Some ACL2 Features That Might Not Be Widely Known in the Community

Tutorial, ACL2 Workshop 2026

Matt Kaufmann (speaker) and
J Strother Moore

November 18, 2026

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

It's a whirlwind tour, focused on pointing to :DOC topics. It's definitely not an exhaustive list!

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

It's a whirlwind tour, focused on pointing to :DOC topics. It's definitely not an exhaustive list!

Slides are posted on the workshop page, with links to topics that you can explore later.

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

It's a whirlwind tour, focused on pointing to :DOC topics. It's definitely not an exhaustive list!

Slides are posted on the workshop page, with links to topics that you can explore later.

The "[DEMO] " mark indicates that I will run ACL2 using forms in file `demos.lsp`, which will also be posted, along with a log, `demos.log`.

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

It's a whirlwind tour, focused on pointing to :DOC topics. It's definitely not an exhaustive list!

Slides are posted on the workshop page, with links to topics that you can explore later.

The "[DEMO] " mark indicates that I will run ACL2 using forms in file `demos.lsp`, which will also be posted, along with a log, `demos.log`.

Feel free to ask questions. Just unmute and interrupt!

Plan

This tutorial is intended to raise awareness of ACL2 features that can boost productivity.

It's a whirlwind tour, focused on pointing to :DOC topics. It's definitely not an exhaustive list!

Slides are posted on the workshop page, with links to topics that you can explore later.

The "[DEMO] " mark indicates that I will run ACL2 using forms in file `demos.lsp`, which will also be posted, along with a log, `demos.log`.

Feel free to ask questions. Just unmute and interrupt!

Let's dive in....

- accumulated-persistence (also accumulated-persistence-subtleties):
  This is often used, but note options like `(show-accumulated-persistence :useless :list)` and note `(accumulated-persistence-oops)`.

- accumulated-persistence (also accumulated-persistence-subtleties):
  This is often used, but note options like
  `(show-accumulated-persistence :useless :list)`
  and note `(accumulated-persistence-oops)`.
- bookdata — Generate file `BK__bookdata.out`;
  let's look at the :DOC.

- accumulated-persistence (also accumulated-persistence-subtleties):
  This is often used, but note options like
  `(show-accumulated-persistence :useless :list)`
  and note `(accumulated-persistence-oops)`.
- bookdata — Generate file `BK__bookdata.out`;
  let's look at the :DOC.
- `break$`, `trace$`, `trace!`, break-on-error, open-trace-file, and close-trace-file:
  I often trace as follows to get a Lisp backtrace. [DEMO]

```
(trace$ (rewrite :entry (break$)))
:mini-proveall
:b ; for CCL (backtrace for SBCL, ...)
```

- accumulated-persistence (also accumulated-persistence-subtleties):
  This is often used, but note options like
  `(show-accumulated-persistence :useless :list)`
  and note `(accumulated-persistence-oops)`.
- bookdata — Generate file `BK__bookdata.out`;
  let's look at the :DOC.
- `break$`, `trace$`, `trace!`, break-on-error, open-trace-file, and close-trace-file:
  I often trace as follows to get a Lisp backtrace. [DEMO]

```
(trace$ (rewrite :entry (break$)))
:mini-proveall
:b ; for CCL (backtrace for SBCL, ...)
```

  But for lots of output on the screen, I often use the following and then look at file `"xx"`.

```
(open-trace-file "xx")
(trace$ ...)
<form>
(close-trace-file)
```

- break-rewrite (also see brr-near-missp)

- break-rewrite (also see brr-near-missp)
- case-match (show source code example, say, add-ignore-to-rest)

- break-rewrite (also see brr-near-missp)
- case-match (show source code example, say, add-ignore-to-rest)
- cw, fmx-cw, fmx, and set-guard-msg [DEMO]

- break-rewrite (also see brr-near-missp)
- case-match (show source code example, say, add-ignore-to-rest)
- cw, fmx-cw, fmx, and set-guard-msg [DEMO]
- ec-call and with-guard-checking [DEMO]

- break-rewrite (also see brr-near-missp)
- case-match (show source code example, say, add-ignore-to-rest)
- cw, fmx-cw, fmx, and set-guard-msg [DEMO]
- ec-call and with-guard-checking [DEMO]
- efficiency — It would be great to have more community contributions to this topic.

