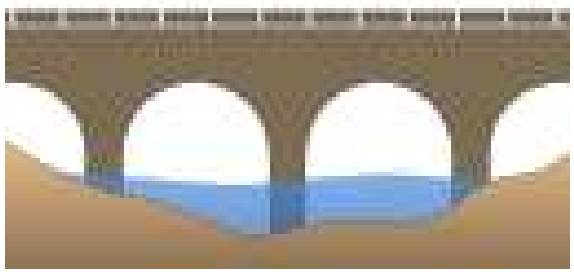
Defining a LISP Interpreter in a Logic of Total Functions

HOL ACL2



Grand goals

- Allow projects combining ACL2 and HOL
  - *e.g.* JVM on ARM
- Solid soundness story
  - Common Criteria EAL7 compatible
- Merge HOL and ACL2 communities
  - well ... at least increase collaboration ...

Today's tiny step

- Compare partial functions in HOL and ACL2

Goal: an ACL2 interpreter in HOL

- Would like to define an ACL2 interpreter in HOL
- Prove interpreter evaluation validates ACL2 axioms
- Problem: functions in HOL and ACL2 must be total
- ACL2 interpreter (EVAL) is not a total function
- We compare HOL and ACL2 methods for partial functions
  - a simple example used here to explain ideas
  - application to EVAL in proceedings paper
  - only made a first proof-of-concept step (pure Lisp)

Partial functions as relations in HOL

- HOL approach is to encode partial functions as relations

$$f \rightsquigarrow R_f \quad \text{where} \quad f(x) = y \;\Leftrightarrow\; R_f(x, y)$$

- One ACL2 approach is to add a 'clock' parameter

$$f \rightsquigarrow f_c \quad \text{where} \quad f_c(0, \cdots) = \mathit{none} \quad \text{(some 'timeout' value)}$$
$$\text{and} \quad f_c(n, \cdots) = \cdots f_c(n-1, \cdots) \cdots$$

- What is connection between relations and clocking?

- Would like: $\vdash R_f(x, y) \;\Leftrightarrow\; \exists n.\, f_c(n, x) = y \;\wedge\; y \neq \mathit{none}$

Representing partial recursive functions

Assume p, q, h, k given, define f recursively by:

$f(x) = \texttt{if}\ p(x)\ \texttt{then}\ q(x)\ \texttt{else}\ h(x, f(k(x)))$

- Define R_f to be the least relation such that:

  $(\forall x.\ p(x) \Rightarrow R_f(x, q(x)))$
  $\wedge$
  $(\forall x\ y.\ \neg p(x) \wedge R_f(k(x), y) \Rightarrow R_f(x, h(x, y)))$

- Clocked version:

  $f_c(n, x) = \texttt{if}\ n = 0\ \texttt{then}\ \mathit{none}\ \texttt{else}$
  $\texttt{if}\ p(x)\ \texttt{then}\ q(x)\ \texttt{else}\ h(x, f_c(n-1, k(x)))$

- One direction works: $\vdash R_f(x, y) \Rightarrow \exists n.\ f_c(n, x) = y$

- Alas, not: $\vdash (\exists n.\ f_c(n, x) = y \wedge y \neq \mathit{none}) \Rightarrow R_f(x, y)$

- Problem is the timeout value *none*

Example illustrating *none* problem

- Consider: $f(x) = \texttt{if } x = 0 \texttt{ then } 0 \texttt{ else } x + f(x-1)$
- Relation version

  $(\forall x.\, x = 0 \Rightarrow R_f(x, 0))$
  $\wedge$
  $(\forall x\ y.\, x \neq 0 \wedge R_f(x-1, y) \Rightarrow R_f(x, x+y))$
- Clocked version

  $f_c(n, x) = \texttt{if } n = 0 \texttt{ then } \mathit{none} \texttt{ else}$
  $\texttt{if } x = 0 \texttt{ then } 0 \texttt{ else } x + f_c(n-1, x-1)$
- If $\mathit{none} = 0$
  then $f_c(1, 2){=}2 \ \wedge\ 2{\neq}\mathit{none}$
  but $\forall y.\, R_f(2, y) \Rightarrow y{=}3$ so not $R_f(2, 2)$
- Can show implication below – but what is *none*?

  $\vdash (\forall x.\, x + \mathit{none} = \mathit{none})$
  $\Rightarrow$
  $\forall y.\, \neg(y{=}\mathit{none}) \Rightarrow ((\exists n.\, f_c(n, x) = y) \Leftrightarrow R_f(x, y))$

Solution: make success and timeout value different

- Need to distinguish success values of $f_c(n,x)$ from *none*
  - make $f_c(n,x)$ return $some(y)$ (success) or *none* (timeout)
  - where $\forall y.\ some \neq none(y)$
- Must explicity propagate *none* value:

$$
\begin{array}{l}
f_c(n,x) = \texttt{if } n{=}0 \texttt{ then } none \texttt{ else} \\
\qquad \texttt{if } p(x) \texttt{ then } some(q(x)) \\
\qquad\qquad \texttt{else case } f_c(n{-}1,k(x)) \texttt{ of} \\
\qquad\qquad\qquad none \quad \rightarrow none \\
\qquad\qquad\qquad \mid some(y) \rightarrow some(h(x,y))
\end{array}
$$

- Then can prove: $\vdash R_f(x,y) \ \Leftrightarrow\ \exists n.\ f_c(n,x) = \mathrm{some}(y)$
- Compare with
  - 'direct' denotational semantics
  - exception monad

This idea works for Lisp evaluator

- See paper for details (unexplained sample below):

$$\vdash (R_{ap}(fn, args, \rho, s) = \exists n.\ \mathrm{some}(s) = apply_c(n, fn, args, \rho))$$
$$\wedge$$
$$(R_{ev}(e, \rho, s) = \exists n.\ \mathrm{some}(s) = eval_c(n, e, \rho))$$
$$\wedge$$
$$(R_{evl}(el, \rho, sl) = \exists n.\ \mathrm{some}(\mathrm{List}(sl)) = evlis_c(n, el, \rho))$$

- No *obvious* reason why can't be done for full ACL2
- Why bother?
  - satisfying and fun (my 1974 PhD was on verifying EVAL)
  - relate HOL-style and ACL2-style formal methods
  - alternative to V&C& for bounded quantification in ACL2 (reviewer comment)

THE END