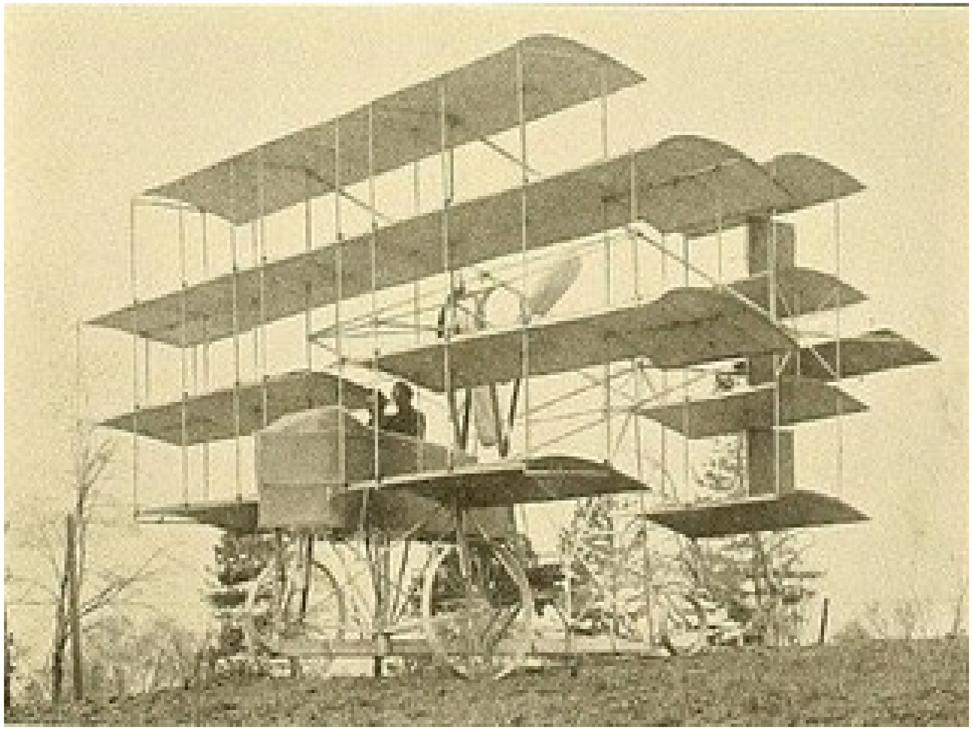
XDOC, and the Future of ACL2 Documentation

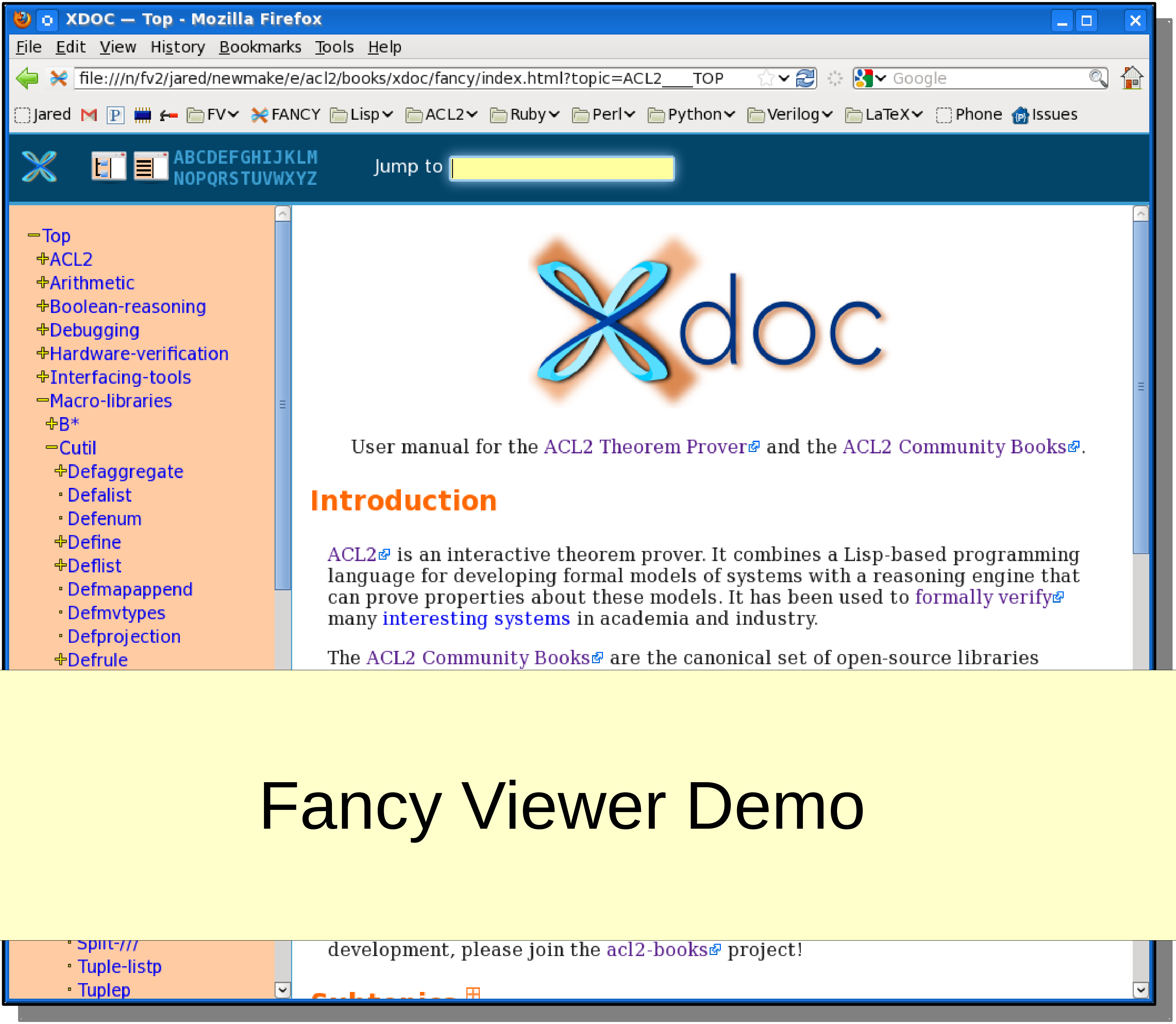


Jared Davis
jared@centtech.com

Part 1

Practical Stuff

What's an XDOC and where can I get one?



User manual for the ACL2 Theorem Prover and the ACL2 Community Books.

Introduction

ACL2 is an interactive theorem prover. It combines a Lisp-based programming language for developing formal models of systems with a reasoning engine that can prove properties about these models. It has been used to formally verify many interesting systems in academia and industry.

The ACL2 Community Books are the canonical set of open-source libraries

development, please join the acl2-books project!

Subtopics ⊞

Fancy Viewer Demo

How to document your books

(the tedious, manual way, for starters)

```
(include-book “xdoc/top” :dir :system)

(defxdoc str
  :short "ACL2 String Library"
  :long "<p>This is a rudimentary string library for ACL2.</p>

<p>The functions here are all in logic mode, with verified guards.  In
cases, some effort has been spent to make them both efficient and relat
straightforward to reason about.</p>

<h3>Loading the library</h3>

<p>Ordinarily, to use the library one should run</p>
@({
 (include-book \"str/top\" :dir :system)
})

<p>The documentation is then ava                                      o
library's functions are found i

<p>If you are willing to accept
@('fast-cat') book for faster st                                      r
details.</p>  ...”)
```

Documentation as Code

```
(include-book “xdoc/top” :dir :system)

(defxdoc str
  :short "ACL2 String Library"
  :long "<p>This is a rudimentary string library for ACL2.</p>

<p>The fun                                     h verified guards.  In
cases, some                                    oth efficient and relat
straightfor

<h3>Loading

<p>Ordinar                                     </p>
@({
 (include-book \"str/top\" :dir :system)
})

<p>The documentation is then available by typing @(':xdoc str').  All o
library's functions are found in the @('STR') package.</p>

<p>If you are willing to accept a trust tag, you may also include the
@('fast-cat') book for faster string-concatenation; see @(see cat) for
details.</p>  ...”)
```

Lightweight

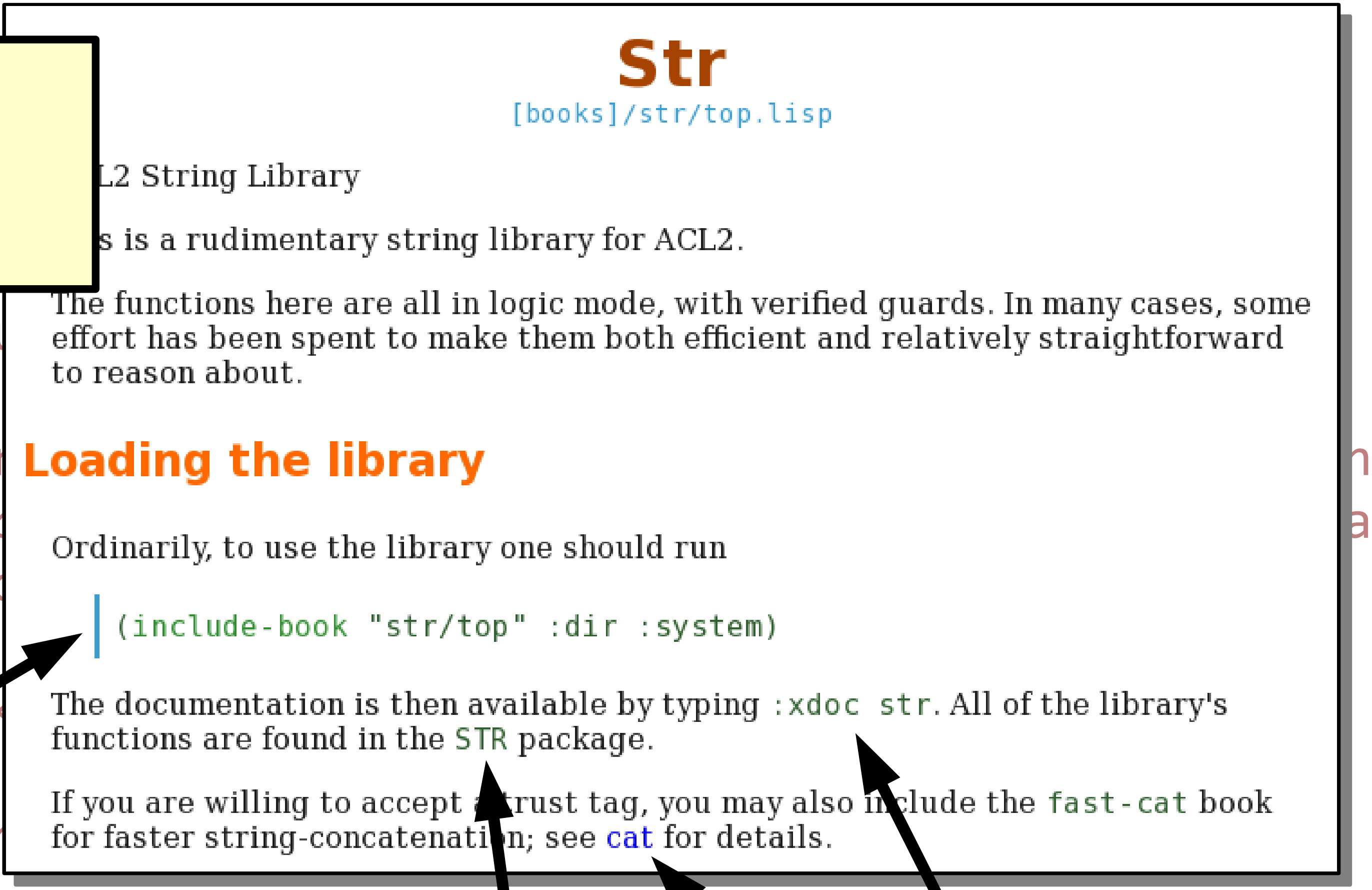
Loads Quickly (< 0.1 sec)

```
(include-book “xdoc/top” :dir :system)

(defxdoc str
  :short "ACL2 String Library"
  :long "<p>This is a rudimentary string library for ACL2.</p>

<p>The functions here are all in logic mode, with verified guards.  In
cases, some effort has been spent to make them both efficient and relat
straightforward to reason about.</p>

<h3>Loading th

<p>Ordinarily,                                                        >
@({
 (include-book
})

<p>The documentation is then available by typing @(':xdoc str').  All o
library's functions are found in the @('STR') package.</p>

<p>If you are willing to accept a trust tag, you may also include the
@('fast-cat') book for faster string-concatenation; see @(see cat) for
details.</p>  ...”)
```

Standard XML Syntax

Tags must be balanced!

