Two Handy `update-nth` Equality Rules

David Greve
Matthew Wilding

Function `update-nth` is an ACL2 builtin function that operates on lists.

```
(defun update-nth (key val l)
   (declare (xargs :guard (true-listp l))
            (type (integer 0 *) key))
  (cond ((zp key) (cons val (cdr l)))
        (t (cons (car l)
                 (update-nth (1- key) val (cdr l))))))
```

The ACL2 user who models using lists and who uses `update-nth` for updating is often faced with proving the equality of terms composed of multiple calls of `update-nth`. This includes ACL2 users who use stobj, which introduces functions that are defined in terms of `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 `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 `(equal (update-nth n v l1) l2)` to the conjunction of:

- `l2` is identical to `l1` on the values preceding the `n`th,
- `l2` is identical to `l1` on the values succeeding the `n`th,
- `l2` has an `n`th element and it is identical to `v`.

The ACL2 book that accompanies this note contains rules that incorporate this strategy. The rule `equal-update-nth-casesplit` breaks an equality term including an `update-nth` into cases. A special case of this rule is used to prove the equality of two `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 l1) l2)
    (and
     (and (equal (nth n l2) v) (< n (len l2)))
     (equal (firstn n (append l1 (repeat (- n (len l1)) nil)))
            (firstn n l2))
     (equal (nthcdr (1+ n) l1) (nthcdr (1+ n) l2))))))

(defthm equal-update-nth-update-nth
  (implies
   (and (integerp n) (<= 0 n) (equal (len l1) (len l2)))
  (equal
   (equal (update-nth n v1 l1) (update-nth n v2 l2))
   (and
    (equal v1 v2)
    (equal (firstn n l1) (firstn n l2))
    (equal (nthcdr (1+ n) l1) (nthcdr (1+ n) l2))))))
```

These rules do not typically lead to a large number of cases, even for equality expressions involving large nests of `update-nth`s. Initially, application of one of these rules doubles the number of `update-nth`s in the term being simplified, but other rules eliminate `update-nth`s. 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 l))
   (if (<= n n2)
       (append (firstn n l) (repeat (- n (len l)) nil))
     (update-nth n2 v (firstn n l))))))

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

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