Adding a Total Order to ACL2

Panagiotis Manolios\textsuperscript{1} and Matt Kaufmann\textsuperscript{2}

\textsuperscript{1} College of Computing, CERCS Lab
Georgia Institute of Technology
manolios@cc.gatech.edu
http://www.cc.gatech.edu/~manolios,
\textsuperscript{2} Advanced Micro Devices, Inc.
matt.kaufmann@amd.com

Abstract. We show that adding a total order to ACL2, via new axioms, allows for simpler and more elegant definitions of functions and libraries of theorems. We motivate the need for a total order with a simple example and explain how a total order can be used to simplify existing libraries of theorems (i.e., ACL2 books) on finite set theory and records. These ideas have been incorporated into ACL2 Version 2.6, which includes axioms positing a total order on the ACL2 universe.

1 Introduction

ACL2 [7, 6, 8] is a logic of total functions. One particularly pleasant consequence is that many properties of functions can be stated as unconditional rewrite rules. For example, we can prove (\texttt{equal (\texttt{\* y (\texttt{\* x z)}) (\texttt{\* x (\texttt{\* y z)})}}) without having to establish that \texttt{x}, \texttt{y}, and \texttt{z} are numbers. Such unconditional rewrite rules lead to simpler libraries of theorems, which in turn improve the ability of ACL2 to reduce large terms automatically and efficiently.

Unfortunately, it is problematic to exploit fully the totality of functions in ACL2 Version 2.5. One is often forced to use rewrite rules with hypotheses because of the lack of a definable total order on the ACL2 universe.

To examine the issue, consider the problem of sorting, i.e., the problem of defining a function that returns an ordered permutation of its input. Since ACL2 is untyped, one might expect that it is possible to define a function \texttt{isort} that satisfies the following two properties (where \texttt{(perm x y)} holds iff \texttt{x} is a permutation of \texttt{y}).

\begin{verbatim}
(perms x (isort x))
(equal (perm x y)
    (equal (isort x)
          (isort y)))
\end{verbatim}

Unfortunately, this does not seem to be the case, even if we restrict \texttt{x} and \texttt{y} to lists of length two. One is forced to write sorting functions for different types, the very thing we are trying to avoid.

The reason why a sorting function with the above properties cannot be defined is that the ACL2 universe is not closed. That is, the ACL2 axioms

\texttt{\textbackslash{}texttt{(\textbackslash{}texttt{\* y (\texttt{\* x z)}) (\texttt{\* x (\texttt{\* y z)})}}) without
do not rule out the existence of bad atoms, atoms that do not satisfy any of ac12-numberp, symbolp, characterp, or stringp. To define a sorting function on the whole of the ACL2 universe, we need a way to order bad atoms. This is not possible since in general the set of bad atoms cannot be linearly ordered; see Section 2.3.

In the next section, we show that extending ACL2 Version 2.5 with a total order on the bad atoms is equivalent to extending it with a total order on the ACL2 universe. In the same section, we also discuss total orders from a set-theoretic point of view. We then consider three example applications where the existence of a total order is used to simplify matters. In Section 3, we define sorting functions that satisfy the above properties. In Section 4 we show how to simplify J Moore's finite set theory book [12]. In Section 5, we show how to simplify a book on records due to Kaufmann and Sumners [14, 10]. Section 6 contains the conclusions. The full proofs can be found in the supporting books, which are freely available. In this paper we often omit definitions, theorems, and parts of events (such as the :rule-classes portion of theorems) for presentation purposes.

2 Total Orders

We show that extending ACL2 Version 2.5 with a total order on the bad atoms is equivalent to extending ACL2 with a total order on the ACL2 universe. The full ACL2 proofs are in the book total-order, which is written for ACL2 Version 2.5. We also discuss total order from a set-theoretic point of view.

2.1 A Total Order for ACL2

We start by defining bad-atom.