Preprocessor!



Str

[books]/str/top.lisp

L2 String Library

s is a rudimentary string library for ACL2.

The functions here are all in logic mode, with verified guards. In many cases, some effort has been spent to make them both efficient and relatively straightforward to reason about.

Loading the library

Ordinarily, to use the library one should run

```
(include-book "str/top" :dir :system)
```

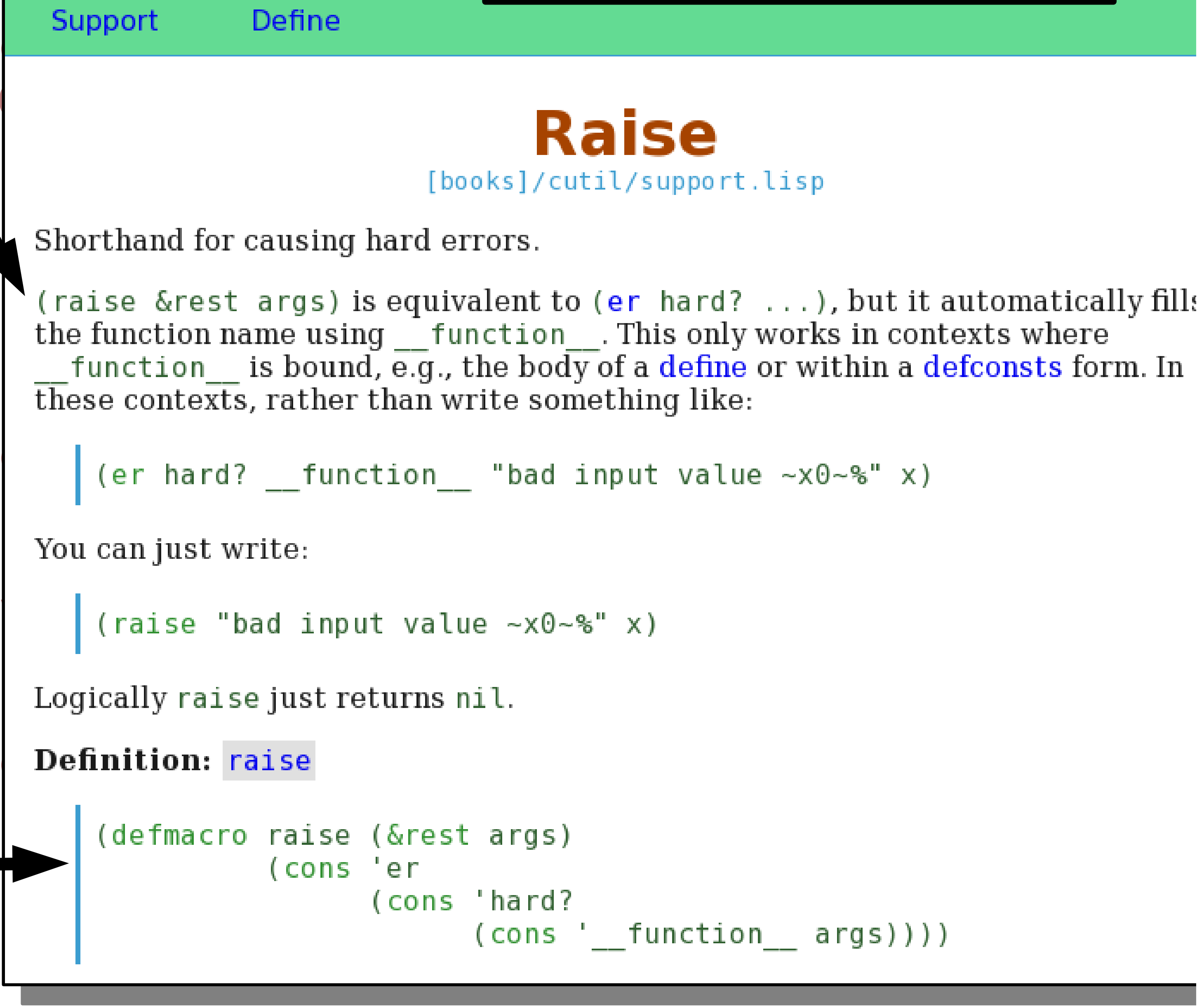
The documentation is then available by typing `:xdoc str`. All of the library's functions are found in the STR package.

If you are willing to accept a trust tag, you may also include the fast-cat book for faster string-concatenation; see cat for details.

```
  :long "<p>This is a ru

<p>The functions here ar                                                  n
cases, some effort has b                                                  at
straightforward to reaso

<h3>Loading the library<

<p>Ordinarily, to use th
@({
 (include-book \"str/top\" :dir :system)
})

<p>The documentation is then available by typing @(':xdoc str').  All o
library's functions are found in the @('STR') package.</p>

<p>If you are willing to accept a trust tag, you may also include the
@('fast-cat') book for faster string-concatenation; see @(see cat) for
details.</p>  ...”)
```

```
(defxdoc raise
  :parents (support define)
  :short "Shorthand for causing hard e
  :long "<p>@(call raise) is equivalen                        , b
automatically fills
works in contexts wh
define) or within a
write something like

@({
 (er hard? __functio
})

<p>You can just writ

@({
 (raise \"bad input
})

<p>Logically @('rais

@(def raise)"
```

Fights Bitrot!



Raise

[books]/cutil/support.lisp

Shorthand for causing hard errors.

(raise &rest args) is equivalent to (er hard? ...), but it automatically fills the function name using __function__. This only works in contexts where __function__ is bound, e.g., the body of a define or within a defconsts form. In these contexts, rather than write something like:

```
(er hard? __function__ "bad input value ~x0~%" x)
```

You can just write:

```
(raise "bad input value ~x0~%" x)
```

Logically raise just returns nil.

Definition: raise

```
(defmacro raise (&rest args)
          (cons 'er
                (cons 'hard?
                      (cons '__function__ args))))
```

xdoc

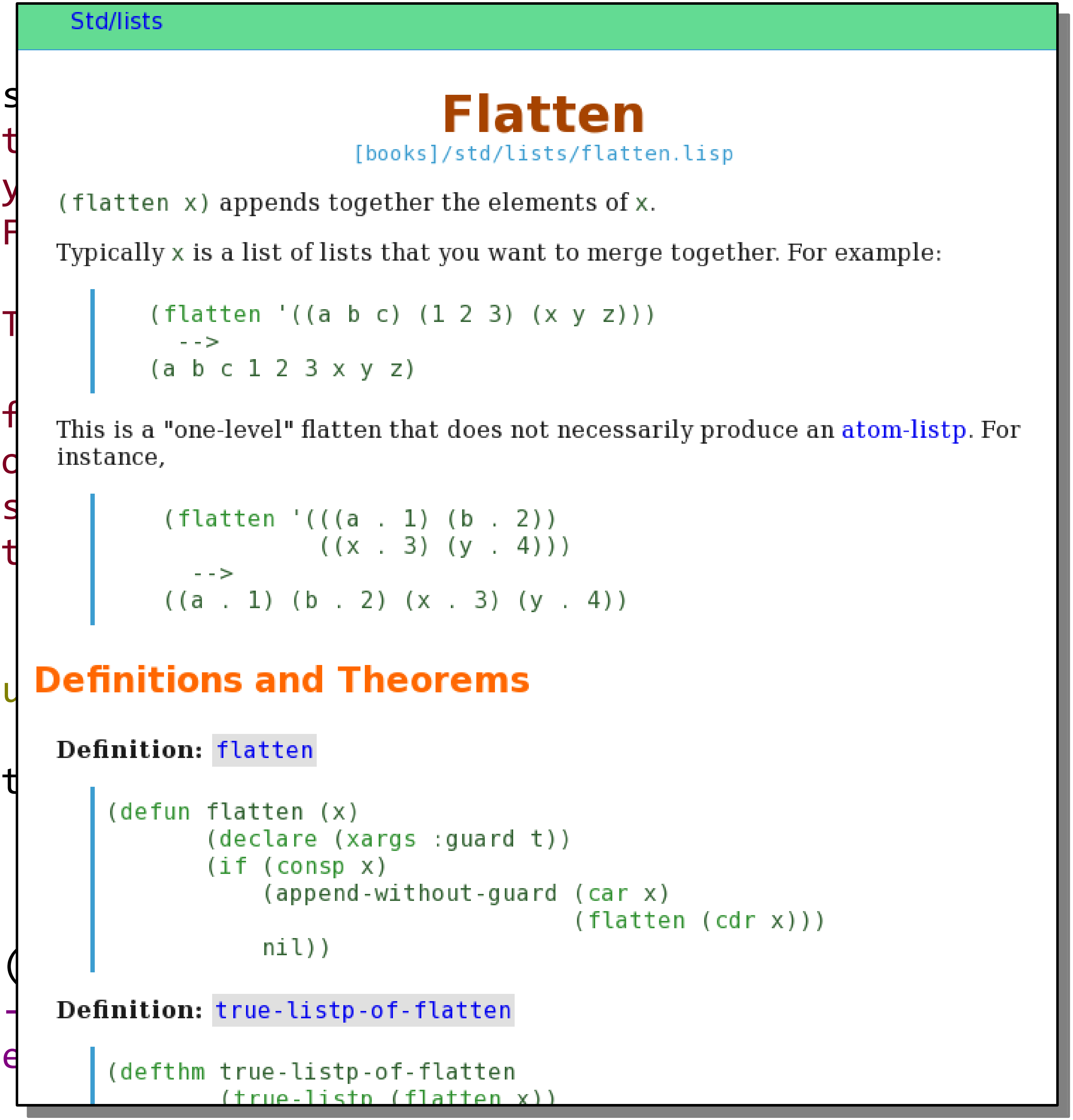
How to document your books

organize and

(the fancy, less tedious way)

```
(defxdoc flatten
  :parents (std/lists)
  :short "@(call flatten) appends together the elements of @('x').”
  :long "<p>Typically @('x') is a list of lists that you want
To merge together.  For example:</p>
...
<h3>Definitions and Theorems</h3>
@(def flatten)
@(thm true-listp-of-flatten)
@(thm flatten-when-not-consp)
@(thm flatten-of-cons)
@(thm flatten-of-list-fix) ...”)

(defund flatten (x)
  (declare (xargs :guard t))
  (if (consp x)
      (append-without-guard (car x) (flatten (cdr x)))
    nil))

(encapsulate ()
  (local (in-theory (enable flatten)))
  (defthm true-listp-of-flatten ...)
  (defthm flatten-when-not-consp ...)
  ...))
```

```
(defxdoc flatten
  :parents (std/lists
  :short "@(call flat
  :long "<p>Typically
To merge together.  F
...
<h3>Definitions and T
@(def flatten)
@(thm true-listp-of-f
@(thm flatten-when-no
@(thm flatten-of-cons
@(thm flatten-of-list

(defund flatten (x)
  (declare (xargs :gu
  (if (consp x)
      (append-without
    nil))

(encapsulate ()
  (local (in-theory (
  (defthm true-listp-
  (defthm flatten-whe
  ...))
```



Std/lists

Flatten

[books]/std/lists/flatten.lisp

(flatten x) appends together the elements of x.

