Note-8-4

ACL2 Sources

ACL2 Version 8.4 (August, 2021) Notes

NOTE! New users can ignore these release notes, because the documentation has been updated to reflect all changes that are recorded here.

Below we roughly organize the changes to ACL2 since Version 8.3 into the following categories of changes: existing features, new features, heuristic and efficiency improvements, bug fixes, changes at the system level, Emacs support, and experimental versions. Each change is described in just one category, though of course many changes could be placed in more than one category.

Note that only ACL2 system changes are listed below. See also note-8-4-books for a summary of changes made to the ACL2 Community Books since ACL2 8.3, including the build system. Also note that with each release, it is typical that the value of constant `*ACL2-exports*` has been extended, and that some built-in functions that were formerly in `:program` mode are now guard-verified `:logic` mode functions.

Changes to Existing Features

`Apply$`, `lambda$`, and `loop$` may be used with badged `:program` mode functions in top-level evaluation. To assign a badge to a `program` mode function, use the new feature `defbadge`.

For calls of the form `(HIDE (COMMENT "..." ...))`, the string is a bit more descriptive. See comment and see hide. Thanks to Mark Greenstreet for helpful discussions leading to this change.

The next one is just a tweak, included merely as an example of many such tweaks that add up to making ACL2 comfortable to use.

The events `defun-nx` and `defund-nx` now disable the executable-counterpart rune for the new function symbol. Thus, after either of these introduces function symbol `f`, the ACL2 rewriter will no longer attempt to simplify a call of `f` on concrete arguments by using evaluation (unless of course that executable-counterpart is enabled first). Thanks to Mark Greenstreet for an email leading to this change. The implementation of this change also fixes a bug: when the default-defun-mode is program mode, `defund-nx` now disables the definition rune for the new function, but that was not previously the case.

Some built-in functions supporting the macro, `position`, now `fix` (coerce) their accumulator argument to a natural number. Thanks to Mihir Mehta for supplying the initial such changes, for the purpose of avoiding numeric type hypotheses. After that, thanks to email correspondence from Warren Hunt, we added built-in `:type-prescription` rules for those same functions so that ACL2 type-set reasoning infers that `position` always returns either `nil` or a natural number.

The macro `defconst` no longer accepts an optional documentation string (which was already being ignored). Thanks to Eric Smith and Alessandro Coglio for suggesting this change, which avoids potential confusion; consider for example `(defconst *c* () "abc")`.

Two improvements have been made in support of the `case-match` macro. (1) The built-in constant `*ACL2-exports*` now includes the `"ACL2"` package symbol, `quotep~`. (2) The built-in function `symbol-name-equal` is now a guard-verified `:logic` mode function. Thanks to Stephen Westfold for email leading to these changes: for (1), pointing out that the special role of `quotep~` for the `case-match` macro applies only to that `"ACL2"` package symbol, not to other symbols with the same name; and for (2), pointing out that the expansion of a `case-match` call that invokes `quotep~` for matching was introducing `:program` mode code.

A call of `comment` is more often inserted when the prover inserts a call of `hide`. See comment for a discussion of such ways in which `comment` is used.

When a rewrite rule's conclusion is of the form `(equiv term1 term2)` where `equiv` is a known equivalence relation, ACL2 generally creates the rule to rewrite an instance of `term1` to the corresponding instance of `term2`, in a context where it is sufficient to preserve `equiv`. However, if that rule is illegal, for example because it would rewrite a variable, then the rule is effectively treated as `(equal (equiv term1 term2) t)`. This behavior is not new, but now an explanatory observation is printed when this happens, as for `foo` in the following example. Thanks to Mihir Mehta for suggesting such an enhancement.

```
(defun my-equiv (x y) (equal x y))
(defequiv my-equiv)
(in-theory (disable my-equiv)) ; optional (avoids warnings for the next form)
(defthm foo (my-equiv x (car (cons x x))))
```

A trust tag (see defttag) is now required to set the ld-prompt to a non-Boolean value, other than the brr prompt, since that can cause printing of the prompt to modify state in rather arbitrary ways.

Improved translation of function calls, especially those that involve congruent stobjs, including better error messages, much improved code comments, and simplified code. Thanks to Sol Swords for sending an example with a misleading error message.

The error message has been improved when a `:linear` rule is illegal due to simplification of its conclusion to a constant by ground evaluation. Thanks to Eric Smith for suggesting such an improvement.

It is no longer an error to repeat a `defun-sk` event after removing or adding the xargs declaration, `:guard t`; the latter is now essentially redundant. It is similarly no longer an error to repeat a `defun-sk` event by adding, removing, or changing keyword `:guard-hints` in an `xargs` declaration. Thanks to Alessandro Coglio for reporting these issues.

The set of `apply$` primitives has been expanded. These are built-in function symbols that do not need a warrant when reasoning about the application of `apply$` to them or using them in `loop$`. Thanks to Alessandro Coglio for noting that some built-in `:logic`-mode functions do not have warrants, in particular, `sublis-var`.

Improved `defwarrant` to be a no-op for `apply$` primitives.

ACL2 now points out when specious simplification takes place; see specious-simplification. Formerly this was the case only with gag-mode turned off; still, `prove` output needs to be on for any such message to be printed (see set-inhibit-output-lst). Thanks to Mihir Mehta for a query that led to this enhancement.

For `fmt` directives `~f` and `~F`, ACL2 now uses the alist component of the `evisc-tuple` argument and the global `evisc-table`. Previously it used these only for directives `~x`, `~y`, `~X`, and `~Y` (and deprecated directives `~p` etc.; see fmt). Thanks to Eric Smith for a query leading to this enhancement.