(defun bad-atom (x)
  (not (or (consp x)
           (ac12-numberp x)
           (symbolp x)
           (characterp x)
           (stringp x))))

We now use defaxiom to introduce a total order. A relation $\preceq$ is a total order if it satisfies the following.

1. $x \preceq x$
2. $x \preceq y \land y \preceq x \Rightarrow x = y$
3. $x \preceq y \land y \preceq z \Rightarrow x \preceq z$
4. $x \preceq y \lor y \preceq x$

A partial order is one satisfying the first three conditions, reflexivity, antisymmetry, and transitivity, respectively. Notice that reflexivity follows from totality, the fourth condition. In the ACL2 logic, we have the following definitions.
(defstub bad-atom<= (* *) => *)

(defmacro boolp (x)
  'or (equal ,x t)
    (equal ,x nil)))

(defun bad-atom<= (boolp (bad-atom<= x y)))

(defaxiom bad-atom<=-antisymmetric
  (implies (and (bad-atom x)
                 (bad-atom y)
                 (bad-atom<= x y)
                 (bad-atom<= y x)
                 (equal x y)))

(defaxiom bad-atom<=-transitive
  (implies (and (bad-atom<= x y)
                 (bad-atom<= y z)
                 (bad-atom<= x z))

(defaxiom bad-atom<=-total
  (implies (and (bad-atom x)
                 (bad-atom y))
           (or (bad-atom<= x y)
                (bad-atom<= y x)))

We say that function f is a total order on objects recognized by o if f satisfies the properties obtained by replacing bad-atom with o and bad-atom<= by f in the four preceding properties.

We use bad-atom<= to define atom-order, a total order on atoms by using <= to compare numbers, char-code to compare characters, string<= to compare strings, symbol-< to compare symbols and bad-atom<= to compare everything else. Moreover, in this order numbers precede characters, which precede strings, which precede symbols, which precede bad atoms. The definition follows.

(defun atom-order (x y)
  (cond ((rationalp x)
          (if (rationalp y)
              (<= x y)
              t))
        ((rationalp y) nil)
        ((complex-rationalp x)
         ((complex-rationalp y)
          (if (complex-rationalp y)
(or (< (realpart x) (realpart y))
   (and (= (realpart x) (realpart y))
       (<= (imagpart x) (imagpart y))))

((complex-rationalp y)
 nil)
((characterp x)
 (if (characterp y)
     (<= (char-code x)
         (char-code y)))
     t))
((characterp y) nil)
((stringp x)
 (if (stringp y)
     (and (string< x y) t)
     t))
((stringp y) nil)
((symbolp x)
 (if (symbolp y)
     (not (symbol<- y x))
     t))
((symbolp y) nil)
(t (bad-atom< x y))))

With the help of a few lemmas, we have that atom-order is a total order on atoms.

We now define a total order on the ACL2 universe as follows.

(defun total-order (x y)
  (cond ((atom x)
          (cond ((atom y)
                  (atom-order x y))
                (t t)))
        ((atom y) nil)
        ((equal (car x) (car y))
         (total-order (cdr x) (cdr y)))
        (t (total-order (car x) (car y)))))

It follows directly from the definition of a total order (at the beginning of this section, and in book total-order in the supporting materials) that total-order is a total order on the ACL2 universe. We have shown that adding a total order to the bad atoms in ACL2 implies that there is a total order on the ACL2 universe. The other direction is straightforward and is given in the supporting book total-order-easy-direction.
2.2 Soundness

We have seen that adding axioms to ACL2 positing the existence of a total order on the whole of the ACL2 universe is equivalent to adding axioms positing the existence of a total order on the bad atoms. An important question is whether adding such axioms preserves soundness.

Soundness is preserved, as we now show. First, note that if the ACL2 axioms are sound, there are models with no bad atoms: although the axioms do not close the universe, neither do they populate it with bad atoms. Thus, if we can show that there is a total order on the ACL2 universe without bad atoms, we are done. A proof of this can be found in the supporting book soundness.