Typically x is a list of lists that you want to merge together. For example:

```
    (flatten '((a b c) (1 2 3) (x y z)))
      -->
    (a b c 1 2 3 x y z)
```

This is a "one-level" flatten that does not necessarily produce an atom-listp. For instance,

```
     (flatten '(((a . 1) (b . 2))
                ((x . 3) (y . 4)))
       -->
     ((a . 1) (b . 2) (x . 3) (y . 4))
```

Definitions and Theorems

Definition: flatten

```
(defun flatten (x)
       (declare (xargs :guard t))
       (if (consp x)
           (append-without-guard (car x)
                                 (flatten (cdr x)))
           nil))
```

Definition: true-listp-of-flatten

```
(defthm true-listp-of-flatten
        (true-listp (flatten x))
(defxdoc flatten
  :parents (std/lists)
  :short "@(call flatten) appends together the elements of @('x').”
  :long "<p>Typically @('x') is a list of lists that you want
To merge together.  For example:</p>
...
<h3>Definitions and Theorems</h3>
@(def flatten)
@(thm true-listp-of-flatten)
@(thm flatten-when-not-consp)
@(thm flatten-of-cons)
@(thm flatten-of-list-fix) ...”)

(defund flatten (x)
  (declare (xargs :guard t))
  (if (consp x)
      (append-without-guard (car x) (flatten (cdr x)))
    nil))

(encapsulate ()
  (local (in-theory (enable flatten)))
  (defthm true-listp-of-flatten ...)
  (defthm flatten-when-not-consp ...)
  ...))
```

Not very DRY!

```
(defsection flatten
  :parents (std/lists)
  :short "@(call flatten) appends together the elements of @('x').”
  :long "<p>Typically @('x') is a list of lists that you want
To merge together.  For example:</p>
[example1]
[example2]”

  (defund flatten (x)
    (declare (xargs :guard t))
    (if (consp x)
        (append-without-guard (car x) (flatten (cdr x)))
      nil))

  (local (in-theory (enable flatten)))
  (defthm true-listp-of-flatten ...)
  (defthm flatten-when-not-consp ...)
  ...)
```

DRYer
Organizes books
Improves :pbt
Indents nicely

xdoc

How to organize and document your books

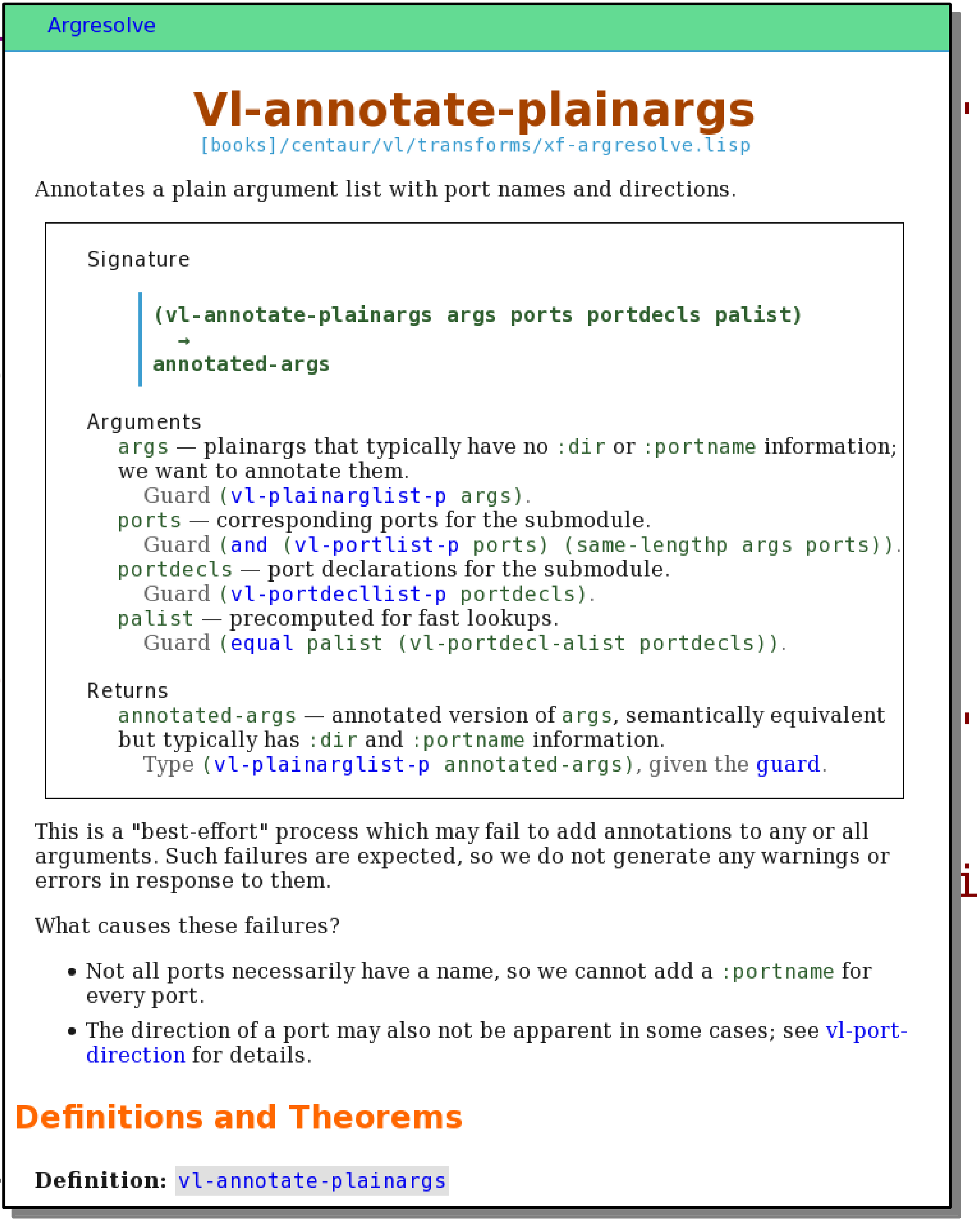
even better

(with less typing and stuff)

```
(defsection flatten
  :parents (std/lists)
  :short "@(call flatten) appends together the elements of @('x').”
  :long "<p>Typically @('x') is a list of lists that you want
To merge together.  For example:</p>
[example1]
[example2]”

  (defund flatten (x)
    (declare (xargs :guard t))
    (if (consp x)
        (append-without-guard (car x) (flatten (cdr x)))
      nil))

  (local (in-theory (enable flatten)))

  (defthm true-listp-of-flatten
    (true-listp (flatten x))
    :rule-classes :type-prescription)

  (defthm flatten-when-not-consp ...)

  ...)
```

```
(define vl-annotate-plainargs
  ((args       "plainargs that typically have no @(':dir') or @(':
                information; we want to annotate them."
               vl-plainarglist-p)
   (ports      "corresponding ports for the submodule"
               (and (vl-portlist-p ports)
                    (same-lengthp args ports)))
   (portdecls "port declarations for the submodule"
               vl-portdecllist-p)
   (palist     "precomputed for fast lookups"
               (equal palist (vl-portdecl-alist portdecls))))
  :returns
  (annotated-args "annotated version of @('args'), semantically e
                   but typically has @(':dir') and @(':portname')
                  vl-plainarglist-p :hyp :fguard)
  :parents (argresolve)
  :short "Annotates a plain argument list with port names and di
  :long "<p>This is a \"best-effort\" process ...”

  (b* (((when (atom args))
         nil)
        (name (vl-port->name (car ports)))
        (expr (vl-port->expr (car ports)))
    ...)
```

```
(define vl-annotate-p
  ((args      "plainargs th
               information;
              vl-plainargli
   (ports     "correspondin
              (and (vl-port
                   (same-le
   (portdecls "port declara
              vl-portdeclli
   (palist    "precomputed
              (equal palist
  :returns
  (annotated-args "annotate
                   but typi
                  vl-plaina
  :parents (argresolve)
  :short "Annotates a plain
  :long "<p>This is a \"bes

  (b* (((when (atom args))
        nil)
       (name (vl-port->name
       (expr (vl-port->expr
    ...)
```



Vl-annotate-plainargs

[books]/centaur/vl/transforms/xf-argresolve.lisp

Annotates a plain argument list with port names and directions.

Signature

```
(vl-annotate-plainargs args ports portdecls palist)
  →
annotated-args
```

Arguments

- `args` — plainargs that typically have no `:dir` or `:portname` information; we want to annotate them.
  Guard `(vl-plainarglist-p args)`.
- `ports` — corresponding ports for the submodule.
  Guard `(and (vl-portlist-p ports) (same-lengthp args ports))`.
- `portdecls` — port declarations for the submodule.
  Guard `(vl-portdecllist-p portdecls)`.
- `palist` — precomputed for fast lookups.
  Guard `(equal palist (vl-portdecl-alist portdecls))`.

Returns

- `annotated-args` — annotated version of `args`, semantically equivalent but typically has `:dir` and `:portname` information.
  Type `(vl-plainarglist-p annotated-args)`, given the guard.

This is a "best-effort" process which may fail to add annotations to any or all arguments. Such failures are expected, so we do not generate any warnings or errors in response to them.

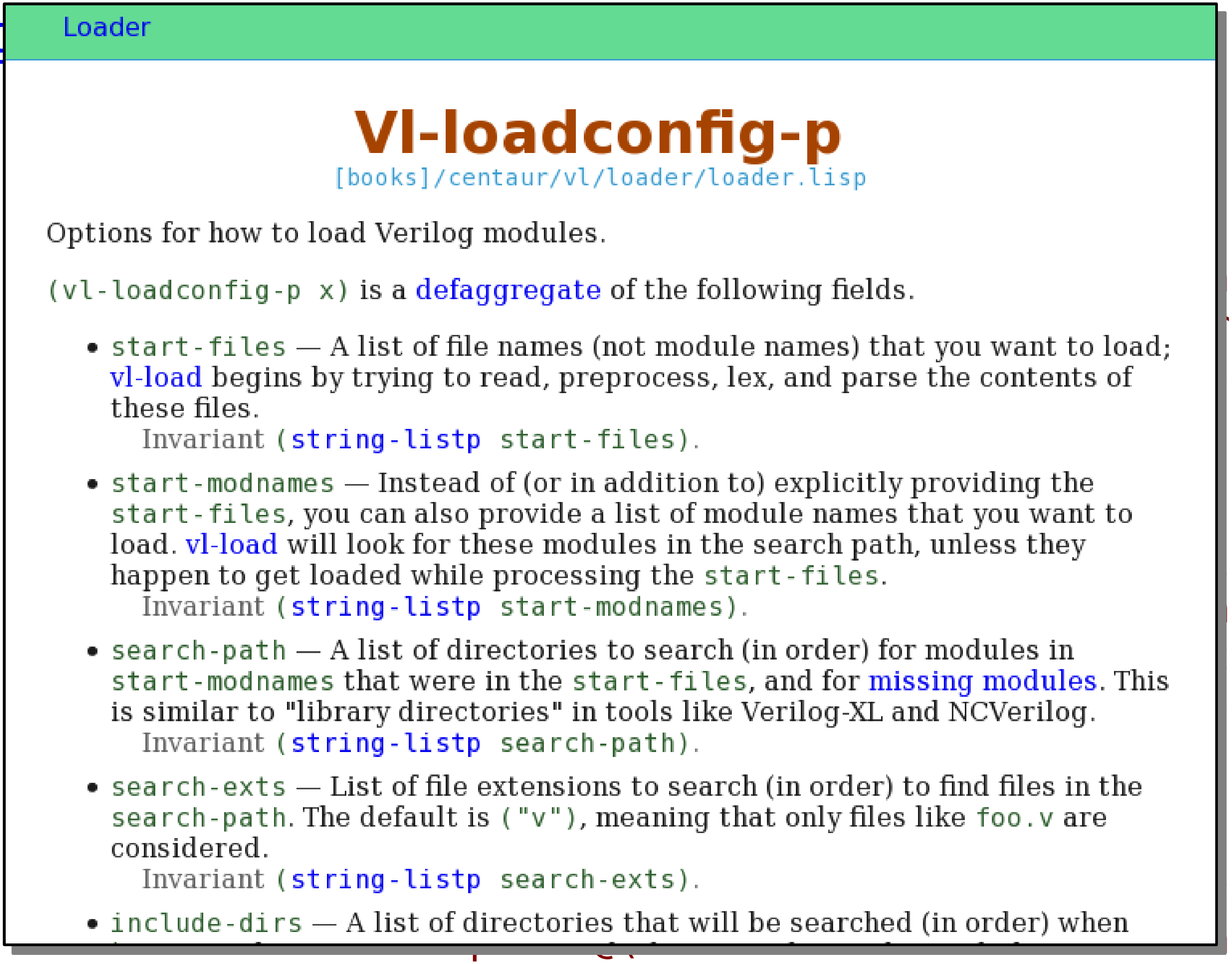
What causes these failures?

- Not all ports necessarily have a name, so we cannot add a `:portname` for every port.
- The direction of a port may also not be apparent in some cases; see vl-port-direction for details.