Reporting has been improved when encountering possible invariance violations for abstract stobjs. Now, when that happens an “illegal-state” is entered, as indicated by the prompt, and instructions are printed for how to proceed at your own risk. See illegal-state.

A `table`’s guard may now reference the ACL2 state. Thus, it may now be a term involving (at most) the variables `WORLD`, `ENS`, and `STATE`.

It is now always redundant to `unmemoize` a function symbol that is not currently memoized. See redundant-events.

The “basic” ruler-extenders (see rulers) now include not only the symbols `return-last` and `mv-list` but also the symbol `if`. As before, the termination analysis always continues through the true and false branches of `IF` calls; but now, by default, it also continues through the first argument of an `IF` call. Thanks to Eric Smith for suggesting this improvement and testing it on some proprietary books.

When supplying `state` as an argument to `defun-nx` (or `defund-nx`), it is no longer necessary to declare `state` as a stobj or use `set-state-ok`. Thanks to Eric Smith for suggesting the possibility of this change.

It had been the case that if `defun-nx` or `defund-nx` is used for a recursive definition, and that form specifies a value for `:ruler-extenders` that omits `return-last` (see rulers for relevant background), then the definition generally fails to be admitted. (This is due to the generated `defun`’s use of `prog2$`, which is a macro that abbreviates a call of `return-last`, which blocks the termination analysis.) That has been fixed, by ensuring that `defun-nx` and `defund-nx` arrange that `return-last` is always among the ruler-extenders of the generated `defun` form. Thanks to Eric Smith for noticing this issue and for a helpful discussion.

Improved handling of linear rules: cause an error with a helpful message when a linear rule is no longer created during `include-book` or the second pass of an `encapsulate` form, and optimize by avoiding certain calculations when the `:trigger-terms` keyword is supplied. Thanks to Eric Smith for sending an example that illustrates the former issue, essentially as included in a comment in community-book `books/system/doc/acl2-doc.lisp`, form `(defxdoc note-8-4 ...)`.

The event macros `value-triple` and `assert-event` have been changed to be more flexible, in particular by providing an option to allow the given form to return multiple values. They are also more efficient, as they no longer evaluate using safe-mode by default. (Technical note: As a consequence of implementation changes, one will rarely if ever see “hard” errors (see er) from these utilities.) Thanks to Eric Smith for requesting that a single utility encompass what is provided by the built-in utility `assert-event` and the community-books utilities `assert!` and `assert!-stobj`. That single utility is now `assert-event`, which in turn uses `value-triple` to check the supplied assertion.

The ACL2 bdd package can now reason using the implicit rewrite rule `(equal (consp (cons x y)) t)`. Thanks to Warren Hunt for requesting this enhancement.

The event macro `with-output` has been improved with several changes and new features (and correspondingly, so has `with-output!`). In addition to improvements to its documentation, these include the following; see `with-output` for more information.

- The keyword argument `:summary` has been replaced by keyword arguments `:summary-on` and `:summary-off`, which control inhibited summary types in analogy to how keyword arguments `:on` and `:off` control inhibited output.
- The keyword arguments `:on` and `:off` may now take value `nil`.
- So-called “on-off specs”, `(:on-off sym1 ... symk)` are now allowed for keyword arguments `:on` and `:off`, which let you turn on or off all output types except the specified `symi`. On-off specs are similarly supported for keyword arguments `:summary-on` and `:summary-off`.
- There is a new keyword for `with-output`, `:ctx`, which supports a sort of global override of event contexts. See with-output and see ctx.
- There is better error checking, in particular for duplicate keywords.

The value returned by a successful event `(encapsulate ... ev_i ... ev_n)` is no longer always `T`. See encapsulate.

Two event macros produce much less output than before, as follows. Thanks to Alessandro Coglio for requesting these changes and to him and Eric Smith for helpful discussions.

- `Defstub` is not only quieter but also does a bit more error checking.
- `Defun-sk` has a new `:verbose` argument for when output is desired after all.

For the `:logic` and `:exec` components of a `defabsstobj` export, signatures no longer need to match. Thanks to Sol Swords for suggesting this improvement. Now, the only such requirements are that the lengths of the input signatures are equal and the lengths of the output signatures are equal.

The utility `set-guard-checking` now prints messages to `(standard-co state)` rather than to `*standard-co*`. (Of course, these are the same by default.)

In a proposed linear rule, hypotheses that are calls of `bind-free` are now accounted for when considering whether each of the `:trigger-terms` has sufficient free variables (see linear, in particular condition (b) there, and see bind-free). Thanks to Dave Greve for reporting this issue. The example below has the indicated failure in ACL2 Version 8.3 but is now accepted.

```
(defstub f (x) t)
(defstub g (x) t)

; The following two events are the same except that the first supplies
; :trigger-terms while the second does not (instead computing trigger terms
; heuristically).  These should either both fail or both succeed; now they
; both succeed.

; FAILED, but now SUCCEEDS after the fix since then variables y and z are
; considered to occur (free) in the hypotheses when determining
; whether the hypotheses and trigger term include enough variables to cover
; those in the conclusion.
(defaxiom ax1
  (implies (bind-free '((z . z))) ; or, (bind-free '((z . z)) (y z))
           (≤ (f x) (g y)))
  :rule-classes ((:linear :trigger-terms ((f x)))))

; SUCCEEDED even before the fix: the bind-free hypothesis fools ACL2
; into believing that otherwise-free variables may be bound by the
; bind-free alist.
(defaxiom ax2
  (implies (bind-free '((z . z))) ; or, (bind-free '((z . z)) (y z))
           (≤ (f x) (g y)))
  :rule-classes :linear)
```

The report for `:brr` when a hypothesis fails to be relieved shows repeated attempts to bind free variables not only as before — that is, when a hypothesis has free variables — but also when the hypothesis is a call of `bind-free` that returns a list of substitutions. Thanks to Dave Greve for bringing up this issue.