We define a good atom as follows.

\[(\text{defun good-atom} \ (x)) \]
\[(\text{and} \ (\text{atom} \ x)) \]
\[(\text{or} \ (\text{acl2-numberp} \ x)) \]
\[(\text{symbolp} \ x) \]
\[(\text{characterp} \ x) \]
\[(\text{stringp} \ x))])\]

A predicate to recognize good objects, objects in the ACL2 universe without bad atoms (i.e., objects built from good-atoms), follows.

\[(\text{defun good-object} \ (x)) \]
\[(\text{if} \ (\text{consp} \ x)) \]
\[(\text{and} \ (\text{good-object} \ (\text{car} \ x)) \]
\[(\text{good-object} \ (\text{cdr} \ x)))]
\[(\text{good-atom} \ x))])\]

In the book soundness we exhibit a total ordering on the good objects. The definitions follow closely those in the previous section, thus we do not present them here.

Finally, we note that a somewhat stronger property holds for the axioms on total order, namely, conservativity: No additional theorems are provable from any ACL2 Version 2.5 books not involving the total order on bad atoms when the axioms on bad-atom are added. This is immediate from the lemma immediately preceding the theorem in Appendix B of [9], which in fact guarantees that it is conservative to add a bijection of the ACL2 universe with the natural numbers.

2.3 Total Orders From a Set-Theoretic Perspective

In this section we review some well-know facts about total orders in a purely set-theoretic setting. Good references include the books by Kunen, Devlin, and Halmos [11, 3, 4] and Part B of the Handbook of Mathematical Logic [1], especially chapter B.2 on the Axiom of Choice [5]. This section can be skipped without impacting readability of the rest of this paper.

Recall the Axiom of Choice, which (among many equivalent formulations) can be stated as follows: every set can be well-ordered. Thus, in ZFC (Zermelo-Frankel set theory with the Axiom of Choice), every set can be totally ordered.
In ZF (ZFC without the Axiom of Choice), one cannot prove that every set can be totally (linearly) ordered. One needs further axioms, but not the full power of the Axiom of Choice. For example, the Boolean Prime Ideal Theorem, a weaker statement than the Axiom of Choice, will do. The Boolean Prime Ideal Theorem (every Boolean algebra has a prime ideal) is equivalent to the Compactness Theorem of model theory: If every finite subset of $S$, a set of first-order sentences, has a model, then $S$ has a model. From the compactness theorem one can prove that every set can be totally ordered.

Define $C_n$, for $n$ a positive integer, to be the following statement.

If $F$ is a family of sets each of which has exactly $n$ elements, then $F$ has a choice function.

$C_2$ is not provable in ZF. An instructive example of this is due to Russell. Consider an infinite set of pairs of shoes. A choice function (a function that chooses one shoe from each pair) is easy to find, e.g., always choosing the right shoe will do. In contrast, consider an infinite set of pairs of socks (where there is no distinction between the socks in a pair). It does not seem possible to construct a choice function, as there is no way to simultaneously choose one of the two socks for each pair. Tarski initiated the investigation of the $C_n$ statements and showed that $C_2$ implies $C_4$. There have been various papers clarifying the relationships between combinations of $C_n$'s, e.g., it is known that $C_2$ implies $C_n$ is provable iff $n$ is $1, 2,$ or $4$.

We raised the $C_n$ problem because there is an obvious parallel between the socks problem and the problem mentioned in the introduction, namely the problem of sorting lists of length two. In both cases, we are required to find a choice function on a collection of pairs, where there is no way to distinguish between elements in a pair.