Definitions and Theorems

Definition: `vl-annotate-plainargs`

```
(defaggregate vl-loadconfig
  :parents (loader)
  :short "Options for how to load Verilog modules."

  ((start-files    string-listp
                   "A list of file names (not module names) that
                    load; @(see vl-load) begins by trying to reac
                    lex, and parse the contents of these files.")

   (start-modnames string-listp
                   "Instead of (or in addition to) explicitly pro
                    @('start-files'), you can also provide a list
                    names that you want to load.  @(see vl-load)
                    these modules in the search path, unless they
                    loaded while processing the @('start-files').

   (search-path    string-listp
                   "A list of directories to search (in order) fo
                    @('start-modnames') that were in the @('start
                    for <see topic='@(url vl-modulelist-missing)'
                    modules</see>.  This is similar to \"library
                    in tools like Verilog-XL and NCVerilog.")
   ...)
(defaggreg
  :parents (
  :short "Op

  ((start-fi
                                                                    at
                                                                    eac
                                                                    .")

    (start-mo
                                                                    prc
                                                                    ist
                                                                    d)
                                                                    hey
                                                                    '))


    (search-p
                                                                     fo
                                                                    art
                                                                    g)
                      modules</see>.  This is similar to \"library
                      in tools like Verilog-XL and NCVerilog.")

    ...)
```



Loader

Vl-loadconfig-p

[books]/centaur/vl/loader/loader.lisp

Options for how to load Verilog modules.

`(vl-loadconfig-p x)` is a defaggregate of the following fields.

- `start-files` — A list of file names (not module names) that you want to load; vl-load begins by trying to read, preprocess, lex, and parse the contents of these files.
  Invariant `(string-listp start-files)`.
- `start-modnames` — Instead of (or in addition to) explicitly providing the `start-files`, you can also provide a list of module names that you want to load. vl-load will look for these modules in the search path, unless they happen to get loaded while processing the `start-files`.
  Invariant `(string-listp start-modnames)`.
- `search-path` — A list of directories to search (in order) for modules in `start-modnames` that were in the `start-files`, and for missing modules. This is similar to "library directories" in tools like Verilog-XL and NCVerilog.
  Invariant `(string-listp search-path)`.
- `search-exts` — List of file extensions to search (in order) to find files in the `search-path`. The default is `("v")`, meaning that only files like `foo.v` are considered.
  Invariant `(string-listp search-exts)`.
- `include-dirs` — A list of directories that will be searched (in order) when

Macros like these aren't hard.

Documentation as Data

The full docs are just a table with a list of topics.

How to get a fancy manual with your stuff in it

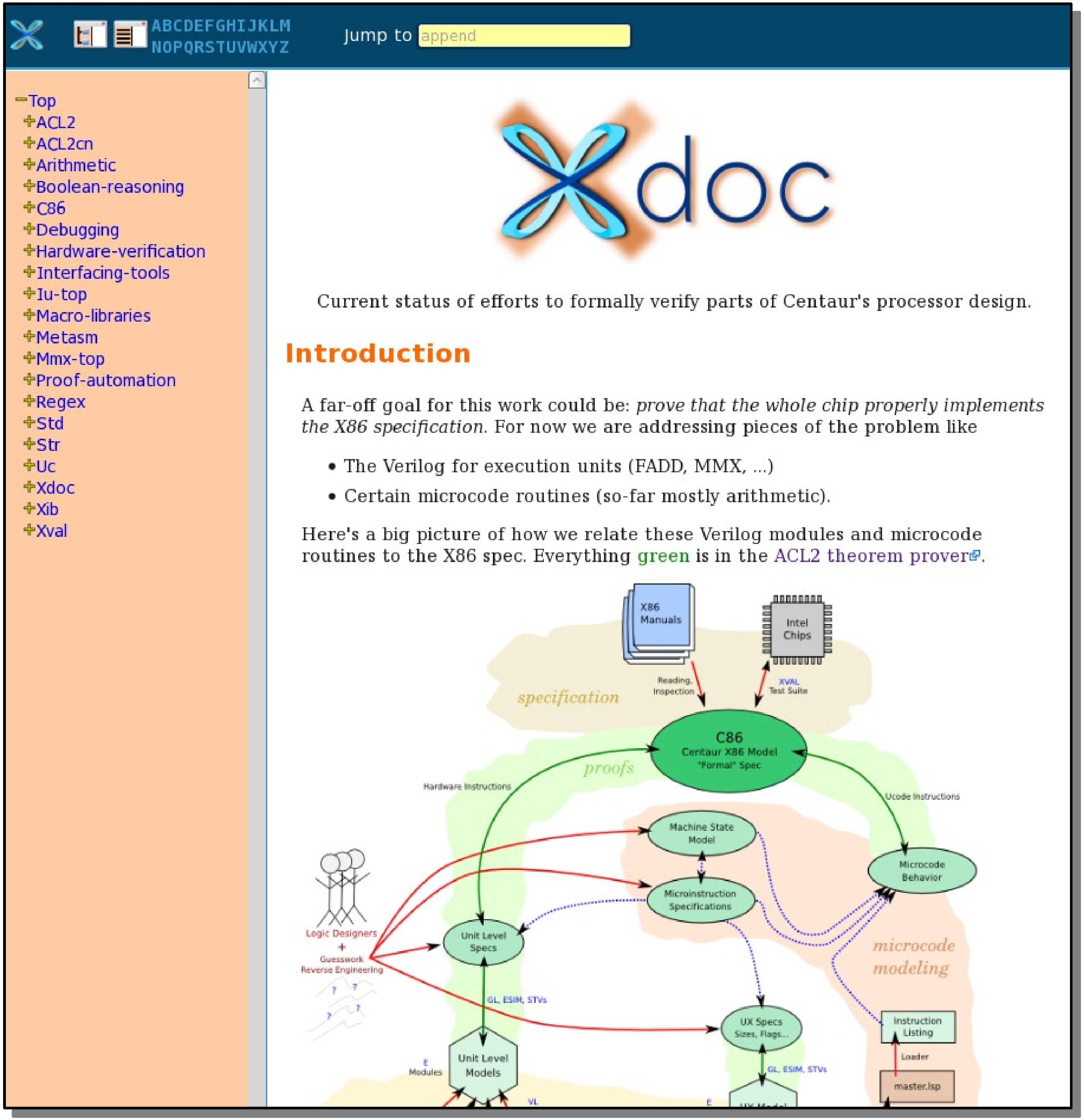
(so you can show your friends)

Xdoc

How to get a fancy manual
with your stuff in it

```
(include-book “your-books”)
(xdoc::save “./my-manual”)
```

(by the way, it's embeddable)



Current status of efforts to formally verify parts of Centaur's processor design.

Introduction

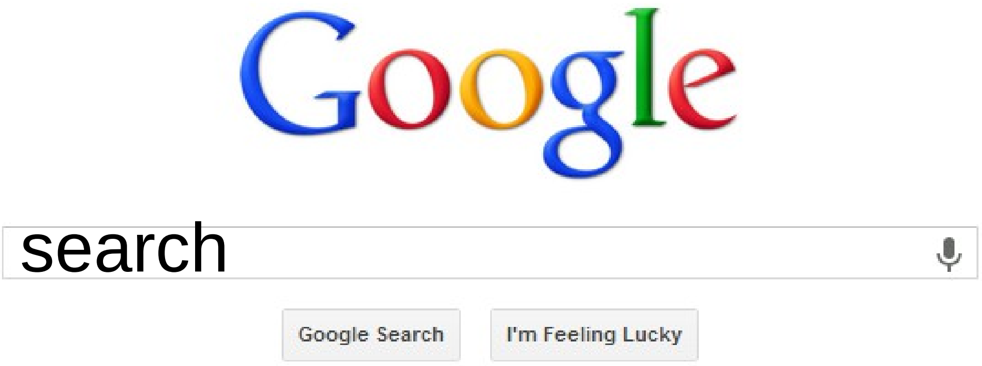
A far-off goal for this work could be: *prove that the whole chip properly implements the X86 specification*. For now we are addressing pieces of the problem like

- The Verilog for execution units (FADD, MMX, ...)
- Certain microcode routines (so-far mostly arithmetic).

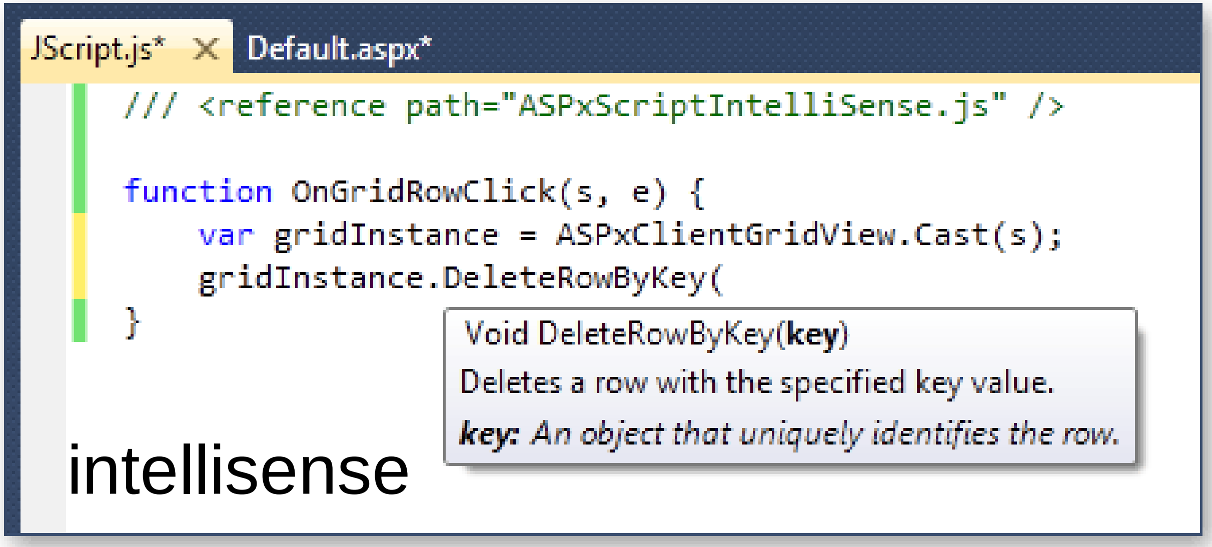
Here's a big picture of how we relate these Verilog modules and microcode routines to the X86 spec. Everything green is in the ACL2 theorem prover.

XDOC
at
Centaur

Demo



search



```
/// <reference path="ASPxScriptIntelliSense.js" />

function OnGridRowClick(s, e) {
    var gridInstance = ASPxClientGridView.Cast(s);
    gridInstance.DeleteRowByKey(
}
```

Void DeleteRowByKey(**key**)

Deletes a row with the specified key value.

key: *An object that uniquely identifies the row.*

intellisense

Like
+1
Follow

[edit source] [add a note]

