Enhancements to ACL2 in Versions 6.2, 6.3, and 6.4

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 improvements to ACL2 made since the 2013 ACL2 Workshop.

1 Introduction

We discuss ACL2 enhancements introduced in versions released between the ACL2 Workshops in May, 2013 and July, 2014: Versions 6.2 (June, 2013), 6.3 (October, 2013), and 6.4 (January, 2014). (These enhancements were thus made after the release of ACL2 Version 6.1 in February, 2013.) Hence this paper is analogous to two papers that correspond to earlier sets of releases [6, 5]. We summarize some of the more interesting of the roughly 100 items in the release notes for these three releases. While those release notes themselves are summaries, they are intended solely as documentation for what has changed: individual paragraphs are not generally intended to make sense except to those who have relevant background knowledge. Moreover, for the sake of completeness those paragraphs are often rather verbose and dense. This paper, on the other hand, is intended to provide a quick way to get up-to-date on the more interesting of the recent ACL2 system improvements. For anyone familiar with ACL2, each brief explanation below is intended to provide sufficient information so that even inexperienced ACL2 users can get a quick sense of the enhancement. Then they can decide whether to read more in the release notes, or even to follow the provided hyperlinks to relevant documentation.

ACL2 is typically revised to support soundness, robustness, and functionality of the system, often in direct response to user requests. Because of the maturity of ACL2, the system has many features, and thus improvements often pertain to aspects of ACL2 that may be unfamiliar to many ACL2 users, especially novice users. Our intention, however, is that this paper will have value to the entire ACL2 community, including newcomers. This paper thus includes a few introductory words about most every feature discussed. Moreover, the online version of this paper contains many links to documentation topics from the online ACL2 User's Manual [3], where one may learn more about those features. Notice that we use underlining, as for "release notes" and "documentation" above, to mark those hyperlinks. (The topic names typically match the printed names, but not always; for example, "release notes" above corresponds to the topic RELEASE-NOTES, which includes a hyphen.)

The focus of this paper is on the user level: our aim is to help ACL2 users to understand what is new with ACL2 (at least to them), in order to increase its effective use in their applications. On the other hand, those who wish to learn about implementation-level changes can look for Lisp comments in topics note-6-2, note-6-3, and note-6-4 in Community Book `books/system/doc/acl2-doc.lisp`.

One concept that arises several times in this paper is that of a *stobj*, or *Single-Threaded OBJect*. Let us review that concept briefly. Logically, a stobj is just a list whose members are called its *fields*. However, syntactic restrictions in their use permit stobjs to be modified destructively, and even to have fields that are represented as (destructively modifiable) arrays in the underlying host Lisp.

We organize the paper along the lines of the release notes, as follows.

Submitted to:
ACL2 2014

- Changes to Existing Features
- New Features
- Heuristic Improvements
- Bug Fixes
- Changes at the System Level
- Emacs Support

We do not discuss one other category in the release notes, Experimental/Alternate versions, though there have been improvements to ACL2(h), ACL2(p), and ACL2(r).

A label precedes each item being discussed, to indicate the relevant version of ACL2 in case you wish to read a longer description for that item in the release notes. For example, if the label is "6.2" then you can read more about the change by visiting the documentation topic, note-6-2.

2 Changes to Existing Features

As the sets of ACL2 users and projects continue to expand, we learn of ways to improve its existing capabilities. Some of these improvements are rather technical and hence may be new to the reader, who can follow links below to learn about them. Our message is twofold:

- ACL2 offers many capabilities beyond "only" a proof engine; and
- if you have avoided features that seemed awkward to use, consider trying them again, because they might have improved.

We give a few examples here, referring the reader to the release notes for a more complete list of such improvements.

6.2 The `trace$` utility, which shows calls and return values for indicated functions, can be configured to give less noisy output.

6.2 The guard-debug utility, which shows the origin of proof obligations generated for guard verification, avoids duplicating that information.

6.2 `Ld`, which invokes a read-eval-print-loop, has a new keyword argument, `:ld-missing-input-ok`, which avoids treating a missing file as an error.

6.2 `Ec-call`, a wrapper for executing function calls in the logic, allows non-stobj arguments in stobj positions.

6.2 Technical improvements have been made to the `meta-extract` capabilities for using facts from the context or world when proving meta-level rules (i.e., of class `meta` or `clause-processor`).

6.3 The `dmr` utility, which supports dynamically watching the rewrite stack, has undergone a few improvements. For example, when the debugger is enabled by evaluating `(dmr-start)`, then subsequent evaluation of `(dmr-stop)` will (once again) disable the debugger.