### 2.4 The Total Order in ACL2 Version 2.6

We have justified adding axioms to ACL2 positing the existence of a total order. We now give an overview of the total order provided with ACL2 Version 2.6. The file `axioms.lisp` contains the definition of function `bad-atom` and axioms positing that `bad-atom<=` is a total order on the bad atoms, as above. The function `alphorder` is defined and shown to be a total order on atoms. The definition of `alphorder` is similar to our definition of `atom-order`. Finally, `lexorder`, a function corresponding to our `total-order`, is defined and shown to be a total order on the ACL2 universe. In directory `books/misc`, the ACL2 distribution for Version 2.6 contains the book `total-order`, which defines an order `<` based on `lexorder`: `(< x y)` if and only if $x$ and $y$ are not equal and `(lexorder x y). Having justified the addition of new axioms positing the existence of a total order, we will now use the Version 2.6 books in the sequel.
3 Sorting

We show how the new total ordering axioms can be used to sort arbitrary ACL2 objects. We then define insertion sort and quicksort and show that they are equal.

Our approach is in line with ACL2 tradition: we define total functions with an intended domain, generally the set of true (null-terminated) lists, while keeping the definitions simple by avoiding any reference to the intended domain. For the most part, we can prove theorems that do not contain assumptions about the intended domain (i.e., true-listp hypotheses). This leads to simpler rewrite rules (see [7] for a full explanation).

3.1 Permutations and Ordered Lists

We start by defining `perm`, a function that checks if its arguments are permutations, and related theorems in the book `perm`.

```lisp
(defun in (a x)
  (cond ((atom x) nil)
        ((equal a (car x)) t)
        (t (in a (cdr x)))))

(defun remove-el (a x)
  (cond ((atom x) nil)
        ((equal a (car x)) (cdr x))
        (t (cons (car x) (remove-el (cdr x))))))

(defun perm (x y)
  (cond ((atom x) (atom y))
        (t (and (in (car x) y)
                (perm (cdr x) (remove-el (car x) y)))))))

We prove that `perm` is an equivalence relation.

(defun equiv perm)

This allows us to use congruence-based reasoning and we prove various congruence rules, including the following.¹

(defun cong perm perm (append x y) 1)
(defun cong perm perm (append x y) 2)
(defun cong perm perm (cons x y) 2)
(defun cong perm equal (in x y) 2)

In the book `perm-order`, we define the function `orderedp` which checks if its argument is ordered, using the total order `lexorder`.

(defun <= (x y) (lexord x y))

¹ Close analogues of these results are proved with the built-in ACL2 function `mini-proveall`. 
(defun orderedp (x)
  (cond ((atom x) (null x))
    (t (or (null (cdr x))
      (and (<= (car x) (cadr x))
        (orderedp (cdr x))))))

3.2 Insertion Sort

In the book insertion-sort, we define insertion sort as follows.

(defun insert (a x)
  (if (consp x)
    (if (<= a (car x))
      (cons a x)
      (cons (car x) (insert a (cdr x))))
    (list a)))

(defun isort (x)
  (if (consp x)
    (insert (car x) (isort (cdr x)))
    nil))

We prove the usual theorems, taking advantage of the total order to eliminate
