(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

```
Certificate theorem for fact:

 []
|- DeclAssum fact_demo_decls env ⇒
   Eval env (Var (Short "fact")) ((NUM --> NUM) fact)
```

certificate theorem (continued)

```
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")))))))]]
```

pretty-printed CakeML version of fact

```
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 ))) ;
```

compiling to bytecode inside HOL4

```
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 (continued)

```
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

$$
\begin{array}{lcl}
\mathit{id} & ::= & x \mid \mathit{Mn}\,.\,x \\
\mathit{cid} & ::= & \mathit{Cn} \mid \mathit{Mn}\,.\,\mathit{Cn} \\
t & ::= & \texttt{int} \mid \texttt{bool} \mid \texttt{unit} \mid \alpha \mid \mathit{id} \mid t\ \mathit{id} \mid (t(\texttt{,}t)^*)\,\mathit{id} \\
 & \mid & t\ \texttt{*}\ t \mid t\ \texttt{->}\ t \mid t\ \texttt{ref} \mid (t) \\
l & ::= & \mathbb{Z} \mid \texttt{true} \mid \texttt{false} \mid \texttt{()} \mid \texttt{[]} \\
p & ::= & x \mid l \mid \mathit{cid} \mid \mathit{cid}\ p \mid \texttt{ref}\ p \mid \texttt{\_} \mid (p(\texttt{,}p)^*) \mid [p\,(\texttt{,}p)^*] \\
 & \mid & p\ \texttt{::}\ p \\
e & ::= & l \mid \mathit{id} \mid \mathit{cid} \mid \mathit{cid}\ e \mid (e\texttt{,}e\,(\texttt{,}e)^*) \mid [e\,(\texttt{,}e)^*] \\
 & \mid & \texttt{raise}\ e \mid e\ \texttt{handle}\ p\ \texttt{=>}\ e\ (\texttt{|}\ p\ \texttt{=>}\ e)^* \\
 & \mid & \texttt{fn}\ x\ \texttt{=>}\ e \mid e\ e \mid ((e\ \texttt{;})^*\ e) \mid \mathit{uop}\ e \mid e\ \mathit{op}\ e \\
 & \mid & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{case}\ e\ \texttt{of}\ p\ \texttt{=>}\ e\ (\texttt{|}\ p\ \texttt{=>}\ e)^* \\
 & \mid & \texttt{let}\ (\mathit{ld}|\texttt{;})^*\ \texttt{in}\ (e\ \texttt{;})^*\ e\ \texttt{end} \\
\mathit{ld} & ::= & \texttt{val}\ x\ \texttt{=}\ e \mid \texttt{fun}\ x\ y^+\ \texttt{=}\ e\ (\texttt{and}\ x\ y^+\ \texttt{=}\ e)^* \\
\mathit{uop} & ::= & \texttt{ref} \mid \texttt{!} \mid \texttt{\~{}} \\
\mathit{op} & ::= & \texttt{=} \mid \texttt{:=} \mid \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{div} \mid \texttt{mod} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{<>} \mid \texttt{::} \\
 & \mid & \texttt{before} \mid \texttt{andalso} \mid \texttt{orelse}
\end{array}
$$

CakeML source grammar (continued)

$$
\begin{array}{lcl}
c & ::= & Cn \mid Cn\ \texttt{of}\ t \\
tyd & ::= & tyn\ \texttt{=}\ c\ (\mid c)^* \\
tyn & ::= & (\alpha(\texttt{,}\alpha)^*)\ x \mid \alpha\ x \mid x \\
d & ::= & \texttt{datatype}\ tyd\ (\texttt{and}\ tyd)^* \mid \texttt{val}\ p\ \texttt{=}\ e \\
 & \mid & \texttt{fun}\ x\ y^+\ \texttt{=}\ e\ (\texttt{and}\ x\ y^+\ \texttt{=}\ e)^* \\
 & \mid & \texttt{exception}\ c \\
sig & ::= & \texttt{:>}\ \texttt{sig}\ (sl\,|\texttt{;})^*\ \texttt{end} \\
sl & ::= & \texttt{val}\ x\ \texttt{:}\ t \mid \texttt{type}\ tyn \mid \texttt{datatype}\ tyd\ (\texttt{and}\ tyd)^* \\
top & ::= & \texttt{structure}\ Mn\ sig^?\ \texttt{=}\ \texttt{struct}\ (d\,|\texttt{;})^*\ \texttt{end;} \mid d\texttt{;} \mid e\texttt{;}
\end{array}
$$

CakeML project directories