Two Handy \texttt{update-nth} Equality Rules

David Greve  
Matthew Wilding

Function \texttt{update-nth} is an ACL2 builtin function that operates on lists.

\begin{verbatim}
(defun update-nth (key val 1)
  (declare (xargs :guard (true-listp 1))
    (type (integer 0 *) key))
  (cond ((zp key) (cons val (cdr 1)))
    (t (cons (car 1)
      (update-nth (1- key) val (cdr 1))))))
\end{verbatim}

The ACL2 user who models using lists and who uses \texttt{update-nth} for updating is often faced with proving the equality of terms composed of multiple calls of \texttt{update-nth}. This includes ACL2 users who use stobj, which introduces functions that are defined in terms of \texttt{update-nth}.

It was not obvious to us at first how best to prove these kinds of lemmas. We initially tried to induct over the list in a manner suggested by the recursive call of \texttt{update-nth}. However, after several fruitless days of trying to prove these kinds of conjectures, we developed a different strategy: rewrite a term of the form (\texttt{equal (update-nth n v 11) 12}) to the conjunction of:

- 12 is identical to 11 on the values preceding the \texttt{n}th,
- 12 is identical to 11 on the values succeeding the \texttt{n}th,
- 12 has an \texttt{n}th element and it is identical to \texttt{v}.

The ACL2 book that accompanies this note contains rules that incorporate this strategy. The rule \texttt{equal-update-nth-casesplit} breaks an equality term including an \texttt{update-nth} into cases. A special case of this rule is used to prove the equality of two \texttt{update-nth} terms where the same value being updated and the lists being updated have the same length.
(defthm equal-update-nth-casesplit
  (implies
   (and (integerp n) (<= 0 n))
   (equal
    (equal (update-nth n v 11) 12)
    (and
     (and (equal (nth n 12) v) (< n (len 12)))
     (equal (firstn n (append 11 (repeat (- n (len 11)) nil)))
        (firstn n 12))
     (equal (nthcdr (1+ n) 11) (nthcdr (1+ n) 12))))))

(defthm equal-update-nth-update-nth
  (implies
   (and (integerp n) (<= 0 n) (equal (len 11) (len 12)))
   (equal
    (equal (update-nth n v1 11) (update-nth n v2 12))
    (and
     (equal v1 v2)
     (equal (firstn n 11) (firstn n 12))
     (equal (nthcdr (1+ n) 11) (nthcdr (1+ n) 12))))))

These rules do not typically lead to a large number of cases, even for equality expressions involving large nests of update-nths. Initially, application of one of these rules doubles the number of update-nths in the term being simplified, but other rules eliminate update-nths. For example, when element locations in these expressions are constants (as they typically are when reasoning about stobjs) and one of the rules above “duplicates” a nest of update-nth function calls, the following rules eliminate at least half the update-nth occurrences.

(defthm firstn-update-nth
  (implies
   (and (integerp n) (<= 0 n) (integerp n2) (<= 0 n2))
   (equal
    (firstn n (update-nth n2 v 1))
    (if (<= n n2)
        (append (firstn n 1) (repeat (- n (len 1)) nil))
        (update-nth n2 v (firstn n 1))))))

(defthm nthcdr-update-nth
  (implies
   (and (integerp n) (<= 0 n) (integerp n2) (<= 0 n2))
   (equal
    (nthcdr n (update-nth n2 v 1))
    (if (< n n2)
        (nthcdr n 1)
        (update-nth (- n2 n) v (nthcdr n 1))))))

The accompanying book also includes analogous rules for update-nth-array.