Enhancements to ACL2 in Versions 5.0, 6.0, and 6.1

Matt Kaufmann
Dept. of Computer Science,
University of Texas at Austin
kaufmann@cs.utexas.edu

J Strother Moore
Dept. of Computer Science,
University of Texas at Austin
moore@cs.utexas.edu

We report on highlights of the ACL2 enhancements introduced in ACL2 releases since the 2011 ACL2 Workshop. Although many enhancements are critical for soundness or robustness, we focus in this paper on those improvements that could benefit users who are aware of them, but that might not be discovered in everyday practice.

1 Introduction

This paper discusses ACL2 enhancements introduced in releases made since the ACL2 Workshop in November, 2011: Versions 5.0 (August, 2012), 6.0 (December, 2012), and 6.1 (expected February, 2013). We thus discuss enhancements made after the release of ACL2 Version 4.3 in July, 2011.

The release notes [3] for those three versions report approximately 200 enhancements, which typically were made in direct response to user feedback or were important to soundness or robustness of the system. Our goal in this paper is not simply to rehash the release notes; rather, it is to highlight important improvements that ACL2 users are not likely to discover by the routine use of ACL2. We do not discuss lower-level improvements to the system that are reported in comments in source file `ld.lisp` for the release notes (e.g., `(deflabel note-5-0 ...)`). Those who dive into the ACL2 sources may wish to peruse these; for example, they will notice that starting in ACL2 6.0, `defrec` defines a recognizer predicate.

Because of the maturity of ACL2, many of the improvements pertain to aspects of ACL2 that may be unfamiliar to novice users. Our hope, however, is that this paper will have value to those users as well, by suggesting new ideas about what can be done with ACL2.

As in a preceding paper of a similar nature in the previous ACL2 workshop [5], we write "see :DOC" to highlight documentation topics. For example, see :DOC release-notes and its subtopics (e.g., see :DOC note-6-0 for changes introduced in ACL2 Version 6.0). Documentation topics are also referenced implicitly using underlining; for example, the topic <u>advanced-features</u> provides a handy summary of advanced features of ACL2 in one place. Each documentation topic reference (of either type) is a hyperlink in the online version of this paper.

Unlike the preceding paper mentioned above, we choose here to organize the paper in the way that we have organized the release notes for several years, as follows.

- Changes to existing features
- New features
- Heuristic improvements
- Bug fixes
- Changes at the system level

To appear in EPTCS.

In each of the five sections corresponding to these topics, we present a few topics in some detail, but in many cases we simply note an improvement and point to relevant documentation.

There are typically two other categories: Emacs support and Experimental/alternate versions. The former has changed little in the last few release notes. As for the latter, there have been significant improvements to ACL2(h), ACL2(p), and ACL2(r); but for these we live within our space limitations, referring readers to the release notes.

One outlier, not included in the categories above, is a series of changes related to licensing and distribution. For Version 5.0, changes were made to satisfy University of Texas policies: the license changed from GPL "Version 2 or later" to GPL Version 2, and the community books [1] — basically, what has been called the regression suite — were moved away from the University of Texas, and are hosted by Google Code. For Version 6.0 we changed the license to a BSD-style license, in order to make it easier for industry groups to take advantage of ACL2.

Acknowledgements

We thank members of the ACL2 community whose feedback has led us to continue making improvements to ACL2, including the following, each mentioned for one or more specific items in the release notes for Version 5.0, 6.0, or 6.1: Harsh Raju Chamarthi, Jared Davis, Ruben Gamboa, Shilpi Goel, Dave Greve, David Hardin, Marijn Heule, Warren Hunt, Anthony Knape, Robert Krug, Camm Maguire, Pete Manolios, Francisco J. Martin Mateos, David Rager, Jose Luis Ruiz-Reina, Anna Slobodova, Eric Smith, Rob Sumners, Sol Swords, Sarah Weissman, and Nathan Wetzler. We expressly thank Warren Hunt for his continued support of the use of ACL2, in particular in projects at the University of Texas. Finally, we thank the reviewers for helpful comments, one of which led us to improve :DOC provisional-certification for the next release.

This material is based upon work supported by DARPA under Contract No. N66001-10-2-4087, by ForrestHunt, Inc., and by the National Science Foundation under Grant Nos. CCF-0945316 and CNS-0910913.

2 Changes to existing features

There are over 50 release note items about changes to existing features. Here we list a few and then present a few others in a bit more detail.

