HOL Developed and HOL Used: Interconnected Stories of Real-World Applications

Michael Norrish

July 2018

Cambridge Context in 1994—People

Recently finished/departed PhDs:

- Richard Boulton (efficient theorem-proving)
- Victor Carreno (real-time systems)
- Jim Grundy (refinement, window inference)
- Monica Nesi (process calculi)
- John Van Tassel (VHDL)
- John Harrison (real numbers, analysis)


... dreaming spires
FLoC 20
HOL developed and HOL used

My Cohort

Fellow PhD students:

- Mark Staples (refinement calculus in Isabelle/ZF)
- Don Syme (theorem-proving for operational semantics)

Starting a `cl.cam.ac.uk` PhD in 1994

Very flexible (more so than modern PhDs?)

- Don Syme changed topic completely after a year

Simultaneously gentle, and “sink-or-swim”:

- Mike suggested C as PhD topic as I got to grips with HOL
- I had a lot to learn

Cambridge Context in 1994—HOL

Powerful system moving beyond hardware verification applications

General purpose tooling:

- Inductive definition package
- Data type definition package
- Arithmetic decision procedures

Theorem-proving for operational semantics builds on all of these

My PhD

Almost entirely as a HOL user:

- mechanised an operational semantics for C (as *per* 1989 standard)
- proved some meta-theorems

Very much in vein of contemporary work applying HOL to operational semantics.

Examined by Tom Melham and Andy Gordon.

JRF and post-PhD Freedom

Won a Junior Research Fellowship at St. Catharine's College

Could not muster much enthusiasm for C

HOL's Continuing Development

Large ESPRIT project, "Prosper" (led by one Tom Melham) employs HOL's then principal developer, **Konrad Slind** in Cambridge.

He and Ken Friis Larsen work on port from SML/NJ to Moscow ML

- Result is `hol98`; first release *Athabasca-1*

I attend various Prosper meetings and develop "opinions".

Parsing, Numbers, ...

Konrad's openness to contributions lets me

- add a record type definition principle;
- completely rework HOL's parsing and pretty-printing infrastructure;
- change the representation of numerals (from "unary" to binary scheme);
- name the relevant release series *Taupo*

Mike and HOL

Combining systems, continues to attack "hardware"-ish problems:

- With **Ken Friis Larsen**, integrates BDD package to allow CTL model checking (and other applications)
- Hardware description languages with Daryl Stewart
- First moves on ACL2 connections with Mark Staples
- (Later) Hardware synthesis with Juliano Iyoda

Mike and HOL: ARM

In 2000, Mike hired **Anthony Fox** on an ARM verification project

- joint work with Graham Birtwistle (Leeds), and support from ARM

This research project has been incredibly fruitful:

- Theorem-proving at scale ...
- ... leading to numerous real-world applications

Evaluation in the Logic

During visit from France, Coq developer **Bruno Barras** implements work-horse `CBV_CONV` (later just "`EVAL`").

Critcal tool for in-logic validation/execution of models

- Given time and expertise, custom tools could do sophisticated things
- Being able to type `EVAL "f arg"` to explore behaviours is an immense productivity boost

More Operational Semantics

HOL's definitional tools scaled (scale) beautifully.

From tutorial examples (combinatory logic):

```
val (redn_rules, _, _) = Hol_reln `
    (!x y f. x --> y   ==>    f # x --> f # y) /\
    (!f g x. f --> g   ==>    f # x --> g # x) /\
    (!x y.   K # x # y --> x) /\
    (!f g x. S # f # g # x --> (f # x) # (g # x))`;
```

More Operational Semantics

HOL's definitional tools scaled (scale) beautifully.