It is now an error for a `:linear` rule's conclusion to generate a polynomial whose terms are all determined to be non-numeric. Thanks to Eric Smith for suggesting such a check. Here is an example of such a rule that was formerly admitted but useless but now causes an error.

```
(defthm bad-rule
        (equal (≤ (foo x) x) nil)
        :rule-classes :linear)
```

In executable code, calls of `hide` in `let`-bindings or `lambda` bodies are no longer considered to represent `ignore` declarations. We thank Alessandro Coglio for raising and discussing this issue. Below is a log, based on that discussion, that was produced using ACL2 before this change. After this change, the second and third calls of `:trans` will now result in errors, as noted in comments below. Previously ACL2 treated the `hide` calls in those two input expressions as though they represented `ignore` declarations, in the sense of translation of the first form shown below. That is still the case when translating for theorems rather than executable code; for details see the “Essay on Using Hide for Ignored Let-bindings” in the ACL2 source code.

```
ACL2 !>:trans (let ((x 0)) (declare (ignore x)) 1)

((LAMBDA (X) '1) (HIDE '0))

=> *

ACL2 !>:trans ((LAMBDA (X) '1) (HIDE '0)) ; error after this change

((LAMBDA (X) '1) (HIDE '0))

=> *

ACL2 !>(untranslate '((LAMBDA (X) '1) (HIDE '0)) nil (w state))
(LET ((X (HIDE 0))) 1)
ACL2 !>:trans (LET ((X (HIDE 0))) 1) ; error after this change

((LAMBDA (X) '1) (HIDE '0))

=> *

ACL2 !>
```

The `defabsstobj` keyword, `:CONCRETE`, has been changed to `:FOUNDATION`. Although the use of `:CONCRETE` is still supported in this release (Version 8.4), it generates a warning that this usage is deprecated and will likely not be supported after this release. Also, documentation and comments now typically speak of “foundational stobj” for the underlying stobj of the abstract stobj (which can be supplied explicitly by the `:FOUNDATION` keyword) rather than “corresponding concrete stobj”; this reflects the fact that the foundational stobj may itself be an abstract stobj (which is not new for this release).

Strengthened error-checking for `stobj-let` to insist that if an updater is supplied explicitly in a binding, then it must be a valid updater. This check was formerly made only if the variable bound in that binding is among the producer variables (see nested-stobjs). For example, the following now causes an error, but it was formerly accepted in spite of the fact that `xyz` is not the updater for the accessor, `top1-fld`; in fact `xyz` is not even defined!

```
(defstobj sub1 sub1-fld1)
(defstobj top1 (top1-fld :type sub1))
(defun f1 (top1)
  (declare (xargs :stobjs top1))
  (stobj-let
   ((sub1 (top1-fld top1) xyz)) ; bad updater!
   (val)
   (sub1-fld1 sub1)
   val))
```

The function symbol `symbol<` replaces the function symbol `symbol-<`. More generally, for every built-in function symbol and theorem name containing `"SYMBOL-<"`, that string in its `symbol-name` is replaced by `"SYMBOL<"`. The function `logical-defun` is similarly replaced by `get-defun-event`. Thanks to Alessandro Coglio for suggesting these changes. Note that the old function names still work in ACL2 Version 8.4, as they are macro-aliases for the corresponding new function names (see add-macro-alias); however, they are deprecated and will probably not be supported in later ACL2 versions. (A deprecation warning is printed each time one of those macros is expanded.)

The keyword `:witness-dcls` of `defun-sk` is deprecated and will probably be unsupported in future ACL2 releases. Use `declare` forms instead; see `defun-sk`.

New Features

It is now possible to assign `badge`s to `:program` mode functions, which allows them to be used by `apply$`, `lambda$` and `loop$` during top-level evaluation. See `defbadge`, which assigns badges but not warrants and which can handle both `:program` and `:logic` mode functions.

A new option for `certify-book`, provided with keyword option `:useless-runes` or environment variable `ACL2_USELESS_RUNES`, makes it possible to speed up repeated certification of a book, sometimes substantially. See useless-runes. Thanks to Sol Swords for reporting a bug (in ACL2 source function `read-file-iterate-safe`) and supplying a fix, which we have incorporated. Also thanks to Eric Smith for requesting that each tuple in a generated “useless-runes” file be printed on a single line starting after a space, thus supporting grep-like tools. Finally, thanks to Alessandro Coglio and Eric Smith for discussions leading to creation of `.sys/` subdirectories to hold useless-runes files.

A new keyword for `defstobj`, `:non-executable`, can be given value `t` to skip memory allocation for the new stobj, by avoiding creation of a “live” (mutable) stobj. See defstobj. Thanks to Warren Hunt for encouraging development of this feature.

The rewriter now rewrites the bodies of certain quoted lambda objects. However, some restrictions apply. See rewrite-lambda-object. In addition some new lemmas and a new metafunction have been added to the book `projects/apply/top`, which users are still encouraged to include in sessions dealing with `apply$`. The new metafunction is named `relink-fancy-scion` and can also cause `lambda` objects to be transformed. See the discussion of that metafunction in rewrite-lambda-object.