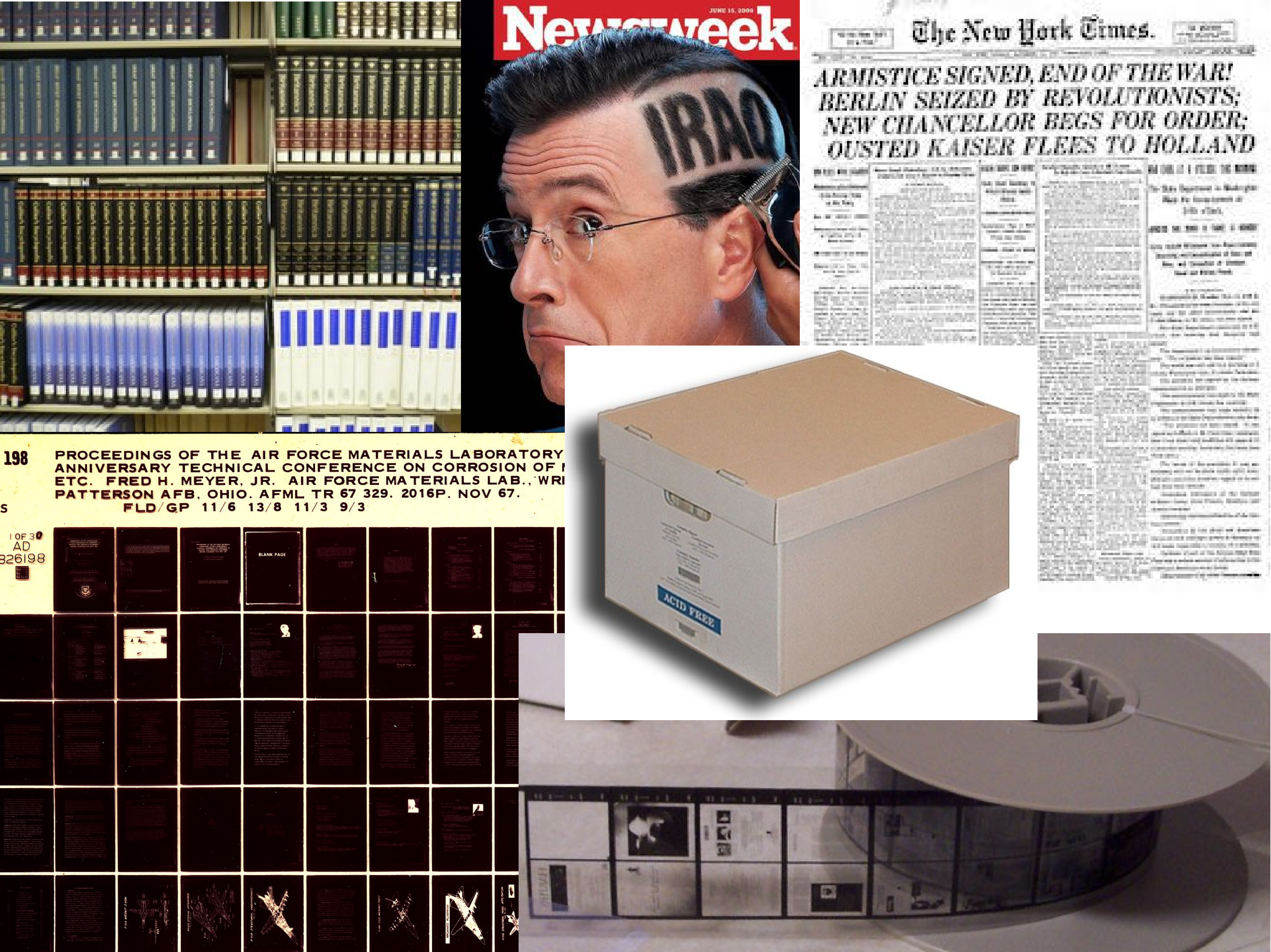
remixes

Wisdom
Linguistics
Alchemy
Order Magic
Distortion Magic
Magic Light
Transmute
Chaos Magic
Meditation
Summoner
Destruction
Concentration
Thesis
Higher Magic

Part 2

Impractical Stuff

The future of ACL2 documentation

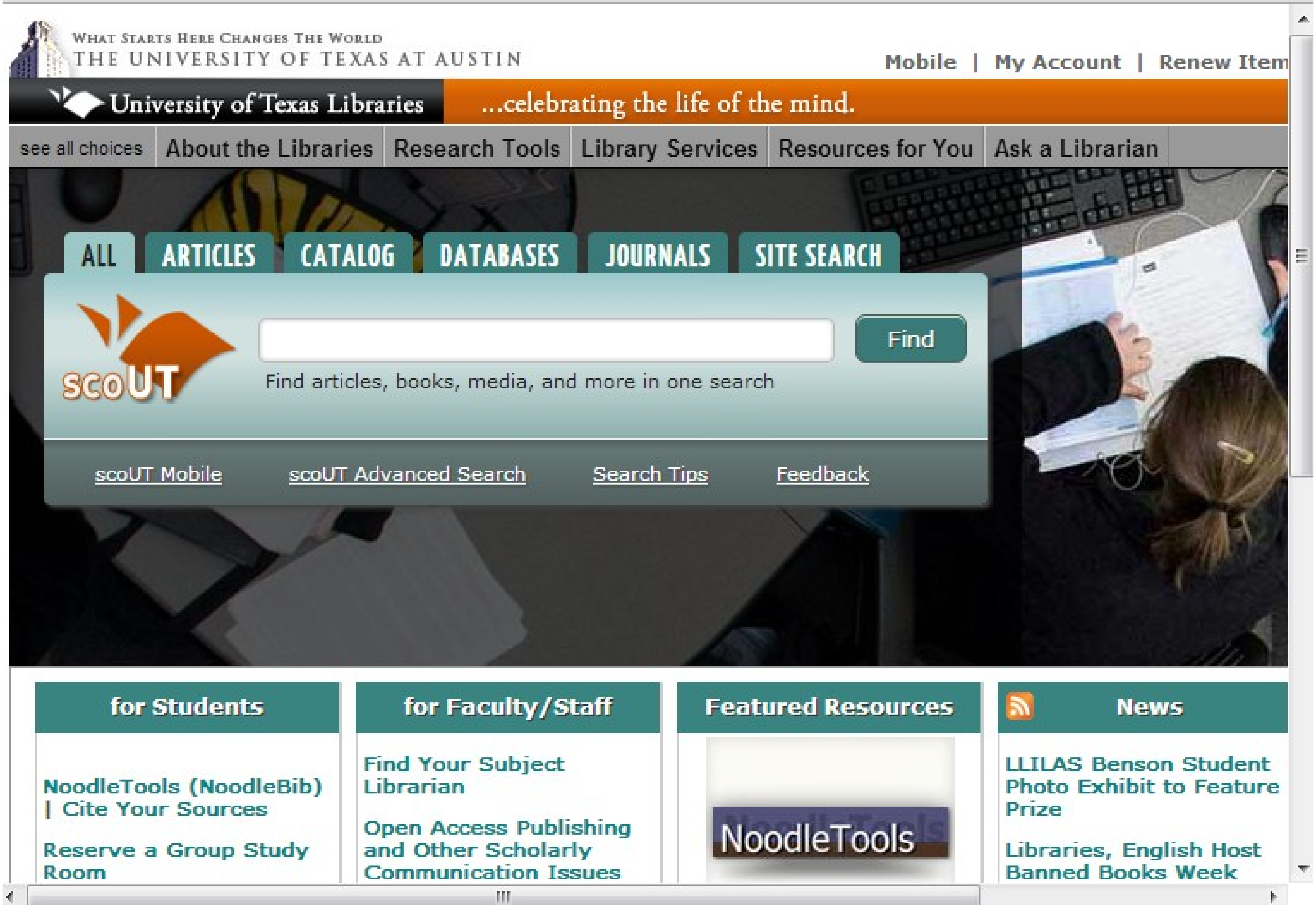


The New York Times.

ARMISTICE SIGNED, END OF THE WAR!
BERLIN SEIZED BY REVOLUTIONISTS;
NEW CHANCELLOR BEGS FOR ORDER;
OUSTED KAISER FLEES TO HOLLAND

198 PROCEEDINGS OF THE AIR FORCE MATERIALS LABORATORY
ANNIVERSARY TECHNICAL CONFERENCE ON CORROSION OF M
ETC. FRED H. MEYER, JR. AIR FORCE MATERIALS LAB., WRI
PATTERSON AFB, OHIO. AFML TR 67 329. 2016P. NOV 67.
S FLD/GP 11/6 13/8 11/3 9/3

1 OF 3
AD
826198



Find articles, books, media, and more in one search

for Students

NoodleTools (NoodleBib) | Cite Your Sources

Reserve a Group Study Room

for Faculty/Staff

Find Your Subject Librarian

Open Access Publishing and Other Scholarly Communication Issues

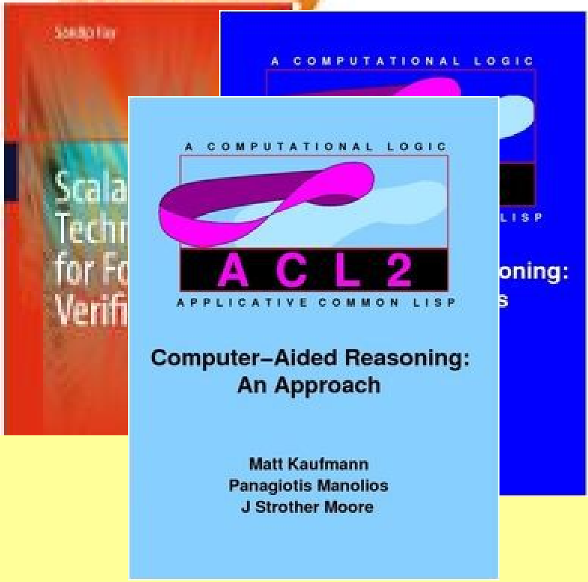
Featured Resources

News

LLILAS Benson Student Photo Exhibit to Feature Prize

Libraries, English Host Banned Books Week

ACL2


Sandip Ray

Scala
Techn
for Fo
Verifi

A COMPUTATIONAL LOGIC

ACL2

APPLICATIVE COMMON LISP

Computer-Aided Reasoning: An Approach

Matt Kaufmann
Panagiotis Manolios
J Strother Moore

Proceedings of the 11th International Workshop on the ACL2 Theorem Prover and its Applications

ACL2 WORKSHOP 2013

Ruben Gamboa
Jared Davis

xdoc

???

Matt Kaufmann
Lawrence C. Paulson (Eds.)

LNCS 6172

Interactive Theorem Proving

FLoC 2010

Springer

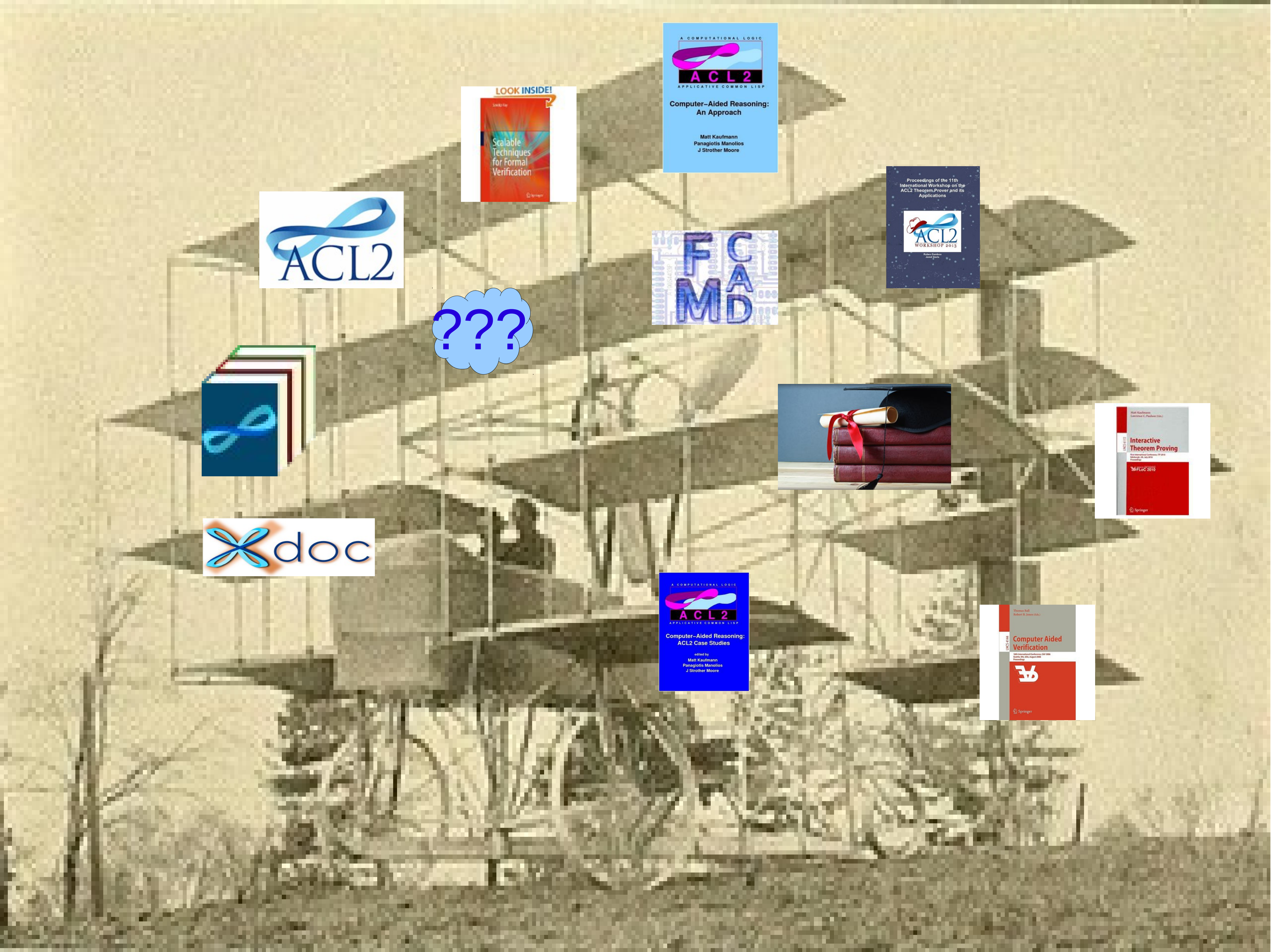
Thomas Ball
Robert B. Jones (Eds.)