To my C semantics
(one of many rules about assignment):

```
{hypotheses = [],
 side_conditions = [
     ``convert_val (strmap s) (v0,t0) (v,lhs_t) /\
       (ok_refs = \x. x IN (se_affects (a, v)) => mb x | 0) /\
       (se' = ref_map_fupd (\rm. BAG_DIFF rm ok_refs) se0) /\
       (se = add_se (a, v) se') /\ (resv = ECompVal v lhs_t)
                             \/
       (!v. ~convert_val (strmap s) (v0, t0) (v, lhs_t)) /\
       (resv = UndefinedExpr) /\ (se = se0)``
 ],
 (* ------------------------------------------------------------- *)
 conclusion = ``^mng (mExpr (Assign CAssign (LVal a lhs_t)
                                           (ECompVal v0 t0)
                                           mb)
                          se0) s (s, ^ev resv se)``},
```

More Operational Semantics

HOL's definitional tools scaled (scale) beautifully.

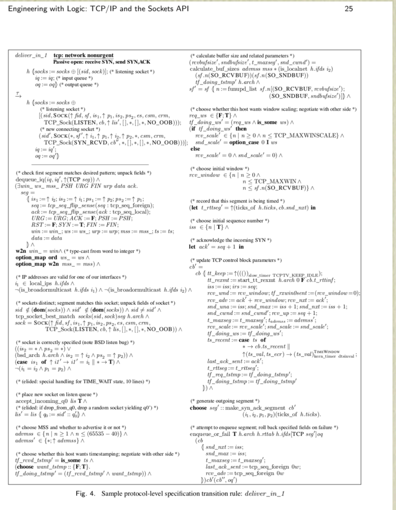
To ARM:

```
val EXEC_INST_def = Define`
  EXEC_INST (ARM_EX (ARM reg psr) ireg exc)
    (dabort_t:num option) data cp_interrupt =
    if ~(exc = software) then
      EXCEPTION (ARM reg psr) exc
    else
      let ic = DECODE_INST ireg
      and (nzcv,i,f,m) = DECODE_PSR (CPSR_READ psr)
      in
        if ~CONDITION_PASSED nzcv ireg then
          ARM (INC_PC reg) psr
        else let mode = DECODE_MODE m in
        if (ic = data_proc) \/ (ic = reg_shift) then
          DATA_PROCESSING (ARM reg psr) (CARRY nzcv) mode ireg
        else if ic = mla_mul then
          MLA_MUL (ARM reg psr) (CARRY nzcv) mode ireg
        else if ic = br then
          BRANCH (ARM reg psr) mode ireg
        else if (ic = ldr) \/ (ic = str) then
          (LDR_STR (ARM reg psr) (CARRY nzcv) mode
             (IS_SOME dabort_t) (HD data) ireg).state
```

More Operational Semantics

HOL's definitional tools scaled (scale) beautifully.

To TCP(?!):