- Functions `READ-ACL2-ORACLE`, `READ-RUN-TIME`, `GET-TIMER`, and `MAIN-TIMER` are no longer untouchable; you can call them in your programs.
- vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
  Macros can take an argument named `STATE`, or which is the name of a stobj. However, these variables are not bound to the "live objects" as you might expect but are treated just like other macro variables.
- The macros `MEMOIZE` and `UNMEMOIZE` now cause a warning rather than an error in (regular, non-HONS) ACL2.
- The macro `DEFUND` may now be used without error with `:PROGRAM` mode specified in an `XARGS` declaration.
- The functions `SYS-CALL` and `SYS-CALL-STATUS` are now guard-verified `:LOGIC` mode functions.

- The environment variable ACL2_COMPILE_FLG provides a default for CERTIFY-BOOK; it was formerly named COMPILE_FLG.

Some other changes

It has been the case since Version 3.6 (August, 2009) that the definition of a function symbol can mention that symbol in the guard and measure. Now, guards specified in ENCAPSULATE signatures may similarly refer to the functions being introduced in the same ENCAPSULATE event.

Some utilities have been improved, so you might want to try them again even if you gave up on them in the past. For example, consider :PL applied to a non-symbol. It didn't work for macro calls, but now it performs macroexpansion (and other transformations to internal form) as a first step; and moreover, among the rule classes that it shows is now the :LINEAR class. Another utility that has been improved is TOP-LEVEL, which no longer causes calls of LD to stop.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

The "with-error-trace" utility, WET, has also been improved. Finally, if you haven't yet tried DEFATTACH, because your code seemed to run a bit slowly using attachments, consider trying again, as efficiency has improved for this utility.

The abbreviated proof output offered by *gag-mode* is now on by default. See :DOC SET-GAG-MODE for a description of gag-mode. If you want a bit of control over the printing of induction schemes and guard conjectures in gag-mode, see the discussion of :GAG-MODE in :DOC SET-EVISC-TUPLE.

For a macro mac, you can now add a pair (mac . fn) to the MACRO-ALIASES-TABLE even when fn has not been defined as a function symbol. This can be useful if you want to define a set of macros early. See :DOC ADD-MACRO-ALIAS.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

When functions such as FMT-TO-STRING (see :DOC printing-to-strings) was introduced in Version 4.3, it printed with a right margin set to 10,000, but now the default right margin settings are used. Thus, for example, the string returned as shown below had no newline characters in Version 4.3. We can return to the default behavior as shown.

```
ACL2 !>(fmt-to-string "~x0"
                      (list (cons #\0 (make-list 20))))
(0
 "
(NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
     NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
")
ACL2 !>(fmt-to-string "~x0"
                      (list (cons #\0 (make-list 20)))
                      :fmt-control-alist
                      `((fmt-soft-right-margin . 10000)
                        (fmt-hard-right-margin . 10000)))
(81
 "
(NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)")
ACL2 !>
```

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

The extended metafunctions have been reworked, with improved handling of forcing and also with the option of returning a tag-tree. Also, a unifying substitution has been added to metafunction contexts, accessed with function MFC-UNIFY-SUBST. See :DOC extended-metafunctions).

Printing of numbers now pays attention to the print radix; see :DOC SET-PRINT-RADIX. For example, before Version 6.0 the final value was printed below as ABCD1234. Notice the use of #u to allow underscores in numbers, which is new.

```
ACL2 !>(set-print-base 16 state)
<state>
ACL2 !>(set-print-radix t state)
<state>
ACL2 !>#uxabcd_1234
#xABCD1234
ACL2 !>
```

3 New features

Of the approximately 50 release note items about new features, we list a few here and then elaborate on a few others below.