6.3 `Bind-free`, a construct that generates a binding alist for free variables in rule hypotheses, can return a list of binding alists to try.

6.3 Evaluation of an event (`progn` $\texttt{event}_1$ ... $\texttt{event}_k$) prints each $\texttt{event}_i$ immediately before evaluating it, just as was already done by evaluating a call of `encapsulate`

6.3 The set-inhibit-warnings utility, which suppresses specified types of warnings, is more predictable. Moreover, it now has a non-local variant, set-inhibit-warnings!.

6.3 Failure messages printed for defun indicate which proof attempt fails: termination or guard verification for the indicated definition.

6.3 The functionality of make-event, a macro-like capability that can involve the ACL2 state object, has been significantly expanded (see Section 3).

6.4 The :pbt ("print back through") utility, which queries the session history, now abbreviates bodies of large defconst forms.

6.4 The utility set-inhibit-output-lst, which supports which type of output to suppress, has had the output type "expansion" replaced by "history".

6.4 The optional :loop-stopper field of a rewrite rule can specify certain functions to ignore when comparing terms in order to avoid looping. Now, each such "function symbol" can actually be a macro alias.

6.4 ACL2 has a :logic mode utility, oracle-apply, for making higher-order function calls. This utility now has more appropriate restrictions codified in its guard.

6.x Error handling is improved for several utilities, including:

- 6.2 errors from the run-time type-checking utility, THE, have been eliminated when guard-checking is :none;
- 6.2 a much more instructive error message is printed for the defpkg "reincarnation" error that is encountered upon attempting to redefine a previously-defined package;
- 6.2 permission problems no longer cause errors for the file utilities open-input-channel and open-output-channel; and
- 6.4 errors are more instructive when permission problems are encountered for include-book.

3 New Features

This section focuses on a few of the more interesting new features recently added to ACL2. As before, the release notes for the indicated versions contain more complete information.

6.2 The defstobj event, which introduces singled-threaded objects (see Section 1), permits stobjs to have fields that are themselves stobjs or arrays of stobjs. In the case of these *nested stobj* structures, fields are accessed using a new construct, stobj-let.

6.2 ACL2 supports metatheoretic reasoning, which may be implemented using extended metafunctions that can access the environment. The logical world — that is, the ACL2 database — is now available to such functions using the function, mfc-world.

6.2 ACL2 supports many kinds of declare forms, including not only some that are processed by the host Common Lisp compiler, but also others processed by ACL2 that are specified using xargs forms. A new xargs keyword, :split-types, can be used to specify that the function's type declarations should be provable from its guard, not add to its guard.

6.2 See quick-and-dirty-subsumption-replacement-step for a way to turn off a potentially expensive prover heuristic.

6.3 The macro-like utility, make-event, evaluates a form to obtain a new form — its *expansion* — and then typically submits that expansion. This utility has been made more flexible by the addition of the following new capabilities, which we discuss below for the benefit of those who already have some familiarity with make-event.

- Expansions may have the form `(:or event1 ... eventk)`. In this case, each $\texttt{event}_i$ is evaluated in turn until one succeeds. That $\texttt{event}_i$ is then treated as the actual expansion, and is not re-evaluated after expansion (as would be necessary without the use of `:or`).
- Expansions may have the form `(:do-proofs event)`, for insisting on performing proofs during evaluation of `event` even in contexts where proofs are normally skipped (such as when rebuild is invoked).
- A keyword argument, `:expansion?`, provides an optimization that can eliminate storing expansions in certificate files.

6.3 See get-internal-time for how to change ACL2's timing reports, so that instead of being based on run time (cpu time) they are based on real time (wall-clock time).

6.3 A new utility, sys-call+, can invoke a shell command just as is done by sys-call. However, while `sys-call` prints the command's output as a side effect but returns `nil`, `sys-call+` actually returns that output as a string.

6.3 A new utility, verify-guards+, is like the verify-guards utility for verifying that calls of the indicated function lead only to function calls that satisfy their guards (preconditions). However, in `verify-guards+`, that argument can be the macro alias for a function. See verify-guards for an example showing why it would be unsound to permit `verify-guards` to take a macro alias as its argument.

6.4 The bookdata utility writes out event data on a per-book basis, for tools such as the one written by Dave Greve that is found in directory `tools/book-conflicts/` of the Community Books.