the need for hypotheses that restrict the types of the list elements.

(defun ordered-sort
  (orderedp (isort x)))

(defun perm-sort
  (perm (isort x) x))

We also prove the following theorem, thereby showing how to solve the problem
we raised in the introduction.

(defun main
  (equal (perm x y)
    (equal (isort x)
      (isort y)))))

Two related theorems, showing that any ordered permutation of an object
equals the isort of the object and that ordered permutations are equal follow.

(defun main2
  (implies (and (orderedp x)
    (perm x y))
    (equal (isort y)
      x)))
(defthm main3
  (implies (and (orderedp x)
                 (orderedp y)
                 (perm x y))
           (equal x y)))

3.3 Quicksort

In the book quicksort, we define quicksort as follows.

(defun less (x lst)
  (cond ((atom lst) nil)
       ((< (car lst) x)
        (cons (car lst) (less x (cdr lst))))
       (t (less x (cdr lst)))))

(defun notless (x lst)
  (cond ((atom lst) nil)
       ((not (< (car lst) x))
        (cons (car lst) (notless x (cdr lst))))
       (t (notless x (cdr lst)))))

(defun qsort (x)
  (cond ((atom x) nil)
       (t (append (qsort (less (car x) (cdr x)))
                  (list (car x))
                  (qsort (notless (car x) (cdr x)))))))

Besides proving the obvious theorems,

(defun qsort-is-ordered
  (orderedp (qsort x))
(defthm perm-qsort
  (perm (qsort x) x))

we also prove the following theorem, with the help of theorem main2, which
relates quicksort and insertion sort.

(defun qsort-main
  (equal (qsort x)
         (isort x))
  :hints ("goal" :use (:instance main2 (x (qsort x)) (y x))))

A referee has noted that the use of a total order on the ACL2 universe can
lead to code that is computationally less efficient than the use of an order on
the intended domain, for example in sorting. As the referee also points out, this
need not be a troublesome issue: it would be straightforward to define two sets
of functions, one using the total order on the ACL2 universe and one using
the domain-specific order, and then prove their correspondence on the intended
domain. Then one can happily use the first set of functions for reasoning and the second set of functions for execution.

4 Set Theory

Various efforts to design ACL2 books on set theory have led researchers to define a function, call it \( c \), to put sets into a canonical form. Ideally, \( c \) should have the property that \( X \) and \( Y \) are equal as sets iff \( \text{equal} (c \ X) (c \ Y) \). It is not possible to define such a function without restricting the set of objects that can be put into canonical form. (See the discussion of pairs of socks in Section 2.3.) In this section, we examine J Moore's book on finite set theory [12] and show how it can be simplified with the use of a total order.

J Moore's books on set theory, which can also be found in the ACL2 distribution in directory books/finite-set-theory/, are included in the supporting books. They are total-ordering-original and set-theory-original. The books total-ordering and set-theory, also part of the supporting books, are based on Moore's books, but use the total order. We now discuss the differences.

The original books include a definition of "ordinary" objects and "standard" atoms.