- See :DOC PRINT-SUMMARY-USER for a way to add to what is printed in event summaries.
- Commands :PL and :PR now have analogues in the proof-checker.
- See :DOC provisional-certification for how to certify books in parallel even when they they are ordered linearly by INCLUDE-BOOK.
- ACL2 now supports multiple instances of a stobj (whether conventional or abstract), known as *congruent stobjs*. See :DOC DEFSTOBJ and see :DOC DEFABSSTOBJ.
- Access to the host Lisp's disassembler is now provided in the ACL2 loop by the DISASSEMBLE$ utility.
- See :DOC DEFTHEORY-STATIC for a variant of DEFTHEORY that behaves the same when a book containing such an event is included, as it does when when the book was certified.
- See :DOC :PSOF for a variant of :PSO that directs proof output hidden by gag-mode to a file. Also see :DOC WOF for a general utility for directing output to a file.
- A new macro, DEFND is just DEFN (i.e., DEFUN with a guard of T) plus a disable just as DEFUND is DEFUN plus a DISABLE.
- New utilities ORACLE-FUNCALL, ORACLE-APPLY, and ORACLE-APPLY-RAW, provide a sort of higher-order capability, by calling a function argument on specified arguments.
- Both INLINE and NOTINLINE declarations are now supported for the FLET utility.
- See :DOC GC-VERBOSE for how to control, in some host Lisps, the printing of garbage-collection messages.
- The utility ADD-MACRO-FN, which is a replacement for ADD-BINOP, lets you choose whether macros are to be displayed as flat right-associated calls, for example, (append x y z) rather than (append x (append y z)).

- The new TIME-TRACKER utility supports annotating your programs to display information during a computation about elapsed runtime.
- The *tau system* is discussed in Section 4.

Some other new features

The utility DEFUN-NX has been improved, for example by avoiding stobj restrictions in the :LOGIC component of an MBE call. Here is an example from Jared Davis that motivated this change; note the call of function MY-IDENTITY on a stobj even though MY-IDENTITY was not declared to take a stobj argument.

```
(defstobj foo (fld))
(defun-nx my-identity (x) x)
(defun my-fld (foo)
  (declare (xargs :stobjs foo))
  (mbe :logic (my-identity foo)
       :exec (let ((val (fld foo)))
               (update-fld val foo))))
```

But there now is another way to violate signatures in non-executable code: by using the utility, NON-EXEC. Note that this time, MY-IDENTITY is defined with DEFN (which is DEFUN with a guard of T), not by DEFUN-NX.

```
(defstobj foo (fld))
(defn my-identity (x) x)
(defun my-fld (foo)
  (declare (xargs :stobjs foo))
  (non-exec (my-identity foo)))
```

There have been many improvements to the documentation, but here we focus on two new topics. The topic advanced-features summarizes some cool features of ACL2 that might not all be widely known, yet may be of interest, especially to experienced users. Another new topic provides a guide to programming with the ACL2 state; see :DOC programming-with-state.

A new event, DEFABSSTOBJ, provides an interface to conventional single-threaded objects known as *abstract stobjs* [2]. These can provide advantages over conventional stobjs in several arenas: execution speed, proof efficiency, use of symbolic simulation, and modularity of proof development.

ACL2 now provides a way to direct the host Lisp compiler to inline calls of a given function. See :DOC DEFUN-INLINE. We expect that you can generally use this utility just as you would use DEFUN to define a function. However, we say a bit more, in part to motivate our design of this utility. Fundamentally, DEFUN-INLINE is simply a macro, as we illustrate by expanding a call of this macro.

```
ACL2 !>:trans1 (defun-inline f (x)
                 (declare (xargs :guard (consp x)))
                 (integerp (car x)))
 (PROGN (DEFMACRO F (X) (LIST 'F$INLINE X))
        (ADD-MACRO-FN F F$INLINE)
        (DEFUN F$INLINE (X)
               (DECLARE (XARGS :GUARD (CONSP X)))
               (INTEGERP (CAR X))))
ACL2 !>
```

Notice that F is defined to be a macro whose calls expands to a corresponding calls of a function, F$INLINE. The invocation of ADD-MACRO-FN arranges that theory functions understand F to mean F$INLINE and that proof output will display calls of F$INLINE as corresponding calls of F. But why didn't we simply support the Common Lisp form (declaim (inline f))? The reason is the support that ACL2 provides for undoing. Imagine that you want F to be inline and then you change your mind — or maybe F is defined in a book that you include locally. How can we arrange for Common Lisp to undo the directive to inline calls of F? Sadly, the Common Lisp language [6] does not provide for a way to do that. The best we can do is to direct F to be notinline — but that could defeat the host Lisp's appropriate inlining of some subsequent definition of F. Our solution is *always* to direct inlining for functions whose name ends in the string "$INLINE", and to provide the illusion that we are defining a function F rather than F$INLINE, using ADD-MACRO-FN as discussed above. Note that analogous considerations hold for utility DEFUN-NOTINLINE.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

We invite the ACL2 community to help us to convert ACL2 system functions from :PROGRAM mode to guard-verified :LOGIC mode. This mechanism is described in some detail in an online document [4]. Here, in brief, are the steps to follow; we would be happy to provide more details leading to improvement of the online document.