Added `toggle-inhibit-warning` and `toggle-inhibit-warning!` to add or delete a warning string from the `inhibit-warnings-table`, rather than setting that entire table as is done by `set-inhibit-warnings`.

It is now possible to instruct the prover to respect requests to disable the definition of the primitive, `mv-nth`, rather than continuing to expand that definition. See theories-and-primitives, in particular the note for advanced users at the end of that topic. Thanks to Alessandro Coglio for requesting this feature.

A new utility, `memoize-partial`, allows memoization for functions that were admitted by adding a formal parameter that decreases on each recursive call (sometimes called a “limit” or a “clock”). Normally that extra parameter can severely impede the utility of memoization; however, the function actually executed does not have that extra parameter. This allows for more memoization hits. Thanks to Mertcan Temel for an inquiry leading to this enhancement, and for helpful discussions.

A new rule-class (see rule-classes) has been added, named `:rewrite-quoted-constant`. Rules in this class can cause the rewriter to replace one quoted constant by an equivalent one under a given equivalence relation. See rewrite-quoted-constant.

The `loop$` parser produces more informative error messages on ill-formed `loop$` statements.

A new `memoize` keyword, `:invoke`, supports the replacement of calls of one function by another. In that sense it is similar to `defattach`; the difference is that with `(memoize f`

`:invoke g`), it is necessary first to prove the equality of `f` and `g`; therefore, ACL2 will compute calls of `f` by calling `g` even during proofs. In particular, the tool add-io-pairs is built on top of this capability; it allows evaluating a function call by fast lookup of a verified input-output pair. Thanks to Eric McCarthy, Alessandro Coglio, and Eric Smith for requesting the latter capability and providing helpful feedback.

A new xargs keyword for `defun`, `:type-prescription`, can be supplied as a formula in the shape of a type-prescription rule. It is checked to be implied by the built-in type-prescription rule computed for the newly-defined function. Thus, it can serve as documentation for the expected type returned by the function; if the implication is not equivalence, a warning is printed. Thanks to Alessandro Coglio and Eric Smith for the suggestion.

Tables may now supply custom error messages for table `:guard` failures. This is accomplished by supplying a table `:guard` that returns two values instead of one. For a return `(mv okp msg)`, if `okp` is non-`nil` then the table guard is considered to be true, that is, it has the same meaning as a non-`nil` single-value return. Also, `(mv nil nil)` has the same meaning and effect as a `nil` single-value return: the table guard fails, and a generic error message is printed about the illegal key/value pair. The new case is a return of `(mv nil msg)`, where `msg` should be a `msgp` — a string or a cons suitable for printing with the `fmt` directive, `~@`. In that case, `msg` is printed (using the `fmt` directive, `~@`) instead of a generic error message. This new feature is now used in some built-in tables. See table.

The proof-builder command `type-alist` has a new optional argument that supports printing the type-alist in an alist format. Thanks to Mihir Mehta for requesting this feature.

A new summary type, `REDUNDANT`, controls whether a message is printed indicating a redundant event (see redundant-events), which however will still always take place if `EVENT` output is not inhibited (see set-inhibit-output-lst). See summary.

A `defabsstobj` event may now specify child stobj fields, for use by `stobj-let`. See defabsstobj, and see community-books file `books/system/tests/abstract-stobj-nesting/README` for a brief guide to the examples in that directory. Thanks to Sol Swords for requesting this feature and helping to design it, as well as for his helpful discussions, feedback, and test files.

The `:congruent-to` keyword is now supported for `defabsstobj`.

`Stobj-let` now allows aliases in the bindings in some cases where this is safe because updating is not involved. Thanks to Sol Swords for suggesting such a change.

Added a function `ctxp` to recognize valid contexts, which are used for printing error message (see ctx). Thanks to Eric Smith for requesting this addition and to Alessandro Coglio for suggesting that it be disabled, for efficiency.

The utilities `brr` and `monitor` now each take an optional argument that avoids output. A new utility, `monitor!`, is a combination of these two with their new optional arguments of `t`, thus also avoiding output. Thanks to Eric Smith for requesting a version of `monitor` that turns on `brr`.

A new utility, `get-guard-checking`, returns the guard-checking value most recently installed, either `t` (when the ACL2 executable was built) or presumably by `set-guard-checking`. Thanks to Eric McCarthy for suggesting this utility.

The function `aset1-trusted` may be used in place of `aset1` to avoid invariant-risk, but is therefore untouchable.

Added a new utility, `print-object$+`, that is like `print-object$` but instead of taking `STATE`, `print-object$+` is a macro that takes keyword arguments to customize the output. See print-object$+. Thanks to Eric Smith for requesting such additional print control.

Heuristic and Efficiency Improvements

These are mostly pretty interesting, but a bit much to explain in this short talk. You're encouraged to take a look!

We changed the lightweight "preprocess" simplifier for "simple" rules in the prover's waterfall, in the case that the term is the application of a defined function symbol to constant (quoted) arguments. As before, if the executable-counterpart rule for that function symbol is enabled, then the call is evaluated in Lisp; and if that evaluation fails (typically because a constrained function is called), then an attempt is made to rewrite the term by applying a simple rule. The change is for how failure is handled, that is, in the case that evaluation fails and the term is not rewritten. Formerly, the term was surrounded by a call of `hide`. Now, that is only done by the main rewriter, not by the "preprocess" simplifier. Thanks to Eric Smith for a communication on the acl2-help list, on a thread started by Mark Greenstreet, that led us towards making this change. Mark's failed proof now succeeds after this change, but here is a simpler example.

