Finite Set Theory *based on* Fully Ordered Lists

Jared Davis
UT Austin

ACL2 Workshop
2004

Motivation (1/2)

- Unique representation for each set
  - No mutual recursion needed for membership, subset, and set equality
  - Unified notion of set, element equality
  - Nested sets supported trivially

- Efficiency characteristics
  - Unordered sets: constant time insert, quadratic subset, equality, difference, intersection, cardinality
  - Ordered sets: all operations are linear

Motivation (2/2)

- A challenging representation for reasoning
  - "I found this approach to complicate set construction to a degree out of proportion to its merits. In particular, functions like *union* and *intersection*, which are quite easy to reason about in the list world (where order and duplication matter but are simply ignored), become quite difficult to reason about in the set world, where most of the attention is paid to the sorting of the output with respect to the total ordering." – J S. Moore
- A learning experience
  - Familiar domain, easy to understand intuitively yet challenging to get ACL2 to prove things
  - Highly recommended for teaching, exercises

Implementation Preliminaries

- A total order over ACL2 objects, <<
  - Taken verbatim from *books/misc/total-order*
  - Fairly intuitive: **`(<< 1 2), (<< 'a 'b), (<< "a" "b")`**

- A set recognizer, *setp*
  - Lists which are totally ordered under the above relation
  - Total ordering implies no duplicates, linear execution

```
(setp X) = (if (atom X)
               (null X)
            (or (null (cdr X))
                (and (consp (cdr X))
                     (<< (car X) (cadr X))
                     (setp (cdr X))))))
```

Primitive Level

- Levels of abstraction: primitive, membership, outer

- Non-set objects as the empty set, the "non-set convention"
  - The *list primitives* (car, cdr, cons, endp) do not respect this convention! E.g.,

```
(car '(2 1))     = 2
(cdr '(1 1))     = (1)
(cons 3 '(2 1)) = (3 2 1)
(endp '(1 1))    = nil
```

  - New *set primitives* as replacements

    head replaces car, *tail* replaces cdr, *insert* replaces cons, *empty* replaces endp, *sfix* is set identity

Membership Level (1/4)

- Set membership is straightforward

```
(in a X) = (and (not (empty X))
                (or (equal a (head X))
                    (in a (tail X)))))
```

  - Nice theorems, e.g., (in a (insert b X)) = (or (equal a b) (in a X))

- Moving away from the set order
  - (not (in (head X) (tail X)))
  - Weak induction scheme for (insert a X)

- Ultimate goal: entirely hide the set order

Membership Level (2/4)

- We introduce a naïve, quadratic subset

```
(subset X Y) = (or (empty X)
                   (and (in (head X) Y)
                        (subset (tail X) Y)))
```

- Pick-a-point strategy for proving subset
  - Intent: (implies (∀a, (implies (in a X) (in a Y))

    (subset X Y))
  - More natural than induction
  - Reduces subset proofs to membership of single elements
  - Cannot represent this as a direct ACL2 theorem!

Membership Level (3/4)

- Implementing the pick-a-point strategy
  - Introduce undefined function symbols *hyps*, *sub,* and *super*, with the following constraint:

```
(implies (and (hyps)
              (in a (sub)))
         (in a (super)))
```

  - Prove the generic theorem *subset-by-membership*

```
(implies (hyps)
         (subset (sub) (super)))
```

  - Use functional instantiation of the above to reduce proofs of subset to proofs of membership.

Membership Level (4/4)

- Computed hints are used to automatically attempt functional instantiation to prove subsets

- Using the pick-a-point method, we prove

```
(defthm double-containment
  (implies (and (setp X)
                (setp Y))
           (equal (equal X Y)
                  (and (subset X Y)
                       (subset Y X)))))
```

- The big picture
  - Set equality proven by double containment, containment via membership.
  - No further use for the set order

Example Proof (1/2)

Theorem: **(subset (tail x) x)**

```
This simplifies, using the :definition SYNP and the :rewrite rule
PICK-A-POINT-SUBSET-STRATEGY, to

Goal':
(SUBSET-TRIGGER (TAIL X) X).

We suspect this conjecture should be proven by functional
instantiation of ALL-BY-MEMBERSHIP.  This suspicion is caused by
PICK-A-POINT-SUBSET-STRATEGY, so if this is not what you want to
do, then you should disable PICK-A-POINT-SUBSET-STRATEGY.
Accordingly, we suggest the following hint:

("Goal'"
 (:USE ((:FUNCTIONAL-INSTANCE ALL-BY-MEMBERSHIP
         (ALL-HYPS (LAMBDA NIL T))
         (ALL-SET (LAMBDA NIL (TAIL X)))
         (ALL (LAMBDA (COMPUTED-HINTS::?X)
                      (SUBSET COMPUTED-HINTS::?X X)))
         (PREDICATE (LAMBDA (COMPUTED-HINTS::?X)
                            (IN COMPUTED-HINTS::?X X)))))
  :EXPAND ((SUBSET-TRIGGER (TAIL X) X))))
```