```lisp
(defun ordinaryp (x)
  (cond ((atom x)
         (or (acl2-numberp x)
             (characterp x)
             (stringp x)
             (symbolp x)))
       (t (and (consp x)
                (ordinaryp (car x))
                (ordinaryp (cdr x))))))

(defun standard-atom (x)
  (or (acl2-numberp x)
      (characterp x)
      (stringp x)
      (symbolp x)))
```

A binary function, \(<<\), is defined and shown to be a total order on ordinary objects. In addition, various functions, including o-fix, a function which converts objects into ordinary objects, are used to define canonicalize, a function that puts ordinary objects into canonical form, and canonicalp, a function that recognizes canonical objects.

With our books, the functions ordinaryp, standard-atom, and o-fix are not required and all of the theorems and definitions that depend on them can be elided or simplified. For example, the theorem
(defthm ordinaryp-set-insert
  (implies (and (ordinaryp e)
                (ordinaryp x))
            (ordinaryp (set-insert e x))))

appears in the original books but is not needed in our books. The theorem

(defthm <<-cons-1
  (implies (and (ordinaryp x)
                (ordinaryp y))
            (<< x (cons x y))))

which gives rise to a conditional rewrite rule is simplified to the following un-
conditional rewrite rule in our books.

(defthm <<-cons-1
  (<< x (cons x y)))

The conceptual simplifications result in better certification times. In one pair
of tests the new books required 3 minutes and 4.9 seconds of user time to certify
whereas the original books required 4 minutes and 21.0 seconds.

5 Records

Many different approaches to dealing with records have appeared in the work
of various ACL2 users. This is not surprising, as records play an important role
in programming. One approach is to use alists, where the car of an element
indicates the field and the cdr indicates the value. For big projects such as the
FM9801 verification effort by Sawada [13] the structures book by Bishop Brock
has been found useful [2]. At the first ACL2 Workshop, Ken Albin gave a talk
where he reviewed some of the approaches to memory modeling and requested
that the community come up with a nice solution.

One problem from which all of the above approaches suffer is the complexity
of the rewrite rules obtained. For example, when using alists one often uses
congruence-based reasoning because there are many ways to represent the same
record: permutations of the alist tend to represent the same record.

A very promising approach to dealing with records in ACL2 is due to Kauf-
mann and Sumners. They developed a book on records with very simple rewrite
rules: specifically, there are no hypotheses that require objects to be well-formed
records. Here are the main theorems proved, where s is a function to set a field
in a record and g is a function to the get the value of a field in a record. The
complete proof scripts are from [14] and are in the book records-original
which is part of the supporting materials.

(defthm s-same-g
  (implies (force (fieldp a))
            (equal (s a (g a r) r)
                  r)))
(defthm g-same-s
  (implies (force (fieldp a))
    (equal (g a (s a v r)) (v))))

(defthm s-same-s
  (implies (force (fieldp a))
    (equal (s a y (s a x r)) (s a y r)))))

(defthm s-diff-s
  (implies (and (force (fieldp a))
                 (force (fieldp b))
                 (not (equal a b)))
    (equal (s b y (s a x r)) (s a x (s b y r))))
  :rule-classes (:(rewrite :loop-stopper ((b a s)))))

(defthm g-diff-s
  (implies (and (force (fieldp a))
                 (force (fieldp b))
                 (not (equal a b)))
    (equal (g a (s b v r)) (g a r)))
  :rule-classes (:(rewrite :loop-stopper ((b a s)))))

We do not give the definitions of s and g here, as they are complicated and not germane to this paper. The interesting thing about this approach is that the focus is on the algebraic properties of the functions in question. Even though the definitions of s and g are messy, it does not matter because the rewrite rules obtained are very simple and when proving theorems we only use the rewrite rules, as the definitions of s and g are disabled.

One drawback is the need for hypotheses about objects being fields. The reason for this is no surprise. We need a way to order objects and since we cannot order bad atoms, we are forced to choose a subset of the ACL2 universe and to require that objects are in this class, by adding hypotheses to our rewrite rules. When we subsequently decide to use some other type of field, we have to revise the book. For example, originally fields were symbols, then they became symbols or integers, and one can imagine many definitions of “field”. The most general hypothesis is probably that they are good objects as defined in the set theory section, above. Using the total order we can now prove the following theorems. The full proof scripts are in the supporting book records. Also, records books that take advantage of the total order are described in [10].

(defthm s-same-g
  (equal (s a (g a r) r) (r)))
  :rule-classes (:(rewrite :loop-stopper ((b a s)))))
(defthm g-same-s
  (equal (g a (s a v r)))
  v))

(defthm s-same-s
  (equal (s a y (s a x r))
          (s a y r)))

(defthm s-diff-s
  (implies (not (equal a b))
            (equal (s b y (s a x r))
                   (s a x (s b y r))))
 :rule-classes ((:rewrite :loop-stopper ((b a s)))))

(defthm g-diff-s
  (implies (not (equal a b))
            (equal (g a (s b v r))
                   (g a r))))

As before, we get a non-trivial reduction in the both the size of the resulting books and the certification time. The size of the original records book is 382 lines, whereas our book is 288 lines. In addition, the certification time was reduced from 74 seconds to 33 seconds. Perhaps more important is the elimination of hypotheses, resulting in more efficient rewrite rules.

6 Conclusion

We presented new ACL2 axioms positing the existence of a total order on the ACL2 universe. These axioms have been added to ACL2 Version 2.6. We showed that a total order often simplifies ACL2 books and leads to simpler and more efficient rewrite rules. We applied the ideas to develop books on sorting and to simplify existing books on finite set theory and records.

Acknowledgements

We received many useful comments from Rob Sumners and the rest of the Austin ACL2 group. We also thank the referees for their useful observations, which we believe have improved the final presentation.

References