

Interactive Theorem Proving and Program Verification

Lecture 10

Pablo Buiras and Karl Palmskog



Academic Year 2019/20, Period 3–4

Based on slides by Thomas Tuerk

Part XVIII

Practical Program Verification with CakeML



- CakeML is a functional programming language in the SML family
- CakeML has a verified compiler which takes a long time to bootstrap in HOL4
- Even without bootstrapping the compiler, we can use CakeML theories to verify (HOL4) functions
- We use the v1009 release of CakeML:

<https://github.com/CakeML/cakeml/releases/download/v1009/cake-x64-64.tar.gz>

<https://github.com/CakeML/cakeml/archive/v1009.tar.gz>

- SML translate function, taking HOL4 function/data as input
- if successful, adds CakeML AST to current program state and outputs equivalence theorem
- has been used to generate and prove correct a significant fraction of the SML basis library for CakeML
- separate from the post-hoc verification environment (better suited for imperative programs)

Simple Finite Map Encoding in HOL4



```
val _ = new_theory "simple_bst";

val singleton_def = Define `
  singleton k v = Node k v Leaf Leaf`;

val lookup_def = Define `
  lookup cmp k Leaf = NONE
  lookup cmp k (Node k' v' l r) =
    case cmp k k' of
    | Less => lookup cmp k l
    | Greater => lookup cmp k r
    | Equal => SOME v';

val insert_def = Define `
  insert cmp k v Leaf = singleton k v
  insert cmp k v (Node k' v' l r) =
    case cmp k k' of
    | Less => Node k' v' (insert cmp k v l) r
    | Greater => Node k' v' l (insert cmp k v r)
    | Equal => Node k' v l r`;
```

```
open preamble ml_progLib ml_translatorLib astPP simple_bstTheory;  
  
fun get_current_prog() =  
let  
  val state = get_ml_prog_state()  
  val state_thm =  
    state |> ml_progLib.remove_snocs |>  
    ml_progLib.clean_state |> get_thm  
  val current_prog =  
    state_thm |> concl |> strip_comb |> #2 |> el 2  
in current_prog end;  
  
val res = translate singleton_def;  
val res = translate lookup_def;  
val res = translate insert_def;  
  
val _ = astPP.enable_astPP();  
print_term (get_current_prog());
```

Pretty Printed Translator Output



```
datatype 'a option = Some ('a ) | None;

datatype ( 'k , 'w ) simple_bst_btree =
Node ('k ) ('w ) (('k , 'w ) simple_bst_btree)
  (('k , 'w ) simple_bst_btree) | Leaf;

fun singleton v1 = (fn v2 => (Node (v1) (v2) (Leaf) (Leaf))));

datatype ternaryComparisons_ordering = Greater | Equal | Less;

fun insert v5 v6 v8 v7 =
case v7
of Leaf => (singleton v6 v8)
| (Node (v4) (v3) (v2) (v1)) => (case (v5 v6 v4)
of Less => ((Node (v4) (v3) (insert v5 v6 v8 v2) (v1)))
| Equal => ((Node (v4) (v8) (v2) (v1)))
| Greater => ((Node (v4) (v3) (v2) (insert v5 v6 v8 v1))));

fun lookup v5 v6 v7 =
case v7
of Leaf => None
| (Node (v4) (v3) (v2) (v1)) => (case (v5 v6 v4)
of Less => (lookup v5 v6 v2)
| Equal => ((Some (v3)))
| Greater => (lookup v5 v6 v1));
```

Organizing the Program Verification Effort



- definitions can be suitable for reasoning or execution, but seldom both
- correctness arguments should be done at high abstraction level
- certification of programs can be separated from correctness reasoning

- 1 encode problem using **proof-friendly** datatypes in HOL4 (lists, sets)
- 2 state and prove main correctness properties **abstractly**, e.g., using relations and pure functions
- 3 figure out and encode execution-friendly datatypes
- 4 refine proof-friendly functions and data to execution-friendly ones
- 5 apply CakeML translator on execution-friendly functions and data
- 6 compile translated functions and data using standalone compiler, or generate machine code directly