Engineering with Logic: TCP/IP and the Sockets API 25

```
deliver_in_1   tcp: network nonurgent
               Passive open: receive SYN, send SYN,ACK

  h ⦇socks := socks ⊕ [(sid, sock)]; (* listening socket *)
    iq := iq; (* input queue *)
    oq := oq⦈ (* output queue *)

τ
→
  h ⦇socks := socks ⊕
      (* listening socket *)
     [(sid, Sock(↑ fid, sf, is1, ↑ p1, is2, ps2, es, csm, crm,
       TCP_Sock(LISTEN, cb, ↑ lis', [], ∗, [], ∗, NO_OOB)));
      (* new connecting socket *)
     (sid', Sock(∗, sf', ↑ i1, ↑ p1, ↑ i2, ↑ p2, ∗, csm, crm,
       TCP_Sock(SYN_RCVD, cb'', ∗, [], ∗, [], ∗, NO_OOB)))];
    iq := iq';
    oq := oq'⦈
─────────────────────────────────────────
(* check first segment matches desired pattern; unpack fields *)
dequeue_iq(iq, iq', ↑(TCP seg)) ∧
(∃win_ ws_ mss_ PSH URG FIN urp data ack.
  seg =
   ⦇ is1 := ↑ i2; is2 := ↑ i1; ps1 := ↑ p2; ps2 := ↑ p1;
     seq := tcp_seq_flip_sense(seq : tcp_seq_foreign);
     ack := tcp_seq_flip_sense(ack : tcp_seq_local);
     URG := URG; ACK := F; PSH := PSH;
     RST := F; SYN := T; FIN := FIN;
     win := win_; ws := ws_; urp := urp; mss := mss_; ts := ts;
     data := data
   ⦈ ∧
w2n win_ = win∧ (* type-cast from word to integer *)
option_map ord ws_ = ws ∧
option_map w2n mss_ = mss) ∧

(* IP addresses are valid for one of our interfaces *)
i1 ∈ local_ips h.ifds ∧
¬(is_broadormulticast h.ifds i1) ∧ ¬(is_broadormulticast h.ifds i2) ∧

(* sockets distinct; segment matches this socket; unpack fields of socket *)
sid ∉ (dom(socks)) ∧ sid' ∉ (dom(socks)) ∧ sid ≠ sid' ∧
tcp_socket_best_match socks(sid, sock)seg h.arch ∧
sock = Sock(↑ fid, sf, is1, ↑ p1, is2, ps2, es, csm, crm,
            TCP_Sock(LISTEN, cb, ↑ lis, [], ∗, [], ∗, NO_OOB)) ∧

(* socket is correctly specified (note BSD listen bug) *)
((is2 = ∗ ∧ ps2 = ∗) ∨
(bsd_arch h.arch ∧ is2 = ↑ i2 ∧ ps2 = ↑ p2)) ∧
(case is1 of ↑ i1' → i1' = i1 ‖ ∗ → T) ∧
¬(i1 = i2 ∧ p1 = p2) ∧

(* (elided: special handling for TIME_WAIT state, 10 lines) *)

(* place new socket on listen queue *)
accept_incoming_q0 lis T ∧
(* (elided: if drop_from_q0, drop a random socket yielding q0') *)
lis' = lis ⦇ q0 := sid' :: q0' ⦈ ∧

(* choose MSS and whether to advertise it or not *)
advmss ∈ {n | n ≥ 1 ∧ n ≤ (65535 − 40)} ∧
advmss' ∈ {∗; ↑ advmss} ∧

(* choose whether this host wants timestamping; negotiate with other side *)
tf_rcvd_tstmp' = is_some ts ∧
(choose want_tstmp :: {F; T}.
tf_doing_tstmp' = (tf_rcvd_tstmp' ∧ want_tstmp)) ∧

(* calculate buffer size and related parameters *)
(rcvbufsize', sndbufsize', t_maxseg', snd_cwnd') =
calculate_buf_sizes advmss mss ∗ (is_localnet h.ifds i2)
  (sf.n(SO_RCVBUF))(sf.n(SO_SNDBUF))
  tf_doing_tstmp' h.arch ∧
sf' = sf ⦇ n := funupd_list sf.n[(SO_RCVBUF, rcvbufsize');
                                 (SO_SNDBUF, sndbufsize')]⦈ ∧

(* choose whether this host wants window scaling; negotiate with other side *)
req_ws ∈ {F; T} ∧
tf_doing_ws' = (req_ws ∧ is_some ws) ∧
(if tf_doing_ws' then
  rcv_scale' ∈ {n | n ≥ 0 ∧ n ≤ TCP_MAXWINSCALE} ∧
  snd_scale' = option_case 0 I ws
else
  rcv_scale' = 0 ∧ snd_scale' = 0) ∧

(* choose initial window *)
rcv_window ∈ {n | n ≥ 0 ∧
                  n ≤ TCP_MAXWIN ∧
                  n ≤ sf.n(SO_RCVBUF)} ∧

(* record that this segment is being timed *)
(let t_rttseg' = ↑(ticks_of h.ticks, cb.snd_nxt) in

(* choose initial sequence number *)
iss ∈ {n | T} ∧

(* acknowledge the incoming SYN *)
let ack' = seq + 1 in

(* update TCP control block parameters *)
cb' =
  cb ⦇ tt_keep := ↑((())slow_timer TCPTV_KEEP_IDLE);
       tt_rexmt := start_tt_rexmt h.arch 0 F cb.t_rttinf;
       iss := iss; irs := seq;
       rcv_wnd := rcv_window; tf_rxwin0sent := (rcv_window = 0);
       rcv_adv := ack' + rcv_window; rcv_nxt := ack';
       snd_una := iss; snd_max := iss + 1; snd_nxt := iss + 1;
       snd_cwnd := snd_cwnd'; rcv_up := seq + 1;
       t_maxseg := t_maxseg'; t_advmss := advmss';
       rcv_scale := rcv_scale'; snd_scale := snd_scale';
       tf_doing_ws := tf_doing_ws';
       ts_recent := case ts of
                      ∗ → cb.ts_recent ‖
                      ↑(ts_val, ts_ecr) → (ts_val)^TimeWindow_kern_timer dtsinval;
       last_ack_sent := ack';
       t_rttseg := t_rttseg';
       tf_req_tstmp := tf_doing_tstmp';
       tf_doing_tstmp := tf_doing_tstmp'
     ⦈) ∧

(* generate outgoing segment *)
choose seg' :: make_syn_ack_segment cb'
                 (i1, i2, p1, p2)(ticks_of h.ticks).

(* attempt to enqueue segment; roll back specified fields on failure *)
enqueue_or_fail T h.arch h.rttab h.ifds[TCP seg']oq
  (cb
   ⦇ snd_nxt := iss;
     snd_max := iss;
     t_maxseg := t_maxseg';
     last_ack_sent := tcp_seq_foreign 0w;
     rcv_adv := tcp_seq_foreign 0w
   ⦈)cb'(cb'', oq')
```

