Using ACL2 for Set Operations

Sets as Ordered Lists without Duplicates

Warren A. Hunt, Jr.
hunt@cs.utexas.edu

Computer Science Department
University of Texas
2317 Speedway, M/S D9500
Austin, TX 78712-0233

February, 2025

Sets as Lists

How might we represent ordered sets using ACL2?

```
(defun << (x y)
  (declare (xargs :guard t))
  ;; LEXORDER is like <=
  (and (lexorder x y)
       (not (equal x y))))

(defun setp (x)
  (declare (xargs :guard t))
  (if (atom x)
      (null x)
    (if (atom (cdr x))
        (null (cdr x))
      ;; Compare the first two objects
      (let ((a (car x))
            (b (cadr x)))
        (and (<< a b)
             (setp (cdr x)))))))
```

What kind of sets are recognized by `setp`?

Can you write an element insertion function?

Set Insertion

Given that our setp recognizer requires any extension to be ordered makes it clear that we need to find the proper insertion place.

```
(defun insrt (e x)
  "Insert E into ordered set X."
  (declare (xargs :guard (setp x)))
  (if (atom x)
      (list e)
    (let ((l (car x)))
      (if (<< e l)
          (cons e x)
        (if (equal e l)
            x
          (cons l
                (insrt e (cdr x))))))))
```

Can you prove this?

```
(defthm setp-insrt
  (implies (setp x)
           (setp (insrt e x))))
```

Set Membership

Can you write a set membership function?

```
(defun mbr (e x)
  (declare (xargs :guard (setp x)
                  :verify-guards nil))
  (if (atom x)
      ...




    ))
```

Why not verify the guards? We don't know this inductive fact:

```
(defthm not-mbr-when-e-is-<<-car-x
  (implies (and (setp x)
                (<< e (car x)))
           (not (mbr e x))))

(verify-guards mbr)  ;; Show TRACE$ difference
```

Is E a Member After Insertion?

After inserting `E` into set `X`, will we find it?

```
(defthm mbr-insrt
  (implies (setp x)
           (mbr e (insrt e x))))
```

If we insert `E` into set `X`, will `A` still be a member?

```
(defthm mbr-a-mbr-insrt
  ;; Item A still a member after any insertion.
  (implies (and (setp x)
                (mbr a x))
           (mbr a (insrt e x))))
```

Set Element Deletion

Can we remove an element from our set leaving it ordered?

```
(defun del (e x)
  "Delete E from set X, or do nothing if no E in X."
  ;; When (SETP x), we can stop after a deletion.
  (declare (xargs :guard (setp x)
                  :verify-guards nil))
  (if (atom x)
      NIL
      ...



  ))
```

Do we have to delete all E items? Can we establish the following?

```
(defthm del-when-e-is-<<-car-x
  (implies (and (setp x)
                (<< e (car x)))
           (equal (del e x) x)))

(verify-guards del)  ;; Show TRACE$ difference
```

Some Facts About DEL

Here are facts that we should know. Can you prove them?

```
(defthm setp-del
  ;; Deletion leaves a set
  (implies (setp x)
           (setp (del e x))))

(defthm not-mbr-del
  ;; Item E should not be a member after its deletion
  (implies (setp x)
           (not (mbr e (del e x)))))

(defthm mbr-a-del-e
  ;; Is item A still a member if different element deleted?
  (implies (and (not (equal a e))
                (setp x))
           (equal (mbr a (del e x))
                  (mbr a x))))
```

Are there other properties we would like to establish?

What about Indexing?

```
(defun nth (n l)
  (if (atom l)
      nil
    (if (zp n)
        (car l)
      (nth (- n 1) (cdr l)))))

(defun update-nth (key val l)
  (cond ((zp key) (cons val (cdr l)))
        (t (cons (car l)
                 (update-nth (1- key) val (cdr l))))))
```

Can you prove:

```
(defthm nth-of-update-nth
  ;; Is what your wrote where you put it?
  (equal (nth i (update-nth i v l)) v))
```

What about?

```
(defthm update-nth-of-nth
  ;; Do you get back the same L?
  (equal (update-nth i (nth i l) l) l))
```