey Tuch's PhD work.

<table>
<tr><td colspan="12">struct x_data</td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="4">char[4]</td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="4">int *</td><td>char</td><td>char</td><td>char</td><td>char</td><td colspan="4">long</td></tr>
</table>

0xb1	0x34	0x52	0xe3	0x00	0x00	0xb4	0xac	0x00	0x00	0xff	0xac

The heap representation includes:

- a 1-dimensional array of bytes in memory.
- a 2-dimensional "tetris" model of pointer/type validity.

Memory Model Ops

This model directly provides

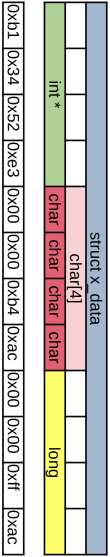
- $p\_valid :: \alpha\ ptr \to heap \to bool$
- $h\_acc :: heap \to \alpha\ ptr \to \alpha$
- $h\_upd :: \alpha\ ptr \to \alpha \to heap \to heap$

It's standard to reason about lifted heaps

- $h\_lift :: heap \to \alpha\ ptr \to \alpha\ option$
- inter-type aliasing is handled automatically
- L4.verified does this

What about intra-type aliasing?

- Tuch's PhD develops a separation logic



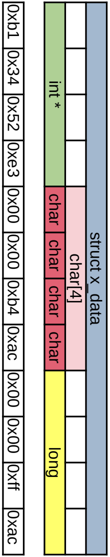
Memory Model Comparison

I've gone into detail here because this memory model is different to some comparable work.

There's no direct equivalent of *provenance* in this model.

However, this is a typed memory model, with a detailed notion of pointer validity.
It is *not* a "portable assembler" model.

This model cannot be exactly sound against the standard (as written). However, the translation validation works, for nontrivial examples.



C Parser Elaboration

The actual parser elaborates C

- creates the "ugly" explicit representation
- builds on Michael Norrish's work on C formalisation here

Some curios:

- Local variables cannot be addressed in C
- Local variables become "normal" stateful variables in Simpl
- Global variables that are not addressed also become variables

Some of this aims to simplify hand reasoning.

Other Limitations

There are a few other limitations.

Most notably, Simpl is intrinsically single-threaded.
There is a proposed replacement, Complx, which addresses some of this.
I personally am no longer up to date with these developments.

There's other work I don't know, e.g. Isabelle/C by Frédéric Tuong and Burkhart Wolff.

AutoCorres

AutoCorres, from David Greenaway's PhD project, automatically *constructs* a monadic program, shallowly embedded, that abstracts the Simpl program.

This might be clearest via a demo . . .

Demo etc

Hopefully there's plenty of time for a demo.