1. Install a local copy of ACL2, and build it using make.
2. Develop a book that includes VERIFY-TERMINATION and VERIFY-GUARDS forms for one or more system functions. For simplicity we assume here that there is a single such function, FN.
3. When necessary, modify ACL2 definitions in your copy, for example by replacing some calls of NULL by corresponding calls of ENDP or by adding or modifying guard declarations. Rebuild your local copy of ACL2 using make.
4. Email us your ACL2 changes and your book, and we will do what is necessary in order to incorporate your book into the ACL2 community books [1] and your changes into the ACL2 sources.
5. Henceforth, the default build of ACL2 will accordingly mark FN as a guard-verified :LOGIC mode function.

We, the ACL2 developers, will check each release that such proofs still go through, using a build that leaves FN in :PROGRAM mode.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

If you have written :META rules or :CLAUSE-PROCESSOR rules, you may have been frustrated that your meta functions and clause processor functions could not assume the correctness of prover computations, for example as performed using MFC-TS (see :DOC extended-metafunctions). A new mechanism, designed with Sol Swords, now provides such a capability; see :DOC meta-extract). The community book clause-processors/meta-extract-simple-test.lisp provides illustrative examples.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

ACL2 rule names, or runes, form the basis of ACL2 theories. But runes do not take into account macro aliases for function symbols. For example, (:definition binary-append) is a rune, and you can use append in a theory expression to abbreviate the set of runes, {(:definition binary-append), (:induction binary-append)}; but you cannot use (:definition append) in a theory expression. Now, however, you can use (:d append) in a theory expression to designate the rune (:definition binary-append). There are four new such abbreviation mechanisms, as follows, where `symb` is a symbol and `symb'` is the macro-aliases dereference of `symb`; e.g., `binary-append` is the macro-aliases dereference of `append`, while `car` is the macro-aliases dereference of itself.

- `(:d symb . r)` designates the rune `(:definition symb' . r)`.
- `(:e symb . r)` designates the rune `(:executable-counterpart symb' . r)`.
- `(:i symb . r)` designates the rune `(:induction symb' . r)`.
- `(:t symb . r)` designates the rune `(:type-prescription symb' . r)`.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

Take a new look at ACL2 output when you have large case splits, which in the past could be difficult to debug. Now, "Splitter Notes" can help you locate sources of your case splits. See :DOC splitter.

4 Heuristic improvements

As ACL2 is a heuristic theorem prover, it orchestrates many techniques to support effective automation of reasoning. The large regression suite, contributed by many users over about 20 years, has helped to tune the prover heuristics so that they often need relatively little of our attention. However, we have made improvements since Version 5.0 that include avoidance of some rewriting loops, two strengthenings of type-set reasoning, and tweaks to the heuristics for automatically expanding recursive function calls during proofs by induction.

ACL2 now expands away calls of so-called *guard-holders* before storing induction schemes. These include `THE` as well as all calls of `RETURN-LAST`. The latter include `MBE`, `PROG2$`,and equality-variants — for example, a call of `MEMBER` expands to the corresponding call of `MEMBER-EQUAL`. Such expansion also occurs before storing constraints generated by `ENCAPSULATE` events.

We may think of the break-rewrite utility as a heuristic, since, when enabled, it chooses debugging information to display to the user. This utility had incurred significant overhead even when disabled, as it is by default. That has been fixed, resulting in elimination of more than 10% of the time required for an ACL2 regression.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

The remainder of this section discusses a feature introduced in Version 5.0 that contributes to the set of primary prover heuristics: the *tau system*. This system is a decision procedure designed to exploit previously proved theorems about monadic Boolean functions. The tau system was extended and improved in Versions 6.0 and 6.1.

The system mines all the axioms, definitions, and proved rules (of any rule class) relating Boolean function symbols of one argument. One might think of these function symbols as recognizing “soft types” such as `integerp`, `consp`, `alistp`, `n32-bit-numberp`, etc. The *tau* of a term is the set of all such recognizers known to hold of the value of the term. The tau of a term is typically computed in a context specifying the tau of other terms (typically including variables and subterms). For example, if an `IF` has the test `(integerp i)`, then when the tau of the true branch is computed, the variable `i` is known to have a tau that contains `integerp` and all the recognizers it is known to imply.

