Formal Verification of LabVIEW Programs with ACL2
(Preliminary Work)

Matt Kaufmann
Jacob Kornerup
Grant Passmore
Mark Reitblatt
BRIEF HISTORY

- Jeff Kodosky started playing around in 2004 with the idea of verifying a LabVIEW program.
- Warren Hunt and J Moore met on occasion with Jeff and Jacob Kornerup over several years, culminating with NI engaging Grant as an intern in 2005.
- Grant implemented a first approach and used it to prove Gauss’s theorem that the sum of the integers from 1 to n is \( n(n+1)/2 \).
- This summer: Alternate approach models LabVIEW programs, including loop structures, directly as ACL2 functions.
- Grant left for Edinburgh late this summer to start his Ph.D. work, and transferred his infrastructure support to Mark Reitblatt, now an NI intern from UT CS.
Every module, primitive or not, takes and returns a single alist that we call a \textit{record}, by calling $S\ast$, “Set”.

Every wire returns a LabVIEW data value, obtained by applying $G$, “get”, to a record.
(DEFUN X0 (IN) (G :IN0 IN))
(DEFUN Y0 (IN) (G :IN1 IN))
(DEFUN N0 (IN) (S* :OUT (+ (X0 IN) (Y0 IN))))
(DEFUN W0 (IN) (G :OUT (N0 IN)))
(DEFUN N1 (IN) (S* :OUT (1+ (W0 IN))))
(DEFUN Z0 (IN) (G :OUT (N1 IN)))
ACL2 REPRESENTATION, p. 3

(defun n$init (in) (s* :IN0 (x in) :IN1 (y in)))
(defun n (in) (s* :OUT (Z0 (n$init in))))
(defun z (in) (g :OUT (n in)))
(thm (equal (z in) (+ 1 (x in) (y in)))))
An assertion is simply a Boolean-valued wire that can be checked at runtime.

Goal: prove that each assertion is true

Focus to date: For-loops and while-loops
FOR-LOOP VERIFICATION IDEA

- We model for-loops in a straightforward way as recursive functions.

- We introduce a generic property and a generic for-loop, and we prove a generic theorem about them.

- For each actual for-loop, we employ functional instantiation to avoid the use of induction.
(encapsulate ; signature and locals omitted
 (defthm prop-generic-step
   (implies (and (natp n) (natp (g :lc in))
     (< (g :lc in) n)
     (prop-generic in))
   (prop-generic (s :lc (1+ (g :lc in))
     (step-generic in))))))

(defun loop-generic (n in) ; measure omitted
   (cond ((or (not (natp n))
     (not (natp (g :lc in)))
     (>= (g :lc in) n))
     in)
   (t (loop-generic n (s :lc (1+ (g :lc in))
     (step-generic in)))))

(defthm loop-generic-thm
   (implies (and (natp n) (natp (g :lc in))
     (prop-generic in))
   (prop-generic (loop-generic n in))))
FOR-LOOP PROPERTY IS PRESERVED:

; User proves this (automatically if lucky):
(defthmd1 _N_4$prop{_N_7$step}
 (implies (and (natp (g :lc in))
   (< (g :lc in) n)
   (_N_4$prop in))
 (_N_4$prop (s :lc (1+ (g :lc in))
   (_N_7$step in))))

(defthml _N_4$prop{_N_7}
 (implies (and (natp n) (natp (g :lc in))
   (_N_4$prop in))
 (_N_4$prop (_N_7$loop n in)))
:hints ("Goal"
 :by (:functional-instance
 loop-generic-thm
 (step-generic _N_7$step)
 (prop-generic _N_4$prop)
 (loop-generic _N_7$loop))
 :in-theory (union-theories '(_N_4$prop{_N_7$step})
 (theory 'minimal-theory)
 :expand ((_N_7$LOOP n in)))
 :rule-classes nil)
(in-package "ACL2")

; Translation to ACL2:
(include-book "gauss2-fns")

; User-editable -- note use of LOCAL!!
(local (include-book "gauss2-work"))

(set-enforce-redundancy t)

(defthm _N_330$INV ; desired result
  (implies (natp (g :_N_10_[FOO] in))
            (g :inv (_N_330 in))))
CURRENT STATUS

- For-loop example and while-loop example completed, in an automatable, scalable style.

- Automatic translation is implemented for some data types.
TO DO:

- More data types (lists) and more faithful translation for bounded integers
- Limited I/O and global variables (just starting but optimistic)
- Better interface support, e.g.:
  - Wizards to help guide proofs, e.g. suggesting our induction-avoiding approach for assertions about for-loops.
  - Assertion management, e.g., automatic removal of proved assertion wires
  - Further investigation into while loops (perhaps \texttt{defpun} and/or assistance for termination)
  - Suitable graphical support to help with conceptualization
- More examples! (Mark’s senior thesis....)
- Real-time verification (for LabVIEW on FPGAs)
- Co-simulation to check translation (hooray for \texttt{mbe}!)
- Reusability (“sub-VIs”)
- Decision procedures (as clause-processors)
- Goal: NI Labs (\url{http://www.ni.com/labs/})