Case Study: Propositional Logic Proof Checker



- proof system taken from the book “Logic for Computer Science” by Huth and Ryan
- system (specification) is a set of inference rules
- correctness of the system is that rules are sound
- an executable proof checker validates that a given proof adheres to the inference rules
- code at <https://github.com/palmskog/fitch>

Example Proof With Box



```
q |- p -> q
[
  1 p assumption
  2 q premise
]
3 p -> q impi 1-2
```

Examples of Inference Rules



$$\frac{\Gamma(l') = \phi \wedge \phi'}{\Gamma, \bar{\phi} \vdash l \phi \wedge e_1 l'} \quad \text{VD_ANDE1} \qquad \frac{\Gamma(l_1) = \phi \rightarrow \phi' \quad \Gamma(l_2) = \neg \phi'}{\Gamma, \bar{\phi} \vdash l \neg \phi \quad \text{MT } l_1, l_2} \quad \text{VD_MT}$$

$$\frac{\Gamma(l') = \phi}{\Gamma, \bar{\phi} \vdash l \phi \vee \phi' \vee i_1 l'} \quad \text{VD_ORI1} \qquad \frac{\Gamma(l') = \phi}{\Gamma, \bar{\phi} \vdash l \neg \neg \phi \neg \neg i l'} \quad \text{VD_NEGNEGI}$$

$$\frac{\Gamma(l') = \perp}{\Gamma, \bar{\phi} \vdash l \phi \perp e l'} \quad \text{VD_CONTE} \qquad \frac{\Gamma(l_1, l_2) = (\phi, \phi')}{\Gamma, \bar{\phi} \vdash l \phi \rightarrow \phi' \rightarrow i l_1 - l_2} \quad \text{VD_IMPI}$$

Reasoning Friendly Function



```
Definition valid_derivation_deriv_imp_i:  
  valid_derivation_deriv_imp_i G l1 l2 p =  
    case p of  
      | prop_imp p1 p2 =>  
        (case FLOOKUP G (INR (l1, l2)) of  
          | SOME (INR (p3, p4)) => p1 = p3 /\ p2 = p4  
          | _ => F)  
      | _ => F  
End  
  
Theorem valid_derivation_deriv_imp_i_sound:  
  !G pl l1 l2 l' p.  
    valid_derivation_deriv_imp_i G l1 l2 p <=>  
    valid_derivation G pl (derivation_deriv l' p  
      (reason_justification (justification_imp_i l1 l2)))  
Proof  
  (* ... *)  
Qed
```

```
Definition valid_derivation_deriv_imp_i_cake:
  valid_derivation_deriv_imp_i_cake t l1 l2 p =
    case p of
    | prop_imp p1 p2 =>
      (case lookup t (INR (l1, l2)) of
      | SOME (INR (p3, p4)) => p1 = p3 /\ p2 = p4
      | _ => F)
    | _ => F
End
```

```
Theorem valid_derivation_deriv_imp_i_eq:
  !t l1 l2 p. map_ok t ==>
    valid_derivation_deriv_imp_i_cake t l1 l2 p =
    valid_derivation_deriv_imp_i (to_fmap t) l1 l2 p
```

```
Proof
rw [valid_derivation_deriv_imp_i_cake,valid_derivation_deriv_imp_i] \
rw [lookup_thm]
QED
```

Generated CakeML Function



```
fun valid_derivation_deriv_impi_cake v17 =
  (fn v14 =>
    (fn v15 =>
      (fn v16 =>
        case v16
        of (Prop_p (v1)) => (0 < 0)
          | (Prop_neg (v2)) => (0 < 0)
          | (Prop_and (v4) (v3)) => (0 < 0)
          | (Prop_or (v6) (v5)) => (0 < 0)
          | (Prop_imp (v13) (v12)) =>
            (case (Map.lookup v17 (let val x = (v14,v15)
in
              (Inr (x))
            end))
            of None => (0 < 0)
              | ((Some (v11))) => (case v11
of ((Inl (v7))) => (0 < 0)
  | ((Inr (v10))) => (case v10
of (v9,v8) => ((v13 = v9) andalso (v12 = v8))))))
          | Prop_cont => (0 < 0)))));
```