- break-rewrite (also see brr-near-missp)
- case-match (show source code example, say, add-ignore-to-rest)
- cw, fmx-cw, fmx, and set-guard-msg [DEMO]
- ec-call and with-guard-checking [DEMO]
- efficiency — It would be great to have more community contributions to this topic.
- fast-cert — This can speed up certification by avoiding the `include-book` check for local incompatibility, but it is unsound.

- measure-debug and guard-debug [DEMO]

```
(defun foo (x y)
  (declare (xargs :measure-debug t))
  (if (or (consp x) (consp y))
      (1+ (foo (cdr x) (cdr y)))
    0))
(defun f (x)
  (declare (xargs :measure (f x)
                  :measure-debug t))
  (if (and (posp x)
           (< (f (1- x)) x))
      (f (f (1- x)))
    0))
```

- measure-debug and guard-debug [DEMO]

```
(defun foo (x y)
  (declare (xargs :measure-debug t))
  (if (or (consp x) (consp y))
      (1+ (foo (cdr x) (cdr y)))
    0))
(defun f (x)
  (declare (xargs :measure (f x)
                  :measure-debug t))
  (if (and (posp x)
           (< (f (1- x)) x))
      (f (f (1- x)))
    0))
```

- oops, ubt, and ubu — [DEMO]

- print-control (also set-print-base-radix set-fmt-hard-right-margin)

- print-control (also set-print-base-radix set-fmt-hard-right-margin)
- print-gv [DEMO] (from :DOC):

```
(defun foo (x)
  (declare (xargs :guard
                  (and (integerp x)
                       (posp (+ -2 x))
                       (posp (+ 3 x))
                       (posp (+ -4 x)))))
  x)
(foo -1)
:print-gv
; Then evaluate the resulting FLET form.
(print-gv :conjunct t)
; Then evaluate the resulting FLET form.
```

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- profile (also profile-acl2 and profile-all)

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- profile (also profile-acl2 and profile-all)
- programming-with-state

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- profile (also profile-acl2 and profile-all)
- programming-with-state
- pso (also psof and note wof) and see set-gag-mode

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- profile (also profile-acl2 and profile-all)
- programming-with-state
- pso (also psof and note wof) and see set-gag-mode
- read-file-into-string – fast!

- printing-to-strings [DEMO] (essentially from :DOC):

```
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30))))
(fmt-to-string "Big list: ~x0"
               (list (cons #\0 (make-list 30)))
               :fmt-control-alist
               '((fmt-soft-right-margin . 10000)
                 (fmt-hard-right-margin . 10000)))
```

- profile (also profile-acl2 and profile-all)
- programming-with-state
- pso (also psof and note wof) and see set-gag-mode
- read-file-into-string – fast!
- rebuild — This can be a lot faster than LD.

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- ruler-extenders (also see induction-coarse-v-fine-grained)

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- ruler-extenders (also see induction-coarse-v-fine-grained)
- save-exec — Note that there are many options.

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- ruler-extenders (also see induction-coarse-v-fine-grained)
- save-exec — Note that there are many options.
- set-constraint-tracking — Let's look at the example in the :DOC.

