(Co)verifying a compiler and a prover: the CakeML project

Freek Wiedijk

Radboud University Nijmegen

2013 10 29
factorial in HOL4

val fact_def = Define ' 
  (fact 0 = 1) /
  (fact (SUC n) = (SUC n) * fact n)
  '"
program extraction in HOL4

val tEVAL = rhs o concl o EVAL;

val _ = set_trace "assumptions" 1;

val translate_fact = translate fact_def;

val fact_ast =
  (rand o rator o rhs o concl o definition) "fact_demo_decls_0";

val fact_string =
  tEVAL ("dec_to_sml_string ^fact_ast");

val _ = (print (fromHOLstring fact_string); print "\n");
Certificate theorem for fact:

\[
[]
\]
\[- \text{DeclAssum fact\_demo\_decls env} \Rightarrow \]
\[\text{Eval env (Var (Short "fact")) ((NUM --> NUM) fact)}\]
Definition of declaration list:

[]
|\- fact_demo_decls =
[Dletrec
   ["fact","v1",
     If (App (Opb Leq) (Var (Short "v1")) (Lit (IntLit 0)))
     (Lit (IntLit 1))
     (App (Opn Times)
       (App (Opn Plus)
         (Let "k" (... ... (... ... )))
         (... ... (... ... ) (Var (... ... )))))
     (Lit (IntLit 1)))
   (App Opapp (Var (Short "fact")))
   (Let "k"
     (App (... ... ) (Var (... ... )) (Lit (IntLit 1)))
     (If (... ... (... ... )) (Lit (IntLit 0))
      (Var (Short "k"))))]]
fun fact v1 = 
  (if (v1 <= 0)
   then 1
   else ((
     let val k = (v1 - 1)
     in
     (if (k < 0)
      then 0
      else k)
     end + 1) * (fact
     let val k = (v1 - 1)
     in
     (if (k < 0)
      then 0
      else k)
     end ))) ;
val translate_fact10 = hol2deep "fact 10";

val fact10_ast =
    (rand o rator o concl) translate_fact10;

val bare_bc_state = tEVAL "
    <|stack := [];
        code := PrintE++[Stop];
        pc := 0;
        refs := FEMPTY;
        handler := 0;
        clock := NONE;
        output := "";
        cons_names := [];
        inst_length := K 0 |>
        "";

val initial_bc_state_no_eval = tEVAL "
    install_code [] (SND (SND compile_primitives)) ^bare_bc_state "
    ";

compiling to bytecode inside HOL4
val fact10_bc_state = tEVAL "'
let (state1,_,code1) = 
    compile_top (initial_repl_fun_state.rcompiler_state)
    (Tdec ^fact_ast) in 
let bc_state1 = install_code (cpam state1) code1
    ^initial_bc_state_no_eval in 
let (state2,_,code2) = 
    compile_top state1 (Tdec (Dlet (Pvar "it") ^fact10_ast)) in 
let bc_state2 = install_code (cpam state2) code2 bc_state1 in 
bc_state2
'";

val fact10_bc = tEVAL "'(^fact10_bc_state).code'";

val fact10_bc_string = (print o fromHOLstring o tEVAL) "'
  FLAT (MAP (\inst. bc_inst_to_string inst ++ "$\n")
  (code_labels (K 0) ^fact10_bc))
'";
bytecode for ‘fact 10’

printC 'r'
printC 'a'
printC 'i'
printC 's'
printC 'e'
printC ''
print
printC '\n'
stop
pushPtr addr 0
pushExc
jump addr 374
load 2
ref
pops 1
load 1
store 3

... plus 1082 more bytecodes ...
running the bytecode outside HOL4

% /opt/src/vml/unverified/bytecode/cakeml-byte fact.bc
val + = <fn>
val - = <fn>
val * = <fn>
val div = <fn>
val mod = <fn>
val < = <fn>
val > = <fn>
val <= = <fn>
val >= = <fn>
val = = <fn>
val := = <fn>
val ~ = <fn>
val ! = <fn>
val ref = <fn>
val fact = <fn>
val it = 3628800
%
CakeML source grammar

\[ id ::= x | Mn.x \]

\[ cid ::= Cn | Mn.Cn \]

\[ t ::= \text{int} | \text{bool} | \text{unit} | \alpha | \text{id} | t \text{id} | (t,t*) \text{id} \]

\[ l ::= \mathbb{Z} | \text{true} | \text{false} | () | [] \]

\[ p ::= x | l | cid | cid \text{p} | \text{ref} \text{p} | _ | (p,p*) | [p,p*] \]

\[ e ::= l | id | cid | cid \text{e} | (e,e,e*) | [e,e]* \]

\[ e ::= \text{raise e} | \text{e handle p => e (| p => e)*} \]

\[ e ::= \text{fn x => e} | \text{e e} | ((e;)e) | \text{uop e} | \text{e op e} \]

\[ e ::= \text{if e then e else e} | \text{case e of p => e (| p => e)*} \]

\[ e ::= \text{let (ld;})* \text{ in (e;})* \text{ e end} \]

\[ ld ::= \text{val x = e} | \text{fun x y = e (and x y = e)*} \]

\[ uop ::= \text{ref} | ! | \sim \]

\[ op ::= = | := | + | - | * | \text{div} | \text{mod} | < | \leq | > | \geq | <> | :: \]

\[ e ::= \text{before} | \text{andalso} | \text{orelse} \]
CakeML source grammar (continued)

\[
c ::= \ Cn \mid Cn \ of \ t \\
\text{tyd} ::= \ tyn = c \ (\mid \ c)^* \\
\text{tyn} ::= \ (\alpha, (\alpha)^*) \ x \mid \alpha \ x \mid x \\
\text{d} ::= \ \text{datatype tyd} \ (\text{and tyd})^* \mid \text{val p = e} \\
\quad \mid \ \text{fun x y}^+ = e \ (\text{and x y}^+ = e)^* \\
\quad \mid \ \text{exception c} \\
\text{sig} ::= \ :> \ \text{sig} \ (\text{sl} \mid ;)^* \ \text{end} \\
\text{sl} ::= \ \text{val x : t} \mid \text{type tyn} \mid \text{datatype tyd} \ (\text{and tyd})^* \\
\text{top} ::= \ \text{structure Mn sig}^? = \text{struct} \ (d \mid ;)^* \ \text{end}; \mid d; \mid e;
\]