Formerly, the commented-out `:do-not` hint was required for the proof of the `thm` call below to succeed, because the definition of g is not simple and hence the "preprocess" simplifier replaced `(g 3)` by a term `(hide (comment ...) (g 3))` before the rewriter could apply the definition of g.

```
(defstub f (x) t)
(defun g (x) (cons x (f x)))
(thm (equal (car (g 3)) 3)
     ;; :hints (("Goal" :do-not '(preprocess)))
     )
```

The ACL2 rewriter has a "being-openedp" heuristic that prevents loops, by saving a stack based on what is currently being rewritten. This can prevent the use of a definition or rewrite rule. Now the heuristic is turned off when the term's function symbol has a non-recursive definition and simplification has just settled down (see hints-and-the-waterfall). To restore the old behavior, i.e., to use the heuristic in all cases — thus providing backward compatibility when a proof fails — evaluate the form: `(defattach-system being-openedp-limited-for-nonrec constant-nil-function-arity-0)`.

When a command executed in a logical world, `w`, is interrupted, the world is reverted to `w`. That reversion process could be very slow if the interrupt was taken during the installation of the new world, as indicated by the messages:

```
Flushing current installed world.
Reversing the new world.
Installing the new world.
```

That process has been sped up significantly. Moreover, the new process avoids a bug reported by Eric Smith, who we thank for sending an example of how the process was interacting badly with `reset-prehistory`. Code implementing that interaction was introduced in Version 4.0 to speed up the process; that code has been eliminated, as it is no longer necessary.

ACL2's type-set reasoning has been slightly strengthened to comprehend Boolean combinations of strong compound-recognizer calls on a single variable when building a context (a so-called type-alist). (By a "strong compound-recognizer call" we mean a unary function that recognizes a union of primitive ACL2 types and has, either explicitly or implicity, a corresponding compound-recognizer rule; examples include `stringp`, `integerp`, and `true-listp`.) In particular, this change can strengthen the result of forward-chaining. Thanks to Eric Smith, who raised this issue by providing an example that we include in a comment, inside the form `(defxdoc note-8-4 ...)` in community-book `books/system/doc/acl2-doc.lisp`.

Type-set reasoning also has been improved for inequalities to take more advantage of the fact that the number 1 constitutes a singleton type. For example, after evaluating `(defstub foo (x) t)` ACL2 is now able to prove `(implies (<= 2 x) (equal (foo (bitp x)) (foo nil)))` where previously it could not. In fact `2` can now be replaced by any rational number that is at least 1.

The function `compress1` has been made more efficient in the following ways.

- `Compress1` is now faster when the `:ORDER` specified by the given array's header is < or > and the alist is properly ordered: header first, then ascending (for `:ORDER <`) or descending (for `:ORDER >`) order of indices, with no value in the alist equal to the `:DEFAULT` specified by the header. In particular, this can cut the time to run `compress1` on an alist containing only the header by more than half, which addresses a request made by Eric Smith (whom we thank for bringing this efficiency issue to our attention).
- For the change noted just above, Eric also noticed that when the default is `nil` then there was no speedup. This led us to fix an existing bug (technical description: for an array with default `nil`, the alist was never considered to be in order).
- Functions `compress1` and `compress2` no longer require the new and old dimensions to agree in order that the underlying raw lisp array is reused. Now it suffices for the new dimension (for `compress1`; each dimension for `compress2`) to be at least as great as the old. (You can still avoid reuse of the raw Lisp array by using `flush-compress`.) Thanks to Eric Smith for suggesting this change.

We found that `accumulated-persistence` was measurably slowing down ACL2 even when it is off. We modified its implementation and measured a time reduction of 3.7% for a proof-intensive book. The modification includes a documented efficiency tweak to `wormhole-eval`.

The linear-arithmetic heuristics have long taken advantage of negated equality hypotheses. These heuristics have been strengthened to take further such advantage. Thanks to Warren Hunt for sending an example proof attempt that failed before this change but now succeeds, and for his encouragement to pursue an improvement to linear arithmetic that can benefit such proof attempts.

The removal of guard-holders has been augmented to include removal of certain “trivial” lambda applications. See guard-holders, in particular for how to restore the legacy behavior. Thanks to Alessandro Coglio for an example and Eric Smith for a subsequent suggestion that led to this enhancement.

Bug Fixes

A soundness bug, present since `loop$` was introduced, was fixed. The bug was manifested when the keyword `:guard` was used as the `loop$` body, as in `(loop$ for v in lst collect :guard)`. (Note: It’s not clear that this bug could be used to prove `nil`.)

A soundness bug was fixed by changing `defabsstobj` to avoid using `mbe` in the definitions generated for the logic. The problem was that the `:logic` and `:exec` forms are not actually equal (they correspond, in the sense of the correspondence predicate), and this can be exploited by using a `:guard-theorem` lemma-instance. For an example proof of `nil` in Version 8.3, see a comment about a `defabsstobj` bug in the form `(defxdoc note-8-4 ...)` in file `books/system/doc/acl2-doc.lisp`.

A soundness bug was fixed in the functions that print to strings, such as `fmt-to-string`; see printing-to-strings. The bug was a dependence of the result on the ACL2 state, even though `state` is not an argument to these functions. More specifically, the dependence was on the `current-package` and the global `evisc-table`. For examples see community-books `books/system/tests/fmt-to-string.lisp` and `books/system/tests/fmt-to-string-pkg.lisp`. Note: We also strengthened the guards on these functions to require that the keys of the `fmt-control-alist` alist argument are all appropriate; in particular, the symbol `current-package` in the `"ACL2"` package is a suitable key, but for example the keyword `:current-package` is not.