6.4 The utilities add-include-book-dir and delete-include-book-dir, whose effects are local to a book, specify directories denoted by `:dir` arguments of include-book. These utilities now have non-local analogues, add-include-book-dir! and delete-include-book-dir!.

6.4 Congruence rules specify equivalence relations to maintain during rewriting. [1] Patterned congruence rules [4] extend the functionality of congruence rules by allowing a more general specification of where an equivalence is to be maintained.

4 Heuristic Improvements

ACL2 development began in 1989, and development for the Boyer-Moore series of provers began in 1971. It is therefore not surprising, at least to us, that there are relatively few changes to prover heuristics in recent ACL2 releases. However, there are a few, because the user community finds new applications of ACL2 that present opportunities for improving the heuristics. The following summary is intended to give a sense of how ACL2's heuristics have been tweaked, without diving into unnecessary details of how they work. (Heuristics are generally not documented at the user level, since we do not expect it to be necessary or useful to understand in any depth how they work in order to be an effective ACL2 user.)

6.2 ACL2 has an *ancestors check* heuristic that can prevent excessive backchaining through hypotheses of rules. The following list (which we invite beginners to skip!) describes ways in which this heuristic has been improved:

- the heuristic no longer allows failure for `force`d hypotheses;
- it is delayed until a quick (type-set) check has a chance to verify or refute the hypothesis; and
- a slight weakening of the heuristic now permits backchaining based on counting variable occurrences.

6.2 The context (type-alist) built from a goal now considers assumptions in a different order that may strengthen that context.

6.3 `:By` hints are intended to specify when a goal is subsumed by a known fact, such as the formula of a `defthm` event. The subsumption check for `:by` hints has been made less restrictive.

6.3 The hint `:do-not preprocess` is intended to cause the ACL2 prover to skip the *preprocess* step of its proof *waterfall*. This hint now also eliminates the preprocess step during the application of `:use` and `:by` hints.

5 Bug Fixes

Among the bugs fixed were soundness bugs, which are all mentioned below. These tend to be obscure and not generally encountered by users, but we consider them particularly important to fix. Among the other bugs fixed, we mention here only a few of the more interesting ones.

Thus we begin with a description of the soundness bugs that were fixed. In contrast to the rest of this paper, we do not attempt to explain much about these bugs, not even how they could lead to proofs on `nil`. Pointers to some of those proofs can be found in the release notes.

6.2 System functions `acl2-magic-mfc` and `acl2-magic-canonical-pathname`, which were not designed for direct invocation by the user, could be exploited to prove `nil`.

6.2 Stobjs could be confused with strings in raw Lisp.

6.3 A stobj could be bound by a `let` or `mv-let` form without being among the outputs of that form, which allowed for serious violations of single-threadednesss.

6.3 (Gnu Common Lisp only) There was a bug in Common Lisp code in the implementation of the utility, `set-debugger-enable`.

6.4 ACL2 supports rules of class `:definition`, which are typically equalities that are much like normal definitions, but can actually have hypotheses. A soundness bug resulted from incorrect application of such rules during the handling of `:expand` hints.

6.4 A stobj recognizer predicate could be violated after updating the stobj.

6.4 The checks made when admitting congruence rules failed to ensure that a certain variable on the right-hand side of the conclusion, which was intended to be a fresh variable, was indeed actually fresh (i.e., did not occur elsewhere in the rule).

Here are a few of the more interesting bugs that were fixed, other than soundness bugs.

6.2 ACL2 supports a notion of *abstract* stobj, which is an abstraction of a corresponding ordinary (*concrete*) stobj. The `defabbstobj` event, which introduces a new abstract stobj, incurs certain proof obligations to ensure proper correspondence between the new abstract stobj and its specified concrete stobj. These proof obligations, in the form of `defthm` events, are printed when submitting a `defabbstobj` event, except for events that have themselves already been admitted. However, the events printed were not always sufficient in order to admit the `defabbstobj` event.

6.3 ACL2's proof output indicates splitters: rule applications that generate more than one subgoal. However, splitters were sometimes alleged when only a single subgoal was generated.

6.3 The utility `wof`, for directing output to a file (as the name stands for "With Output to File"), could cause an error when no error was appropriate. This problem also occurred with the `:psof` ("Print Saved Output to File") utility, since `psof` is defined in terms of `wof`. (`:Psof` is similar to the utility `:pso` ("Print Saved Output"), as both print proof output that had been suppressed by gag-mode; but `:psof` prints saved proof output to a file, while `:pso` prints to the terminal, which can take much longer.)