LNCS 4144

Computer Aided Verification

18th International Conference, CAV 2006
Seattle, WA, USA, August 2006
Proceedings

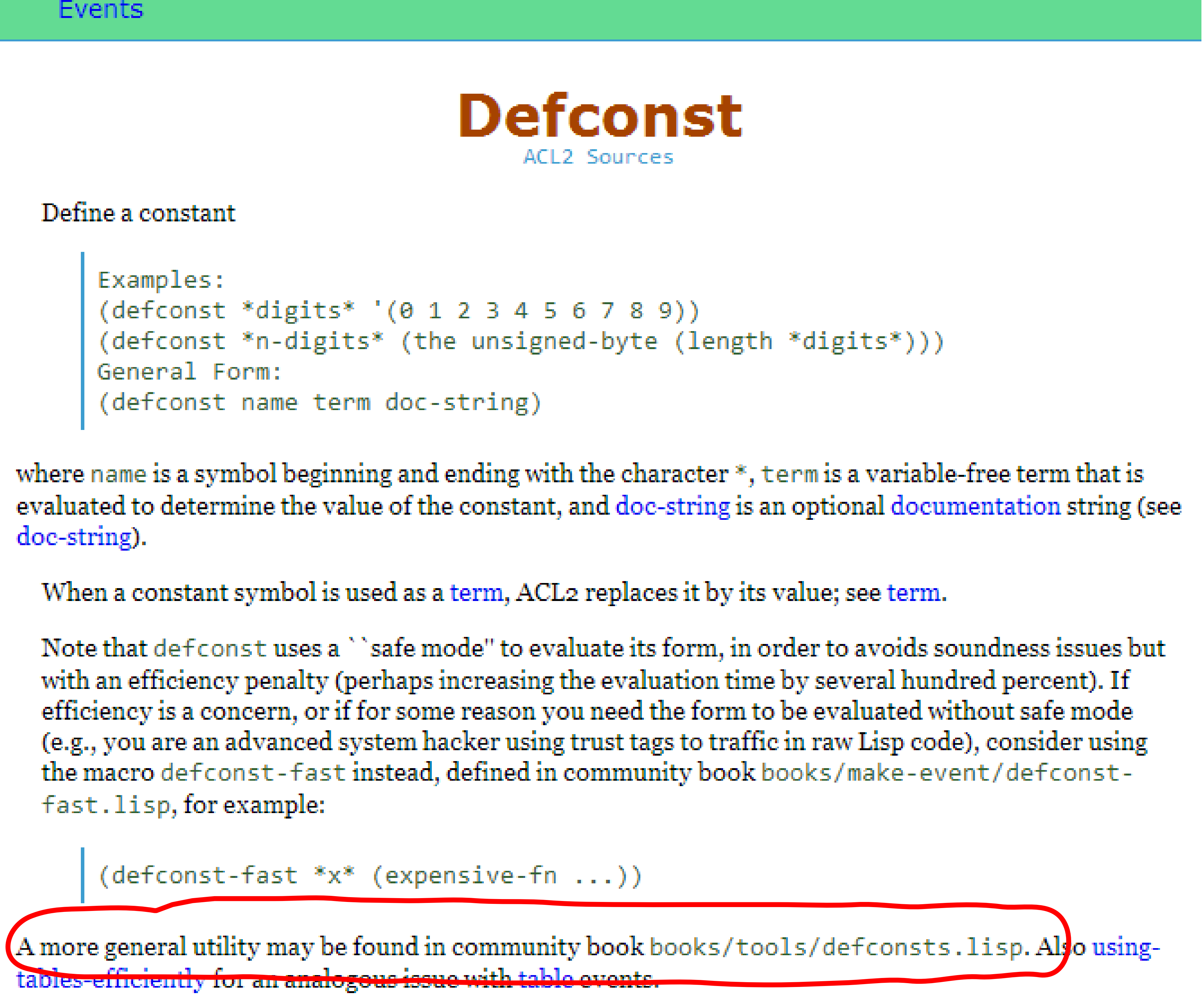
FMCAD


LOOK INSIDE!
Sandip Ray
Scalable Techniques for Formal Verification
Springer
A COMPUTATIONAL LOGIC
ACL2
APPLICATIVE COMMON LISP
Computer-Aided Reasoning: An Approach
Matt Kaufmann
Panagiotis Manolios
J Strother Moore
ACL2
Proceedings of the 11th International Workshop on the ACL2 Theorem Prover and its Applications
ACL2
WORKSHOP 2013
Ruben Gamboa
Jared Davis
FMCAD
???
Xdoc
Matt Kaufmann
Lawrence C. Paulson (Eds.)
LNCS 6172
Interactive Theorem Proving
First International Conference, ITP 2010
Edinburgh, UK, July 2010
Proceedings
FLoC 2010
Springer
A COMPUTATIONAL LOGIC
ACL2
APPLICATIVE COMMON LISP
Computer-Aided Reasoning: ACL2 Case Studies
edited by
Matt Kaufmann
Panagiotis Manolios
J Strother Moore
Thomas Ball
Robert B. Jones (Eds.)
LNCS 4144
Computer Aided Verification
18th International Conference, CAV 2006
Seattle, WA, USA, August 2006
Proceedings
Springer

We really ought to unify this.

Scal
Techn
for Fo
Verifi
A COMPU LOGIC
ACL2
APPLICATIVE COMMON LISP
Computer-Aided Reasoning
ACL2 Case Studies
edited by
Matt Kau
Moore
Comp
J Strother Moore
Proc
Int
Theorem Prover and its Applications
ACL2
WORKSHOP 20
Ruben Gamboa
Jared Davis
LNCS 6172
Interact
Theorem Pro
FLoC 2010
Robert B. Jones
LNCS 4144
Computer Aided Verification
18th International Conference, CAV 2006
Seattle, WA, USA, August 2006
Proceedings
FMCAD
ACL2
Xdoc
???

1. We should really integrate the book and system docs.



Defconst

ACL2 Sources

Define a constant

```
Examples:
(defconst *digits* '(0 1 2 3 4 5 6 7 8 9))
(defconst *n-digits* (the unsigned-byte (length *digits*)))
General Form:
(defconst name term doc-string)
```

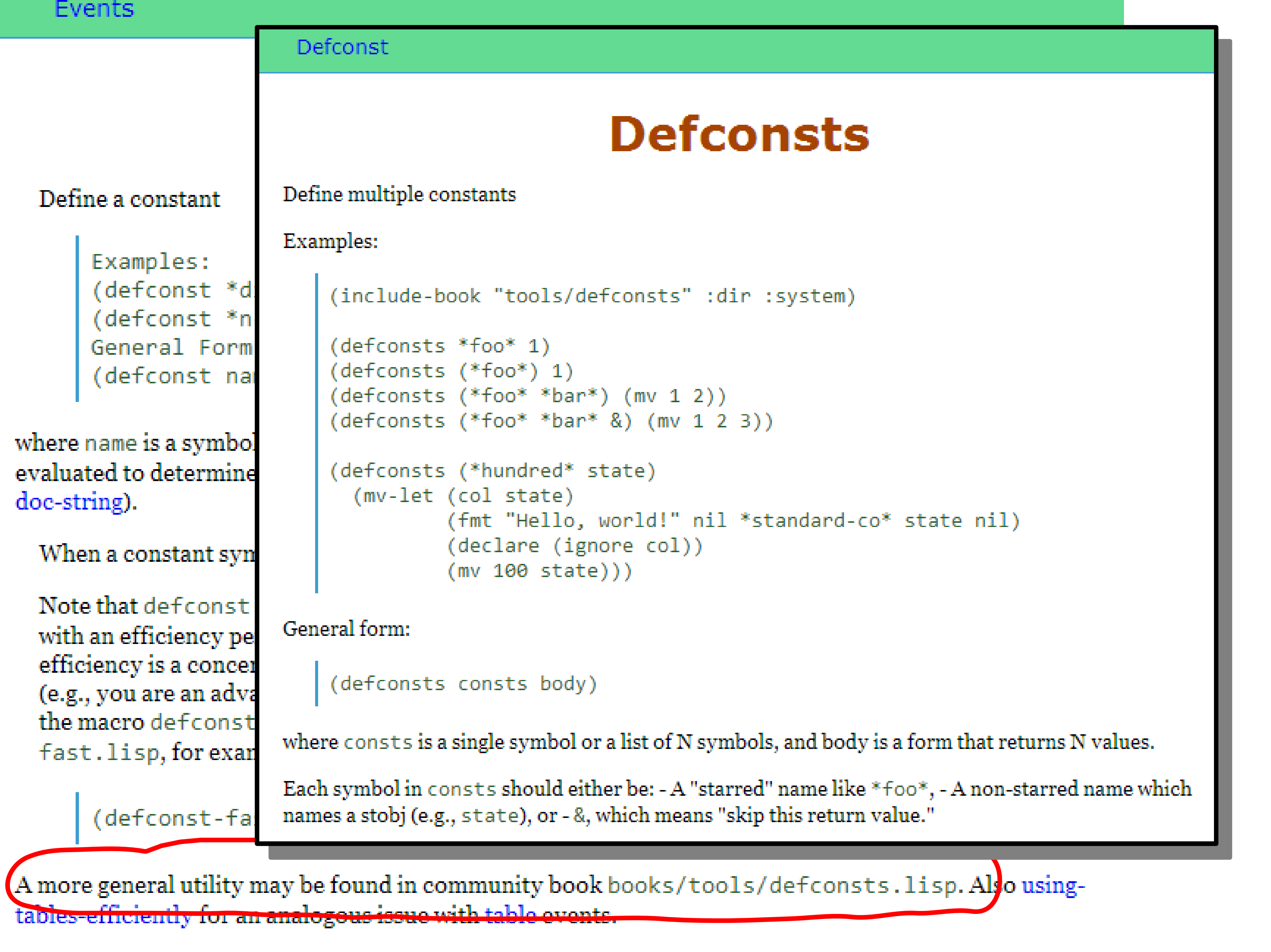
where `name` is a symbol beginning and ending with the character `*`, `term` is a variable-free term that is evaluated to determine the value of the constant, and doc-string is an optional documentation string (see doc-string).

When a constant symbol is used as a term, ACL2 replaces it by its value; see term.

Note that `defconst` uses a ``safe mode'' to evaluate its form, in order to avoids soundness issues but with an efficiency penalty (perhaps increasing the evaluation time by several hundred percent). If efficiency is a concern, or if for some reason you need the form to be evaluated without safe mode (e.g., you are an advanced system hacker using trust tags to traffic in raw Lisp code), consider using the macro `defconst-fast` instead, defined in community book `books/make-event/defconst-fast.lisp`, for example:

```
(defconst-fast *x* (expensive-fn ...))
```

A more general utility may be found in community book `books/tools/defconsts.lisp`. Also using-tables-efficiently for an analogous issue with table events.



Events

Define a constant

```
Examples:
(defconst *d
(defconst *n
General Form
(defconst na
```

where name is a symbol
evaluated to determine
doc-string).

When a constant sym

Note that defconst
with an efficiency pe
efficiency is a concer
(e.g., you are an adva
the macro defconst
fast.lisp, for exan

```
(defconst-fa
```

A more general utility may be found in community book books/tools/defconsts.lisp. Also using-tables-efficiently for an analogous issue with table events.