It was possible to prove `nil` by counterfeiting `record-expansion` calls, which are normally only created by the implementation in support of `make-event` calls. This soundness bug has been fixed. (The bug was perhaps impossible to hit in ordinary usage, where `record-expansion` is not used explicitly.) A proof of `nil` before this fix may be found in a comment in ACL2 source function, `corresponding-encaps`.

The mechanism for tracking warrants needed during a proof had a bug, which might be a soundness bug if one uses `apply$` or `loop$`. That bug has been fixed.

Fixed a bug that was preventing use of the RDTSC hardware instruction in SBCL on most x86-based platforms, and possibly erroneously attempting to make use of that instruction on some other platforms. Thanks to Keshav Kini for a query that led to this fix. Also restricted RDTSC to x86-based platforms, thanks to a suggestion by Curtis Dunham, to enable Arm builds of ACL2.

The use of `apply$` on calls of `if` no longer cause raw Lisp errors. (The same is true for calls of the subroutine `apply$-prim` of `apply$`.) For example, these calls now evaluate without error: `(apply$ 'if '(nil nil nil))` and `(apply$-prim 'if '(nil nil nil))`.

We made the following fixes to the `accumulated-persistence` utility.

- Applications of type-prescription rules that were erroneously labeled as "useless" are now appropriatedly labeled as "useful".
- The documentation for `accumulated-persistence` now correctly states that the form `(accumulated-persistence-oops)` undoes the clearing effect of `(accumulated-persistence t)`, rather than of `(accumulated-persistence nil)`.
- Uses of definition rules because of `:expand` hints, either explicitly given by the user or generated by ACL2's heuristics for doing induction, were not being recorded by `accumulated-persistence`. They are now.

Fixed a bug that was causing books to be included as "uncertified" after their certification stored checksums (see book-hash). Thanks to Keshav Kini for reporting this bug.

ACL2 could occasionally simplify subterms of a call of `hide` even without any applicable rules or `:expand` hints. (Technical note: the problematic source function was `normalize`.) We considered this to be a bug, so it has been fixed.

The notion of redundancy for `defstobj` events was too weak, as evidenced by the following example. Consider the following book, named `"bug.lisp"`.

```
(in-package "ACL2")
(defstobj st fld)
(defun foo (st) (declare (xargs :stobjs st)) (fld st))
```

After certifying this book, a raw Lisp error could occur after evaluating the following forms, because the compiled definition of `foo` from including the book referenced a function, `fld`, which is now a macro.

```
(defstobj st fld :inline t)
(include-book "bug") ; The defstobj event in this book is redundant here.
(foo st)
```

The bug has been fixed by requiring a redundant `defstobj` event to be syntactically identical to the pre-existing corresponding `defstobj` event.

A raw Lisp error would occur when the value of an `:instructions` hint is not a true (null-terminated) list. ACL2 now produces an informative error message in that case.

Improved `:print-gv` to do its computation in the ACL2 world in which the guard violation took place. See print-gv; in particular, that topic concludes with a discussion of this issue. Thanks to Eric Smith for reporting this issue along with a simple example.

The following `defun-sk` bugs were fixed.

- The guard was being ignored (that is, treated as `t`) when `:constrained t` was specified in a `defun-sk` event. Thanks to Alessandro Coglio for reporting this bug.
- `Defun-sk` could fail when the formal parameters include stobjs or `state`.
- `Defun-sk` provided no direct way to allow formals to be ignored. Now, `ignorable` declarations are permitted.

For Lisps that do not compile on the fly (that is, Lisps other than CCL and SBCL), evaluation of `:comp t` effectively unmemoized functions that were not already compiled. This has been fixed.

When a stobj hash-table's "init" function was called with size 0, an error could occur for host Lisp GCL, complaining that size 0 is not allowed for hash-tables. This has been fixed by using size 1 instead of 0 in this case.

Reporting by break-rewrite has been fixed for cases when failure is due to a hypothesis with a backchain-limit of 0. There were actually two such bugs: one due to improper handling of linear rules (which might have occurred even for other backchain-limits besides 0), and one due to the erroneous assumption that backchaining has only just begun at the point of failure. Thanks to Mihir Mehta for sending an example that exhibited both bugs. A discussion of the second bug, including a simple example, may be found in a comment in the ACL2 source function, `tilde-@-failure-reason-phrase1-backchain-limit`.

`:OR` hints that contain a single list (which satisfies `keyword-value-listp`) were being mishandled. This has been fixed. Thanks to Dave Greve who sent an example `defthm` event specifying `:hints (("Goal" :or ((:in-theory (e/d () ()) :nonlinearp t))))`, which formerly caused a Lisp error.

Fixed a bug in tracking the cbd that could cause failures of `include-book` in raw-mode. Thanks to Warren Hunt for a query leading to this fix.

Fixed a bug that was preventing some deep patterned-congruence rules from being applied. This could occur when on the left-hand side, the outermost function symbol is the same as the next function symbol going in towards the variable that differs on the right-hand side. Thanks to Mihir Mehta for reporting this bug and including a replayable example.

Fixed several issues with `fmt` (and related printing utilities) pertaining to linebreaks. These include the following, many of which are illustrated in a new file, `system/tests/fmt-tests-input.lsp` (search there for “2020”).