6 Changes at the System Level

Some of the more sweeping changes to ACL2 have taken place far from its theorem prover. Here we touch briefly on just a few of those.

6.2 From the beginning of ACL2, Gnu Common Lisp (GCL) has been among the host Lisps on which ACL2 can be built. Now, the ANSI version of GCL is also a host Lisp on which ACL2 (and ACL2(h)) can be built.

6.2 The previous system for certifying the Community Books has been updated; in particular, it is largely based on `cert.pl` and other utilities maintained by Jared Davis and Sol Swords. See books-certification.

6.3 ACL2 is now available between releases, via SVN, at `http://acl2-devel.googlecode.com`. Disclaimer (as per the warning message printed at startup): *The authors of ACL2 consider svn distributions to be experimental; they may be incomplete, fragile, and unable to pass our own regression.* That said, we have seen few problems with SVN distributions.

6.4 The ACL2 documentation is now maintained in the XDOC format developed and implemented by Jared Davis. Indeed, it is now recommended to peruse the acl2+books combined manual. That manual includes not only the ACL2 User's Manual but also topics from the Community Books, such as the XDOC topic itself as well as `cert.pl` (mentioned above).

7 Emacs Support

The primary Emacs-related ACL2 enhancement is a new utility introduced in Version 6.4, ACL2-Doc, for browsing documentation inside Emacs. This utility takes the place of Emacs Info, which is no longer supported because of the transition to XDOC discussed in Section 6. Emacs users will find this utility to be a nice alternative to using `:doc` at the terminal. It can be used to browse either the acl2+books combined manual or the ACL2 User's Manual. This utility is loaded automatically into Emacs when loading the standard file `emacs/emacs-acl2.el`.

8 Conclusion

We have outlined some of the more interesting and important changes to ACL2 in Versions 6.2, 6.3, and 6.4. We hope that a quick read of this paper will enable ACL2 users to focus quickly on topics of particular interest, while easily following links (in the online version of this paper) in order to learn

more about those topics, with the result of becoming more effective ACL2 users. Many more changes (about 100 altogether) are described in the release notes for these versions, and many changes at a lower level are described in comments in the source code for those release notes, such as `(defxdoc note-6-2 ...)`, in Community Book `books/system/doc/acl2-doc.lisp`.

A critical component in the continued evolution of ACL2 is feedback from its user community, which we hope will continue! We also appreciate contributions by the user community to the large body of Community Books [2]. These books put demands on the system and help us to test improvements.

Acknowledgements

We thank members of the ACL2 community whose feedback has led us to continue making improvements to ACL2. In particular, we thank the following, who are the people mentioned in one or more specific items in the release notes for Version 6.2, 6.3, or 6.4: Harsh Raju Chamarthi, Jared Davis, Jen Davis, Caleb Eggensperger, Shilpi Goel, Dave Greve, Warren Hunt, Robert Krug, Camm Maguire, David Rager, Gisela Rossi, Sol Swords, Raymond Toy, and Nathan Wetzler,

We expressly thank Warren Hunt for his continuing support of ACL2 use and development for many years at UT Austin.

We are grateful to Shilpi Goel, Warren Hunt, Robert Krug, Sandip Ray, and Nathan Wetzler for feedback on a draft of this paper.

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 No. CCF-1153558.

9 Bibliography

References

[1] Bishop Brock, Matt Kaufmann & J Strother Moore (2008): *Rewriting with Equivalence Relations in ACL2*. *Journal of Automated Reasoning* 40(4), pp. 293–306, doi:10.1007/s10817-007-9095-9. Available at `http://dx.doi.org/10.1007/s10817-007-9095-9`.

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

[3] Matt Kaufmann & J Strother Moore: *ACL2 User's Manual*. See URL `http://www.cs.utexas.edu/users/moore/acl2/current/manual/index.html`.

[4] Matt Kaufmann & J Strother Moore: *An Extension of Equivalence-based Rewriting*. To appear, Proceedings of ITP 2014.

[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] Matt Kaufmann & J Strother Moore (2013): *Enhancements to ACL2 in Versions 5.0, 6.0, and 6.1*. In Ruben Gamboa & Jared Davis, editors: Proceedings International Workshop on the *ACL2 Theorem Prover and its Applications*, Laramie, Wyoming, USA , May 30-31, 2013, *Electronic Proceedings in Theoretical Computer Science* 114, Open Publishing Association, pp. 5–12, doi:10.4204/EPTCS.114.1.