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- ruler-extenders (also see induction-coarse-v-fine-grained)
- save-exec — Note that there are many options.
- set-constraint-tracking — Let's look at the example in the :DOC.
- set-deferred-ttag-notes

- redo-flat [DEMO]

```
(encapsulate ()
  (defun f1 (x) x)
  (defun f2 (x) x)
  (defun f3 (x) y)
  (defun f4 (x) x))
:redo-flat
:pbt 0
```

- ruler-extenders (also see induction-coarse-v-fine-grained)
- save-exec — Note that there are many options.
- set-constraint-tracking — Let's look at the example in the :DOC.
- set-deferred-ttag-notes
- set-induction-depth-limit and set-subgoal-loop-limits

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo'(a b c d e)
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo'(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo'(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo(acl2::cdr '(a b c d e))
```

- set-iprint (also without-evisc and set-evisc-tuple) [DEMO]
- set-raw-mode-on! and related raw-mode utilities
- set-raw-proof-format and set-raw-warning-format [DEMO]
- set-warnings-as-errors
- sharp-bang-reader — example:

```
ACL2 !>(defpkg "FOO" '(a c))
[[Output Elided]]
 "FOO"
ACL2 !>'#!foo(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo'(a b c d e)
(A FOO::B C FOO::D FOO::E)
ACL2 !>#!foo(acl2::cdr '(a b c d e))
(FOO::B C FOO::D FOO::E)
ACL2 !>
```

- system-attachments (e.g., heavy-linear-p and assume-true-false-aggressive-p; see also set-dwp)

- system-attachments (e.g., heavy-linear-p and assume-true-false-aggressive-p; see also set-dwp)
- system-utilities

- system-attachments (e.g., heavy-linear-p and assume-true-false-aggressive-p; see also set-dwp)
- system-utilities
- time$

- system-attachments (e.g., heavy-linear-p and assume-true-false-aggressive-p; see also set-dwp)
- system-utilities
- time$
- time-tracker

- system-attachments (e.g., heavy-linear-p and assume-true-false-aggressive-p; see also set-dwp)
- system-utilities
- time$
- time-tracker
- trans, trans1, and (maybe less well-known) trans!, trans*, and trans*- [DEMO] (also illustrates with-output)

```
(include-book "std/util/define" :dir :system)
; Probably not very helpful:
:trans1 (define f (x) (cons x x))
; DEFINITELY not very helpful:
:trans (define f (x) (cons x x))
; Shows the generated defun, etc.:
:trans* t (define f (x) (cons x x))
```

We can use `:trans*` on the resulting `defsection` call.

- walkabout [DEMO]

```
(walkabout (w state) state) ; then, walk
```

- walkabout [DEMO]

```
(walkabout (w state) state) ; then, walk
```

- wet: with-error-trace [DEMO]

```
(defun f1 (x) (car x))
(defun f2 (x) (f1 x))
(defun f3 (x) (f2 x))
(f3 7)
(wet (f3 7))
```

- walkabout [DEMO]

```
(walkabout (w state) state) ; then, walk
```

- wet: with-error-trace [DEMO]

```
(defun f1 (x) (car x))
(defun f2 (x) (f1 x))
(defun f3 (x) (f2 x))
(f3 7)
(wet (f3 7))
```

- with-brr-data
  Simple example from the :DOC: [DEMO]

```
(with-brr-data
   (thm (equal (append x y) (append y x))))
(cw-gstack-for-subterm (append y (cdr x)))
```

Other topics, as time permits

- advanced-features
- cons-with-hint
- df
- flet and macrolet
- gthm and tthm
- forward-chaining-reports
- history (pbt, pr, pl, pl2, pf, pe, pcb, and pcs)
- loop$
- make-event
- minimal-theory
- non-exec (also defun-nx and defund-nx)
- proof-builder
- puff and puff*
- reset-prehistory and disable-ubt
- rewrite-equiv

Other topics, as time permits (continued)

- set-subgoal-loop-limits
- sharp-f-reader and sharp-u-reader
- stobj and:
  - add-global-stobj
  - attach-stobj]
  - defabsstobj
  - defstobj
  - nested-stobjs
  - stobj-tables
  - swap-stobjs
  - with-global-stobj
  - with-local-stobj
- tc (also tcp and tca)
- with-prover-step-limit, set-prover-step-limit, and with-prover-time-limit
- wormhole