- A space past the `fmt-soft-right-margin` now results in a linebreak even in the case of tilde-space (‘`~ `’).
- As before, for puctuation after a tilde-x (‘`~x`’) directive, ACL2 avoids printing in column 0 after a linebreak. This desirable behavior now extends to the case that ‘`~x`’ is inside a tilde-atsign (‘`~@`’) directive. For example, after `(set-fmt-hard-right-margin 10 state)` try either `(fmx "~@0." (msg "~x0" '(ab de gh)))` or (fmx "~#0~[~x0~/~x0~]." ’(ab de gh)).
- Spaces are respected after tilde-y (‘`~y`’) directives. For example, `(cw "~y0 A~%" 3 4)` now prints the letter `A` in column 2 rather than column 0. Thanks to Eric Smith and Alessandro Coglio for suggesting this change.
- An extra character is sometimes permitted before deciding that a tilde-x (‘`~x`’) directive causes a linebreak.
- Tilde directives that cause no printing are often ignored when deciding whether to keep a punctuation mark with a pretty-printed expression. For example, when setting both right margins (soft and hard) to 10, the following no longer prints a comma in column 0: `(fmx "~x0~@1, more~%" 'aaaaaaaaaa "")`.
- Fixed a `double-rewrite` warning, which was breaking the word “is”. Thanks to Mihir Mehta for pointing this out.

After setting state global `trace-co`, for example with `(f-put-global 'trace-co (@`

`standard-co) state)` (say, after setting standard-co to an open output channel), printing of the trace level such as "`1>`" and "`<1`" will now go to that channel. Formerly, this could fail after setting `trace-co` directly rather than using `open-trace-file`.

There are improvements to `certify-book` and `include-book`, which pertain to the loading of compiled files by `include-book` before events in the book are processed. That process supports preservation of honses and fast-alists, and reporting of "stolen" alists (see for example with-stolen-alist). Relevant tests, with implementation-level comments, may be found in the new community-books directory, `books/system/tests/early-load-of-compiled/`. Thanks to Sol Swords for helpful discussions.

Improved a utility that builds sets of clauses, which improves the reliability of using a lemma-instance of the form `(:guard-theorem <name> nil)`. Thanks to Eric Smith for reporting this problem with a simple example, and to Dave Greve for following up with a related example; both now work as one would expect.

Fixed printing of the ACL2 state in error messages, specifically when executing a non-executable function.

Fixed a bug that could cause a raw Lisp error when processing a linear rule with a `bind-free` hypothesis, when that hypothesis does not specify a list of variables (in its second argument). Thanks to Dave Greve for reporting this bug by sending a simple example. (Technical note: the fix was in the definition of source function all-vars-in-hyps.)

For `defabsstobj`, a suitable error now occurs when the `:LOGIC` version of an abstract stobj export has the foundational stobj as a formal parameter that is declared as a stobj. Formerly, a confusing hard error could occur in this case.

An unfortunate "Proof skipped" could be printed during the `include-book` phase of `certify-book` for certain uses of `make-event`, including calls of `thm`. This has been fixed.

Fixed an implementation error that was being reported when a `stobj-let` form updates two array fields each with at least two indices that are not all natural numbers. For an example that caused this error, see community-book `books/system/tests/nested-stobj-two-updates.lisp`.

Fixed a bug in resizing stobj arrays whose elements are specified to be `(signed-byte 30)` or a subtype of that type. Thanks to Eric Smith for reporting this bug with a reproducible example.

For a stobj-let call occurring in other than a definition body, the top-level bindings of variables to child stobj accessors was treated as let* rather than as let. That was at odds with the documentation, and has been fixed.

Fixed a bug that caused an error when attempting to redefine a function for which a :compound-recognizer rule has been proved. Thanks to Eric McCarthy for reporting this bug (GitHub Issue #1273) with a reproducible example.

Raw mode now does a better job of maintaining global stobj values. Here is an example that illustrates the fix; also see set-raw-mode, which has been extended to explain a related but remaining issue.

```
(defstobj st2 (ar :type (array t (8)) :resizable t))
(set-raw-mode-on!)
(resize-ar 20 st2)
(ar-length st2) ; formerly 8, but now 20 as expected
```

For any proof-builder command of the form (= term1 term2 atom ...), the keyword arguments were ignored. This has been fixed, and also the documentation for ACL2-pc::= has been improved.

Consider when a community-book is included using an ACL2 executable different from the one that certified the book, with those two executables being located in different directories. The book is now considered to be uncertified in that case. To avoid this issue see include-book for a discussion of environment variable ACL2_SYSTEM_BOOKS.

Changes at the System Level

(SBCL only) Filenames are now read as ASCII (specifically, ISO-8859-1) when the host Lisp is SBCL, which formerly was not the case. Thanks to Stephen Westfold for suggesting this change.

ACL2 can once again be built on CMU Common Lisp (CMUCL) (though we have only done minimal testing). The problem turned out to be with ACL2, not CMUCL: low-level Lisp code in the ACL2 sources was destructively modifying a quoted constant. (For implementation details, see *fncall-cache* in source file translate.lisp.) The bug was discovered when considering modification of the build process to compile ACL2 source files when the host Lisp is SBCL. Thanks to Stas Boukarev for pointing us in the right direction to debug this error.

There is now a way to save the untranslated bodies of built-in `:program` mode functions. WARNING: This is not officially supported! But it may be useful for tools such as a linter. See `GNUmakefile`: search there for `:acl2-save-unnormalized-bodies`, where you will see that a more general capability is actually available for evaluating forms before the build begins. Thanks to Eric Smith for correspondence leading to this enhancement.