For purposes of the tau system, Boolean monadic functions are tracked, as are equalities and inequalities with constants. As of Version 6.1, the tau system was extended to track intervals. For example, the tau for a term might, in addition to saying that the value of the term is an integer (and thus also a rational and not a cons), lies in the interval between 0 and 15 but is not 3 or 7.

Of special importance are *signature rules* that allow the tau system to compute the tau of a function application by computing the tau of the actuals. Tau also tracks other forms of rules that relate the known predicates, and it allows signatures for the various values returned by multiple-value functions. The tau system also provides a way for the user to define, verify, and install "bounder" functions which can be used to compute an interval containing a function's output from the intervals containing its input.

It is possible to prove certain theorems by tau reasoning alone. Such formula are often, informally, thought of as being mere consequences of "type checking." The tau system is designed to recognize such formulas rapidly. It is thought the tau system, if properly "programmed" with rules, will be helpful in verifying guard conjectures.

The tau documentation has grown extensively since Version 5.0. We recommend that the interested reader see :DOC introduction-to-the-tau-system.

5 Bug fixes

We have continued to improve ACL2 by eliminating more than 50 bugs. In this section we mention only a few that may have the most effect on how people use ACL2.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

The time reports in event summaries have been much improved. As far as we know, they now accurately report runtime (cpu time). Of course, you can use the `TIME$`utility for reports of realtime and runtime that avoid the accounting done by ACL2.

The `FLET` construct no longer has any requirements for returning stobjs.

6 Changes at the system level

In this section we pick a few additions and improvements that are outside the realm of what one might normally think of as "ACL2 features".

The character encoding for reading from files — and for some host Lisps also for reading from the terminal — is now iso-8859-1, also known as latin-1. See :DOC character-encoding.

You can now build the ACL2 documentation locally (using `make DOC`). Previously, the graphics had been omitted when doing so.

If you want to run a parallel regression using 'make', you should now avoid the '`-j`' option. Instead, use `ACL2_JOBS=`*n* where *n* is the maximum number of jobs to run in parallel. This change is in support of including the `centaur/` books in such regressions. (Those books had formerly only been certified in regressions done for ACL2(h); see :DOC hons-and-memoization.) Note that you should still use '`-j`' if you are certifying books residing in a particular directory, rather than doing a full regression.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

The search button near the top of the ACL2 home page will lead you to two search utilities: one for the documentation, and one for the community books.

7 Conclusion

We have presented an outline of changes to ACL2 in Versions 5.0, 6.0, and 6.1. Our focus has been to describe changes that can affect one's daily use of ACL2 but might otherwise go unnoticed. Many more changes (close to 200 altogether) may be found in the vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

release notes for these three versions, and many changes at a lower level are described in comments in the source code for those release notes (`(deflabel note-5-0 ...)` etc.).

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

A critical component in the continued evolution of ACL2 is feedback from the user community. We hope that you'll keep that feedback coming! Another contribution of the user community is the large body of Community Books [1], which put demands on the system and help us to test improvements. Please keep these coming, too!

References

[1] The ACL2 community: *ACL2 Community Books*. See URL `https://code.google.com/p/acl2-books/`.

[2] Shilpi Goel, Warren A. Hunt, Jr. & Matt Kaufmann (2013): *Abstract Stobjs and Their Application to ISA Modeling*. In: *Proceedings 11th International Workshop on the ACL2 Theorem Prover and its Applications*.

[3] Matt Kaufmann & J Strother Moore: *ACL2 documentation topic: RELEASE-NOTES*. See URL `http://www.cs.utexas.edu/users/moore/acl2/current/RELEASE-NOTES.html`.

[4] Matt Kaufmann & J Strother Moore: *Instructions for modifying ACL2 system code*. See URL `http://www.cs.utexas.edu/users/moore/acl2/open-architecture/how-to-make-patches.txt`.

[5] Matt Kaufmann & J Strother Moore (2011): *How Can I Do That with ACL2? Recent Enhancements to ACL2*. In David Hardin & Julien Schmaltz, editors: *ACL2*, *EPTCS* 70, pp. 46–60. Available at `http://dx.doi.org/10.4204/EPTCS.70.4`.

[6] Guy L. Steele, Jr. (1990): *Common LISP: the language (2nd ed.)*. Digital Press, Newton, MA, USA.