




# Updates to the ACL2 Community Books

---

(Centaur Edition)  
Sept. 2015-May 2017



# Broad Categories

---

Mostly improvements on existing libraries

- STD
- SV and VL (hardware modeling)
- FTY (type definitions)
- Ipasir incremental SAT solver interface
- Misc

# std/util/define(s)

---

- defret/defret-mutual -- theorems with return values already bound

```
(define foo-bar (x y z)
  :returns (mv (foo) (bar))
  ...
  ///
  (defret foo-preserves-natp
    (implies (natp x) (natp foo))))
```

- More DRY; reduces amount of code to modify when adding a formal or return value
- :hints ((... :expand (<call>)))
- :rule-classes ((:forward-chaining :trigger-terms (foo)))

# std/util/define(s)

---

More:

- `ret b* binder` -- automatically bind return values by name

```
(define foo-bar (x y z)
  :returns (mv (foo) (bar)) ... )
...
(b* (((ret fb) (foo-bar x y z)))
  (list fb.foo fb.bar))
```
- **Post-define hooks** (not documented)
  - `(local (std::add-default-post-define-hook :fix))` for FTY

# std/stobj

---

Mostly moved from centaur/misc, not strictly new

- Def-1d-arr, def-2d-arr
- Defabsstobj-events (submit all events necessary for defabsstobj)
- Defstobj-clone (create congruent stobj)

# FTY

---

- Generates xdoc documentation for type definitions
- Improved representation for memory efficiency in product types:
  - `(NIL . NIL) → NIL`
- Bitstructs:

```
(defbitstruct mxcsr
  (flags fp-flags-p)
  (daz bitp)
  (masks fp-flags-p) ...)
```
- Defvisitor -- generates code to traverse a complicated type hierarchy, do something to objects of certain types

# SV and VL

---

- Improved procedural statement support (break/continue/return)
- Supports sequential cosim tests
- Memory efficiency & performance improvements
- SVTV state machine mode (experimental, see “sv/tutorial/counter.lisp”)

# Ipasir incremental SAT interface

---

- Standard interface to incremental SAT libraries
- Logical story accurately (?) modeled by abstract stobj
- Shared library interface (no writing out files)
- Aignet integration



# Miscellaneous

---

Ongoing library development:

- centaur/bitops
- aignet (added abc connection)

Others:

- Satlink: use LRAT checker to verify unsat proofs
- GL -- new flex-bindings utility for complicated BDD variable orderings
- centaur/misc/bound-rewriter: utility for solving certain inequalities when nonlinear arithmetic is too slow

# Tracking Updates

---

(thanks Shilpi!)

Suggestion: Maintain book update notes as we go, in a common file

`docs/book-changes.txt` (?)

Somewhat less granular (but more detailed?) than commit messages


Incorporate into documentation (note-books-?.?) before releases



# Updates to the ACL2 Community Books

---

(Kestrel Edition)  
Sept. 2015-May 2017



# Kestrel Books

---

All new since the ACL2-2015 Workshop:

- `kestrel/abnf/`: ABNF (Augmented Backus-Naur Form) formalization, verified grammar parser, and grammar operations.
  - Described in a rump talk at the ACL2-2017 Workshop.
- `kestrel/soft/`: SOFT (Second-Order Functions and Theorems), a macro library to mimic second-order functions and theorems in ACL2.
  - Described in a paper at the ACL2-2015 Workshop.
  - A few updates since the paper, described in an XDOC topic.
- `kestrel/utilities/`: A collection of various utilities.
  - Described in the following slides.
  - Some contributed by Matt Kaufmann and Jared Davis.

# General-Purpose, Logic-Mode Utilities

---

- `*-theorems.lisp`: Theorems about things defined outside the Kestrel Books, e.g. lists, osets, terms.
- `characters.lisp`: Functions and theorems on (lists of) characters.
- `strings.lisp`: Functions and theorems on strings.
- `osets.lisp`: Functions and theorems about osets and types osets.
- `symbol-*-alists.lisp`: Typed alists defined via `std::defalist`.
- `nati.lisp`: Fixtype for natural numbers plus infinity.
- `integers-from-to.lisp`: Functions and theorems for lists/osets of integers from min to max.
- `typed-tuples.lisp`: Macro to recognize tuples with given component types.
- `maybe-msgp.lisp`: Recognizer for `msgp` or `nil`.
- `maybe-unquote.lisp`: Function to remove wrapping quote, if any.