Fig. 4. Sample protocol-level specification transition rule: *deliver_in_1*

Network Semantics

With **Peter Sewell** and Keith Wansbrough:

- Showed that HOL could handle large detailed semantics
  - first UDP and then TCP
  - both definitions, and generation of theorems in a novel style

- Developed custom tooling (the real HOL strength) to validate semantics against sniffed traces

TCP Work Driving HOL Development

Large terms, large theorems, large simplification sets...

Leading to:

- Another kernel implementation (more efficient with large numbers of bound variables)
  - suitably opaque & well-designed term API
- Dictionaries / trees in place of lists in various places

+ efficient evaluation...

Portability + Scalability = Better Tools

While a Cambridge post-doc, **Scott Owens** ports HOL to Poly/ML

- working with Sewell on hardware memory models
- fantastic speed-boost
- forces cleaner code
- allows powerful tools

Extending the HOL Diaspora

In 2003, I moved to Canberra.

HOL contributions came from

- Cambridge (Mike, students, postdocs)
- Oxford (Joe Hurd, Ashish Darbari)
- Australia (me and some students)
- USA (Konrad Slind, Peter Homeier, Joe Hurd)
- ...

A small, effective and harmonious developer community

Other Subsequent Work

Indirectly using C expertise:

- wrote "parser" tool to load seL4 C source code into Isabelle for verification project at NICTA (now Data61)
- HOL + ARM model allows for *post hoc* validation of this down to binary level

With Aditi Barthwal:

- formalisation of theory of context-free languages and parsing
    - later useful in CakeML

Still to Come

Yet more operational semantics:

- μVM project with Blackburn, Hosking and Moss

More HOL development:

- broader visibility (`github`) 99% a good thing
- responsiveness to demands of major applications (*i.e.*, mostly CakeML)
- learning lessons from Isabelle's more extensive engineering

Mike

- Had a massive influence on my research career
- An energising emphasis on combining rigour with real-world applications
- Built a system; more importantly built community around it