Defconst

Defconsts

Define multiple constants

Examples:

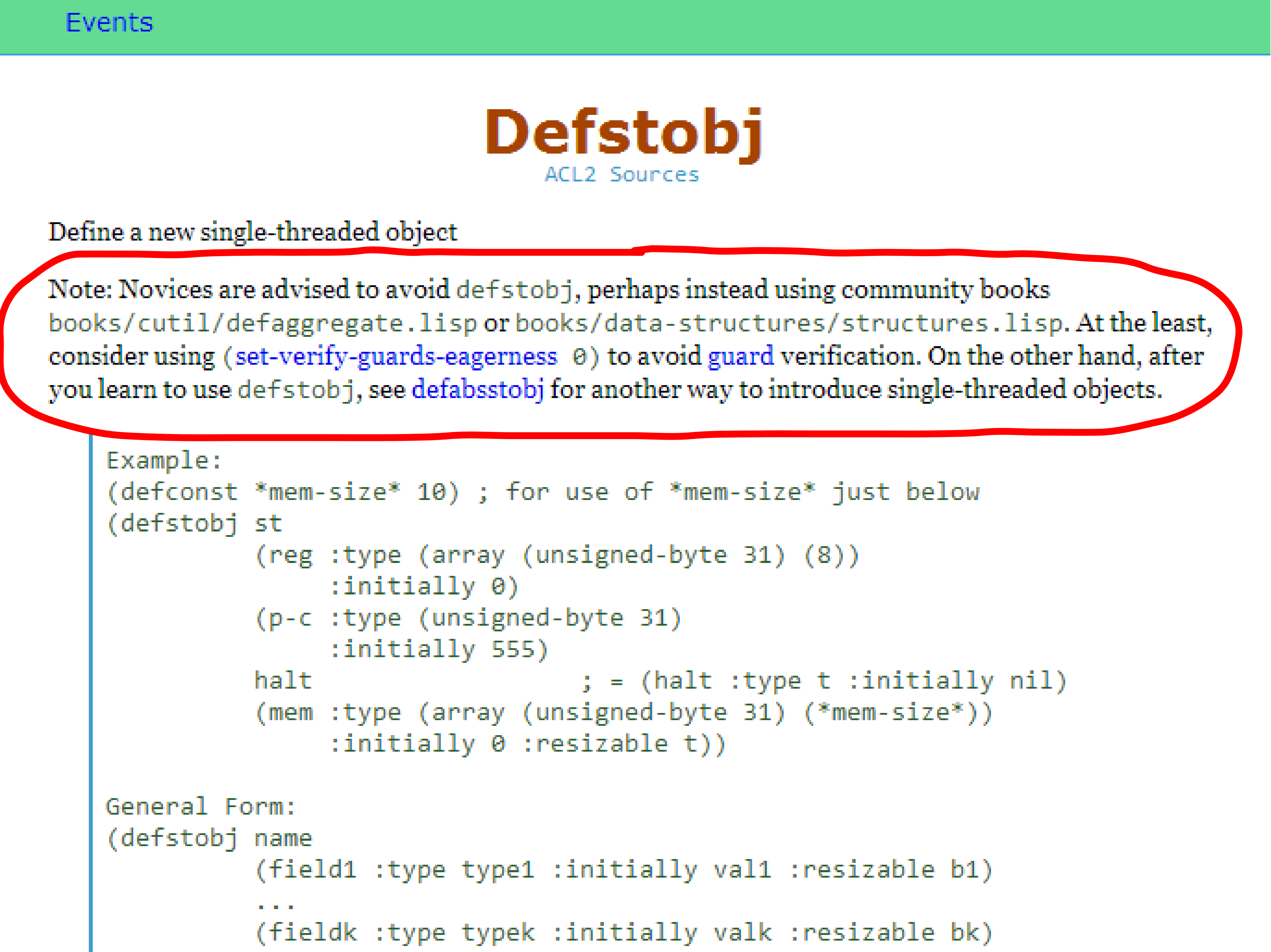
```
(include-book "tools/defconsts" :dir :system)

(defconsts *foo* 1)
(defconsts (*foo*) 1)
(defconsts (*foo* *bar*) (mv 1 2))
(defconsts (*foo* *bar* &) (mv 1 2 3))

(defconsts (*hundred* state)
  (mv-let (col state)
          (fmt "Hello, world!" nil *standard-co* state nil)
          (declare (ignore col))
          (mv 100 state)))
```

General form:

```
(defconsts consts body)
```

where consts is a single symbol or a list of N symbols, and body is a form that returns N values.

Each symbol in consts should either be: - A "starred" name like *foo*, - A non-starred name which names a stobj (e.g., state), or - &, which means "skip this return value."



Events

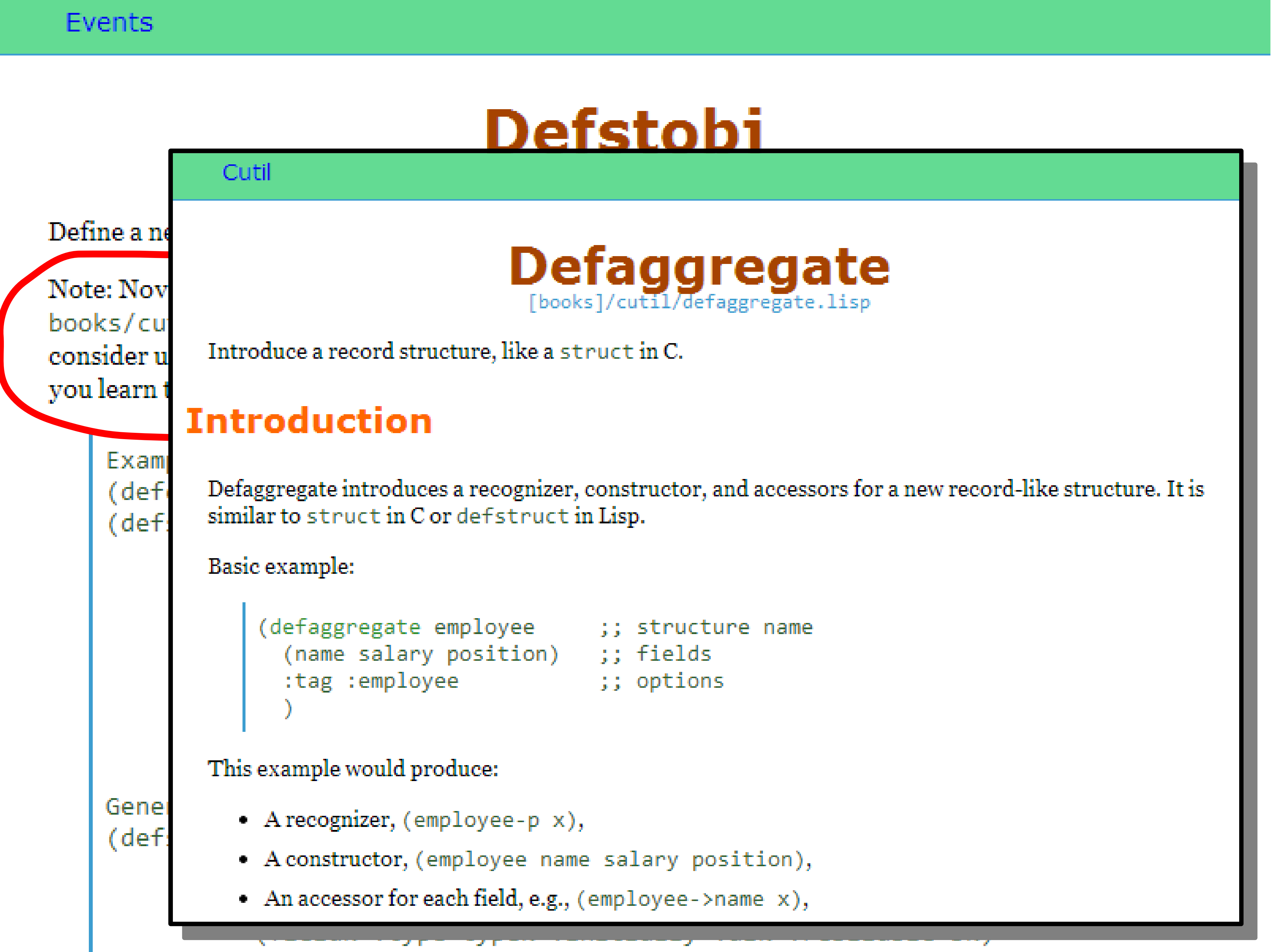
Defstobj

ACL2 Sources

Define a new single-threaded object

Note: Novices are advised to avoid defstobj, perhaps instead using community books books/cutil/defaggregate.lisp or books/data-structures/structures.lisp. At the least, consider using (set-verify-guards-eagerness 0) to avoid guard verification. On the other hand, after you learn to use defstobj, see defabsstobj for another way to introduce single-threaded objects.

```
Example:
(defconst *mem-size* 10) ; for use of *mem-size* just below
(defstobj st
          (reg :type (array (unsigned-byte 31) (8))
               :initially 0)
          (p-c :type (unsigned-byte 31)
               :initially 555)
          halt                  ; = (halt :type t :initially nil)
          (mem :type (array (unsigned-byte 31) (*mem-size*))
               :initially 0 :resizable t))

General Form:
(defstobj name
          (field1 :type type1 :initially val1 :resizable b1)
          ...
          (fieldk :type typek :initially valk :resizable bk)
```



Defstobj

Define a ne

Note: Nov
books/cu
consider u
you learn t

```
Examp
(def
(def
```

```
Gene
(def
```

Defaggregate

[books]/cutil/defaggregate.lisp

Introduce a record structure, like a `struct` in C.

Introduction

Defaggregate introduces a recognizer, constructor, and accessors for a new record-like structure. It is similar to `struct` in C or `defstruct` in Lisp.

Basic example:

```
(defaggregate employee     ;; structure name
  (name salary position)   ;; fields
  :tag :employee           ;; options
  )
```

This example would produce:

- A recognizer, `(employee-p x)`,
- A constructor, `(employee name salary position)`,
- An accessor for each field, e.g., `(employee->name x)`,

Io

ACL2 Sources

Input/output facilities in ACL2

```
Example:
(mv-let
  (channel state)
  (open-input-channel "foo.lisp" :object state)
  (mv-let (eofp obj state)
          (read-object channel state)
          (.
            .
             (let ((state (close-input-channel channel state)))
                   (mv final-ans state))..)))
```

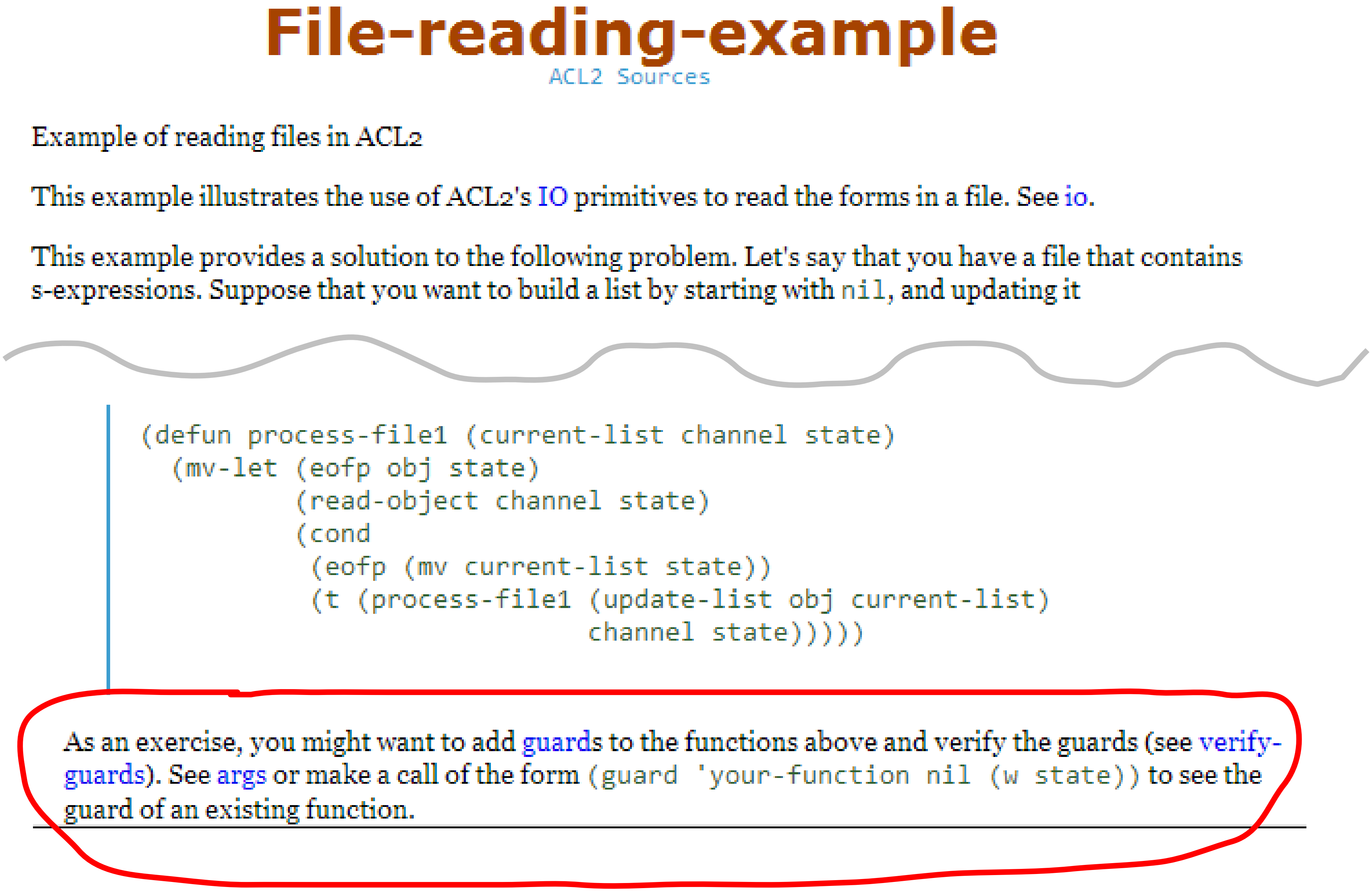
Also see file-reading-example.