# Utilities for Worlds and Terms

---

- `world-queries.lisp`: Query properties of functions, macros, theorems, events, and currently included books.
- `defun-sk-queries.lisp`: Recognize, and retrieve the constituents of, functions that *may* have been introduced via `defun-sk`.
- `defchoose-queries.lisp`: Recognize, and retrieve the constituents of, functions that have been introduced via `defchoose`.
- `term-utilities.lisp`: Recognizers, checkers, translators, and constructors for terms and lambdas.

Meant to complement the built-in world and term utilities (topic `system-utilities`).

# Utilities for Processing User Macro Inputs

---

- enumerations.lisp: Types of certain typical inputs.
- error-checking.lisp: Functions to check for erroneous conditions and generate soft errors with informative and consistent messages.
  - Mostly generated via a def-error-checker macro, also in that file.
- doublets.lisp: Function doublets-to-alist, inverse of built-in alist-to-doublets, with inversion theorems.
- prove-interface.lisp: Programmatic interface to the prover, e.g. to prove applicability conditions of program transformations, but more general.
- named-formulas.lisp: Manipulate named formulas, e.g. applicability conditions of program transformations.

# Utilities to Support Event Generation

---

- `event-forms.lisp`: Shallow recognizers of (lists of) event forms, and functions to generate function or theorem introduction macro variants.
- `install-not-norm-event.lisp`: Generator of install-not-normalized event forms.
- `fresh-names.lisp`: Make a name new by appending \$ signs as needed.
- `numbered-names.lisp`: Manage and generate names accompanied by numeric indices, e.g. `f{1}`, `f{2}`, ...
- `user-interface.lisp`: Control the output generated on the screen.
- `directed-untranslate.lisp`: Untranslate a term in a way that resembles a related given term, useful e.g. when transforming terms.
- `minimize-ruler-extenders.lisp` (1/2): Retrieve and manipulate ruler extenders.



# Other Utilities (1)

---

- `minimize-ruler-extenders.lisp` (2/2): Minimize the ruler extenders of the enclosed function definition.
- `auto-termination.lisp`: Attempt to prove the termination of the enclosed function by finding a matching termination theorem in the ACL2 world.
- `untranslate-preprocessing.lisp`: Macro to update the untranslation preprocessing function with a new constant to keep closed in screen output.
- `testing.lisp`: Macros to create tests, some based on `must-succeed/fail`.
- `ubi.lisp`: Undo history back to longest initial segment of `include-book` and related commands.

## Other Utilities (2)

---

- `define-sk.lisp`: A define-like version of `defun-sk`, with extended formals etc.
- `defmacroq.lisp`: Define a macro that quotes arguments not wrapped in `:eval`.
- `defthmr.lisp`: Define a theorem as a rewrite rule if possible.
- `acceptable-rewrite-rule-p.lisp`: Check if a proposed rewrite rule is acceptable.
- `copy-def.lisp`: Make a copy of a function definition and prove it equivalent.
- `make-termination-theorem.lisp`: Make a version of a function's termination theorem that calls stubs and thus is suitable for functional instantiation.
- `non-ascii-pathnames.lisp`: Support for file names with character codes above 255 (e.g. Unicode).
- `verify-guards-program.lisp`: Ephemeraly verify guards of program-mode functions, useful for validation.

# Remarks

---

- Some of the Kestrel Utilities could be moved to more central/fitting books.
  - A few Kestrel additions to other books already exists.
- Coming soon: `kestrel/apt/`, with an initial subset of APT (Automated Program Transformations), including the latest `simplify-defun`.
- The Kestrel Books are mostly based on the STD libraries.
- It would be nice to have more “unity” in some of the ACL2 Community Books.