Starting with ACL2 Version 7.0, the availability of hash consing (see hons) and the other features described in hons-and-memoization have been part of a default ACL2 build; meanwhile, support for “classic” ACL2 has essentially been discontinued. These features require a certain Lisp action (implementation note: pushing `:hons` onto `*features*`) that had been done during the ACL2 build by the ‘`make`’ process (with code in `GNUmakefile`). Now that action is taken unconditionally in the ACL2 source code. Thanks to Petter Gustad for reporting an error when attempting to build ACL2 without using ‘`make`’; this change should fix that bug. With this change you can no longer attempt to build “classic” ACL2(c) without editing ACL2 source file `init.lisp`; environment variable `ACL2_HONS` no longer has any effect. (On a related technical note: An obscure feature `:memoize-hack` seems not to be used anywhere, and has been eliminated.)

(SBCL only) Handling in ACL2 of the SBCL “read-cycle-counter” has been modified to reflect its handling in recent SBCL versions (starting around late 2018 or early 2019). This avoids an ACL2 build error on some platforms. Thanks to John R. Strohm for reporting such a problem (on a Raspberry Pi).

(SBCL only) Increased the number of special variables that can be created, which allowed community book `books/kestrel/apt/schemalg-template-proofs.lisp` to certify.

Fixed builds that use a relative pathname for the LISP environment variable (for host Lisps CCL, SBCL, Allegro CL, and CMUCL; GCL and LispWorks didn’t seem to have this problem). Thanks to Mihir Mehta for bringing this issue to our attention; and thanks to Andrew Walter for reporting a problem with our first solution for SBCL and proposing an alternative, which we adopted.

Fixed the process for running ACL2 without building an executable image. Some initialization that was missing from that process is now included. Also, added the missing command, (lp), to the instructions in section “Running Without Building an

Executable Image” on the “Obtaining and Installing ACL2” web page (accessible from the “Obtaining, Installing, and License” link on the ACL2 home page). Thanks to Petter Gustad for an inquiry leading to these improvements.

When invoking ‘`make`’ to build the ACL2 executable, output from the build process is now written to file `make.log`. The terminal output is now minimal. If there is already a file `make.log`, it is first moved to `make.log.bak`. See file `GNUmakefile` for additional documentation. Thanks to Alessandro Coglio and Eric Smith for suggesting this change, and to David Rager for pointing out associated changes to make for Jenkins builds.

The startup banner now has a cleaner look (see startup-banner). Thanks to Alessandro Coglio and Eric Smith for key suggestions for improvement.

(CCL only) When an attempt to build ACL2 fails, a more informative error message is printed when the error is caused by invoking CCL as a soft link on the Unix PATH rather than as a regular file. Thanks to Mertcan Temel for feedback leading to this change.

A script `bin/acl2` has been added below the main ACL2 direcctory. It may be used in place of `saved_acl2`, invoked from any directory. If the `bin` subdirectory is on one’s Unix `PATH` then of course `acl2` will invoke this script (unless a different `acl2` is in a directory that is earlier on that path). Thanks to Alessandro Coglio for suggesting that an `acl2` script be made available with ACL2, and to him and Eric Smith for subsequent discussions on that topic.

When an error occurs while loading an ACL2-customization file, ACL2 quits with exit code 1. Thanks to Eric Smith for suggesting the quit and to Eric McCarthy for suggesting exit code 1 in that case.

ACL2 now does a more thorough job of doing proofs when “`make proofs`” is executed. In particular, proofs were formerly skipped but are now performed for `defun` forms containing the explicit `xargs` declaration, `:mode :logic`, and for `defthm` events evaluated with default-defun-mode `:logic` outside the so-called “pass 2 files”.

EMACS Support

The ACL2-doc search commands (‘`s`’ and ‘`S`’) were seen to use all available memory on a linux system, during the process of initializing the `acl2-doc-search` buffer that is used for doing the searching. That problem has been solved: that buffer is now loaded from a file that is built by the manual-building process and is downloaded by the acl2-doc ‘`D`’ command.

Highlighting in lisp-mode (inside Emacs) has been improved, both by recognizing more keywords and by highlighting of some new arguments, in particular the first argument of `defthm`. Thanks to Vivek Ramanathan, both for pointing out the `defthm` issue and for suggesting code that was incorporated into the changes.

The ACL2-doc browser now queries when first loading an ACL2+books manual if you have a newer web-based version. A “yes” response will use the (out-of-date) manual, while a “no” response will generally produce a new query asking if you want to download the manual from the web. This change was made in support of building the manual more quickly, as the acl2-doc manual is no longer built by default. See ACL2-doc for how to do that build and other details. Thanks to Alessandro Coglio, Eric Smith, and Sol Swords for discussions about speeding up the build of the manual.

Fixed certain hangs in the ACL2-doc browser. For example, when standing on whitespace near the left margin of topic *ACL2-exports*, the ’g’ command could formerly hang.

Experimental Versions

ACL2(p) now runs much more reliably on host Lisp SBCL (by taking advantage of a locking mechanism provided by SBCL). Thanks to David Rager for providing this improvement.

In ACL2(p), `set-waterfall-parallelism` no longer causes an error when there are `override-hints` if the argument is the existing value of the state global, `'waterfall-parallelism`, or upon a transition of waterfall-parallelism to `nil`. Thanks to David Rager for raising this issue (see GitHub Issue #1171) and discussing its resolution.

Subtopics

Note-8-4-books
Release notes for the ACL2 Community Books for ACL2 8.4