Example Proof (2/2)

```
[Note:  A hint was supplied for our processing of the goal below.
Thanks!]

We now augment the goal above by adding the hypothesis indicated by
the :USE hint.  The hypothesis can be derived from ALL-BY-MEMBERSHIP
via functional instantiation, bypassing constraints that have been
proved when processing the event SUBSET-REFLEXIVE, provided we can
establish the constraint generated.  Thus, we now have the two
subgoals shown below.

Subgoal 2
(IMPLIES (IMPLIES T (SUBSET (TAIL X) X))
         (SUBSET (TAIL X) X)).

But we reduce the conjecture to T, by case analysis.

Subgoal 1
(IMPLIES (IN ARBITRARY-ELEMENT (TAIL X))
         (IN ARBITRARY-ELEMENT X)).

But simplification reduces this to T, using the :rewrite rule IN-TAIL
and the :type-prescription rule IN-TYPE.

Q.E.D.
```

Top Level

- Union, intersection, difference, deletion, cardinality
  - Naïve definitions used for easy reasoning
  - Extensive set of theorems, including distributing unions over intersections, DeMorgan laws for distributing differences, etc.
  - *No use of set order*
  - Proofs are automatic, i.e., no manual hints are needed
  - Proofs are natural, following the traditional style of hand-written set theory proofs

Achieving Efficiency (1/3)

- Review of MBE, Guards
  - Guards let us state “intended domains” of functions
  - MBE lets us provide two definitions for a function:

    logical definition is used for reasoning

    executable definition is used for evaluation
  - Obligation: both definitions must be logically equal when the function's guards are satisfied.

- We “MBE in” efficient executable definitions
  - Logical definitions do not change
  - *All of our theorems, proof techniques are still valid!*

Achieving Efficiency (2/3)

- Recall: set primitives (head, tail, ...) must examine entire set to enforce the non-set convention.
  - But even checking (setp X) is linear

- We guard primitives to operate only on sets, and use MBE to substitute:

(sfix X)	=> X	O(n) => O(1)
(head X)	=> (car X)	O(n) => O(1)
(tail X)	=> (cdr X)	O(n) => O(1)
(empty X)	=> (endp X)	O(n) => O(1)

  - Insert is left alone, becomes linear automatically
  - Non-sets convention becomes computationally “free”

Achieving Efficiency (3/3)

- More substitutions
  - Linear subset replacement is introduced
  - Cardinality becomes *len*
  - Linear union, intersection, difference introduced
    - Equivalences proven through double containment
    - Even membership properties of fast definitions are difficult to establish (must be based on set order)

- Library operations are now linear time
  - Yet reasoning is not impacted
  - We still get to use our simple, clean definitions

Adding a Sort

- Insertion is a problem for efficiency
  - Building large sets is quadratic, versus linear for unordered sets

- Mergesort added to address this issue
  - MBE is again useful: mergesort is logically repeated insertion, executably $O(n \log_2 n)$
  - Simple to implement, uses union to perform the merge
  - This will never be as efficient as unordered sets

Instantiable Extensions

- A theory of quantification
  - Given predicate P, extend this predicate across sets:

    (all<P> X) = forall a in X, (P a ...) holds

    (exists<P> X) = exists a in X such that (P a ...) holds

    (find<P> X) = find an a in X such that (P a ...) holds
  - Instantiate these definitions, and dozens of theorems including a custom pick-a-point strategy for all<P> with a single macro call.

- Higher order functional programming
  - Filter sets, returning all elements satisfying a predicate
  - Map functions over sets, reason about inverses, etc.

Future Directions

- Further automating functional instantiation?
- Custom set orders?
- Improving the viability of instantiable theories?
  - Current system introduces hundreds of events
  - Writing the macros to instantiate them is tedious
- Applying these techniques to other containers?

Thanks!

- Sets library available under the GPL
  - Old version distributed with ACL2 2.8 and 2.9
  - Current version:

    http://www.cs.utexas.edu/users/jared/osets/

- Special thanks to...

  Sandip Ray, Robert Krug, Matt Kaufmann, Eric Smith, J Moore, Hanbing Liu, Serita Nelesen, Omar El-Domeiri, and everyone at UT's weekly ACL2 Seminar.