Typed ACL2 Records

David Greve and Matthew Wilding

Rockwell Collins Advanced Technology Center
Cedar Rapids, IA 52498 USA

{dagreve, mmwildin}@rockwellcollins.com

Abstract

We show a macro for introducing operations on typed records. The underlying theorems proved about these records include what is proved about records introduced using the standard ACL2 record book [2], as well as an additional theorem about the type of the elements.

1 Background

The standard ACL2 distribution contains a "records" book that provides an unconventional implementation of two functions: `g` (for "get") and `s` (for "set"). The implementation allows for the proof of a simple set of theorems that are useful for reasoning about records. [1]

1. `(equal (g a (s a v r)) v))`
2. ```
   (implies (not (equal a b))
     (equal (g a (s b v r)) (g a r))))
   ```
3. `(equal (s a (g a r) r) r))`
4. `(equal (s a y (s a x r)) (s a y r)))`
5. ```
   (implies (not (equal a b))
     (equal (s b y (s a x r)) (s a x (s b y r))))
   ```

[1] Rob Sumners first suggested a records book with easy-to-use properties in discussions with Matt Kaufmann. Kaufmann solved the problem, as did others (including the authors) in response to his issuing a challenge to the ACL2 list. Kaufmann and Sumners ultimately created a version that exploits the total order [3] added to ACL2 2.6, which is the version that is distributed with ACL2 [2].

These theorems have the desirable property that there are few hypotheses. In particular, nothing need be established about the record structure for these theorems to hold, which simplifies proofs involving `g` and `s`. It is not at all obvious that there exists an implementation of `g` and `s` such that these theorems hold. For example, note that a straightforward implementation of these functions using association list functions fails to satisfy property 5. It's hard to define functions that satisfy all of these properties with a single implementation!

2 A Modest Extension

We have found this formulation of records useful for modeling machine state for models such as [1]. However, we discovered in late 2002 that we sometimes need yet another theorem to hold of records: that the value returned by `g` is of an appropriate type. That is, we require that

6. `(typep (g a r))`

for a `typep` predicate provided by us. Note that this fact is unconditional and therefore easy to use in proofs about functions that are defined in terms of record operations. However, as with the standard record implementation, it's not obvious that there exists an implementation that has all these properties, and an implementation satisfying each of the properties 1-6 was not initially apparent to us. Note for example that a straightforward implementation that uses the "standard" records book with the modification that `g` "fixes" values not of the record type fails to satisfy property 3.

The file associated with this paper contains an extension to the ACL2 records book that satisfies these requirements. We introduce the macro `defrecord` to define accessor and updater functions for a record structure with elements of a particular type. The macro also introduces the needed theorems for this datastructure, which includes theorems that are similar to the theorems introduced in the standard ACL2 records book as well as an additional fact related to the user-defined type function.

For example, the following sequence:

```
(defun sbp16 (x)
  (declare (xargs :guard t))
  (signed-byte-p 16 x))

(defun fix-sbp16 (x)
  (declare (xargs :guard t))
  (if (sbp16 x) x 0))
```

```
(defrecord sbp :rd getbv :wr putbv :fix fix-sbp16 :typep sbp16)
```

introduces a "get" function `getbv` and the "set" function `putbv` that implement those operations on the record. The user also provides a "fix" function that coerces its argument to a user-desired type and a "typep" predicate that identifies values of the right type. The macro introduces theorems that correspond to the theorems of the standard ACL2 records books (with proper operation names) except in two particulars.

- The "get-of-set" theorem of the ACL2 standard library (number 1 in the list above) is modified somewhat. A similar theorem reflects the fact that the values of the record are of the user-specified type. As an example, the invocation of `defrecord` above introduces the theorem

  ```
  (defthm getbv-same-putbv-hyps
     (implies (equal a b)
       (equal (getbv a (putbv b v r))
              (fix-sbp16 v))))
  ```

- `defrecord` introduces the desired type theorem. The invocation of `defrecord` above introduces the theorem

  ```
  (defthm sbp16-getbv (sbp16 (getbv a r)))
  ```

The implementations of the functions generated by this macro are obscure, but the approach employed to enable hypothesis-free type rules are similar to those used to guarantee hypothesis-free access and update rules.

The `defrecord` macro relies upon the currently-distributed records book documented in [3]. We believe that the macro generates theorems that prove automatically for all sensible parameters assuming that the appropriate functions are enabled properly. The theorems prove automatically on all the examples to which we have applied it.

It is easy to prove simple properties about operations on records that have been introduced using `defrecord`. For example, the following proves quickly:

```
(defun swap (a1 a2 rec)
  (putbv a1 (getbv a2 rec) (putbv a2 (getbv a1 rec) rec)))

(defthm swap-swap
  (equal
   (getbv addr (swap a1 a2 (swap a1 a2 x)))
   (getbv addr x))
  :hints (("goal" :in-theory (enable getbv-of-putbv-redux))))
```

The `swap-swap` theorem could be proved about a record using the untyped records of the standard ACL2 distribution. However, `defrecord` also provides type information that can be useful. For example,

```
(defun addloc (l1 l2 rec)
  (+ (getbv l1 rec) (getbv l2 rec)))

(verify-guards addloc)
```

The guard of the `+` macro requires that the arguments be numeric, which they are known to be because of the type theorem.

3 Conclusion

The `defrecord` macro provides a convenient method by which to add conventional record access functions and provides theorems we find useful for reasoning about operations on those records.

References

[1] David Greve, Matthew Wilding, and David Hardin. High-speed, analyzable simulators. In *Computer-Aided Reasoning: ACL2 Case Studies.* Kluwer Academic Publishers, 2000. Also `http://hokiepokie.org/docs`.

[2] M. Kaufmann and R. Sumners. Efficient rewriting of operations on finite structures in ACL2. In *Proceedings of the Third International Workshop on the ACL2 Theorem Prover and Its Applications*, Grenoble, France, April 2002.

[3] Panagiotis Manolios and Matt Kaufmann. Adding a total order to ACL2. In *Proceedings of the Third International Workshop on the ACL2 Theorem Prover and Its Applications*, Grenoble, France, April 2002.