For advanced ways to control printing, see print-control.

For a discussion of formatted printing, see fmt.

To control ACL2 abbreviation (``evisceration'') of objects before printing them, see set-evisc-tuple, see without-evisc, and see set-iprint.

To redirect output to a file, see output-to-file.



File-reading-example

ACL2 Sources

Example of reading files in ACL2

This example illustrates the use of ACL2's IO primitives to read the forms in a file. See io.

This example provides a solution to the following problem. Let's say that you have a file that contains s-expressions. Suppose that you want to build a list by starting with nil, and updating it

```
(defun process-file1 (current-list channel state)
  (mv-let (eofp obj state)
          (read-object channel state)
          (cond
           (eofp (mv current-list state))
           (t (process-file1 (update-list obj current-list)
                             channel state)))))
```

As an exercise, you might want to add guards to the functions above and verify the guards (see verify-guards). See args or make a call of the form (guard 'your-function nil (w state)) to see the guard of an existing function.

File-reading-example

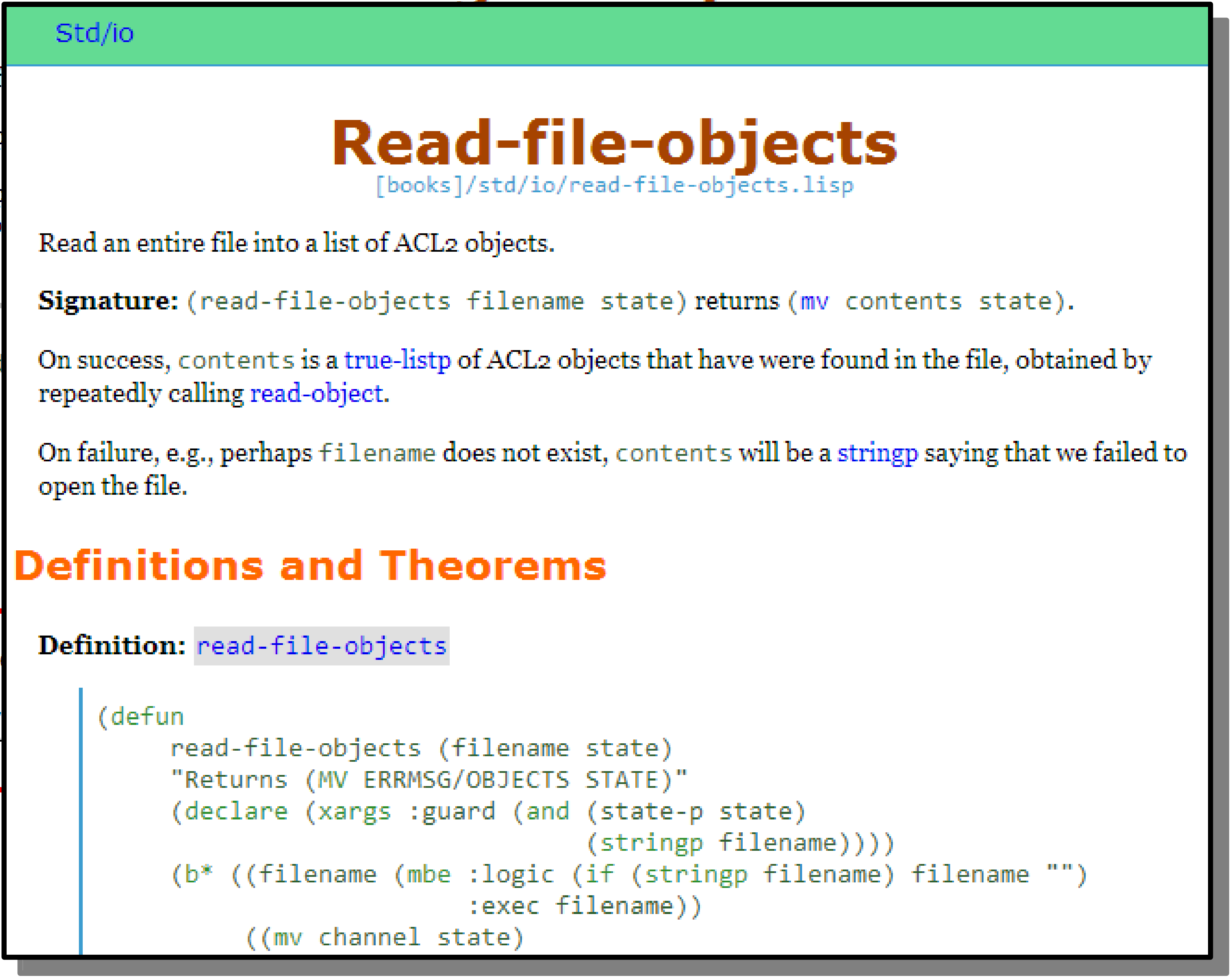
Example of

This examp

This examp
s-expressio

(d

As an ex
guards).
guard of



Std/io

Read-file-objects

[books]/std/io/read-file-objects.lisp

Read an entire file into a list of ACL2 objects.

Signature: `(read-file-objects filename state)` returns `(mv contents state)`.

On success, `contents` is a true-listp of ACL2 objects that have were found in the file, obtained by repeatedly calling read-object.

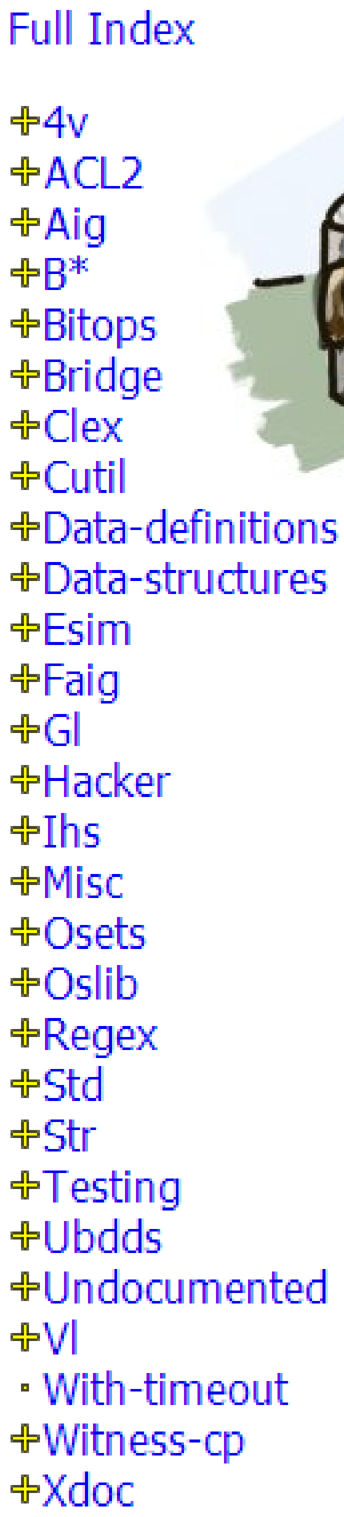
On failure, e.g., perhaps `filename` does not exist, `contents` will be a stringp saying that we failed to open the file.

Definitions and Theorems

Definition: `read-file-objects`

```
(defun
     read-file-objects (filename state)
     "Returns (MV ERRMSG/OBJECTS STATE)"
     (declare (xargs :guard (and (state-p state)
                                 (stringp filename))))
     (b* ((filename (mbe :logic (if (stringp filename) filename "")
                         :exec filename))
          ((mv channel state)
```

2. We should really improve our topic hierarchy.



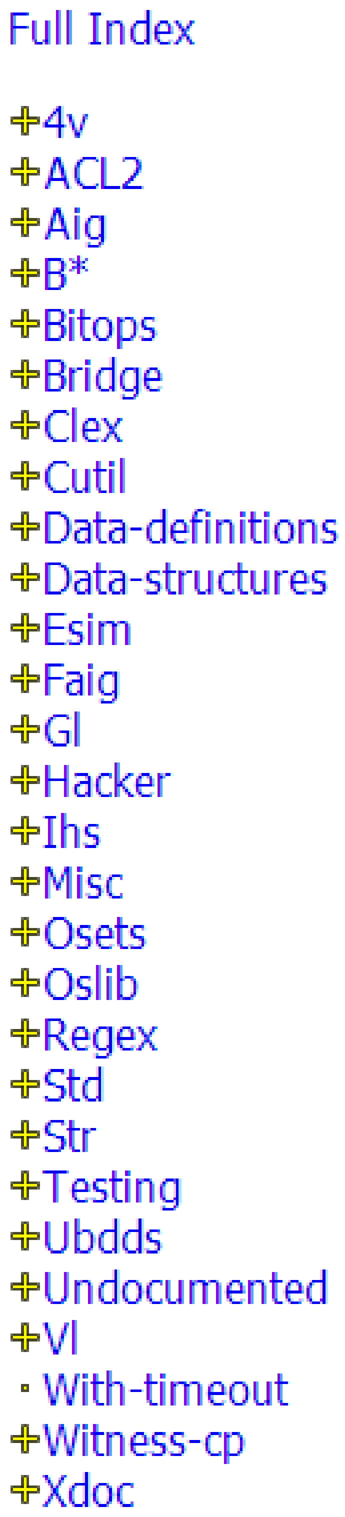
Full Index

+4v
+ACL2
+Aig
+B*
+Bitops
+Bridge
+Clex
+Cutil
+Data-definitions
+Data-structures
+Esim
+Faig
+Gl
+Hacker
+Ihs
+Misc
+Osets
+Oslib
+Regex
+Std
+Str
+Testing
+Ubdds
+Undocumented
+Vl
· With-timeout
+Witness-cp
+Xdoc

-- The topics on the left side are descriptive but kind of a hodge-podge. For example, perhaps "osets" could be under a topic named "sets", and it doesn't seem to me that "esim" is an intuitive name unless one knows the history...

-- We could use an "introduction to the books" topic that could be the default page and have a link to it sit above "full index" in the top left frame...

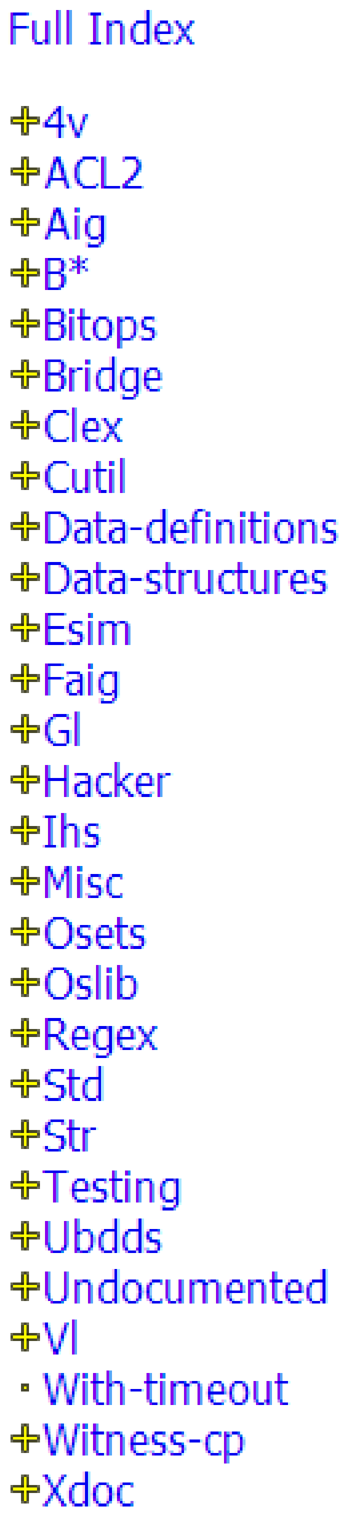
David Rager, acl2-books Issue 63



Full Index

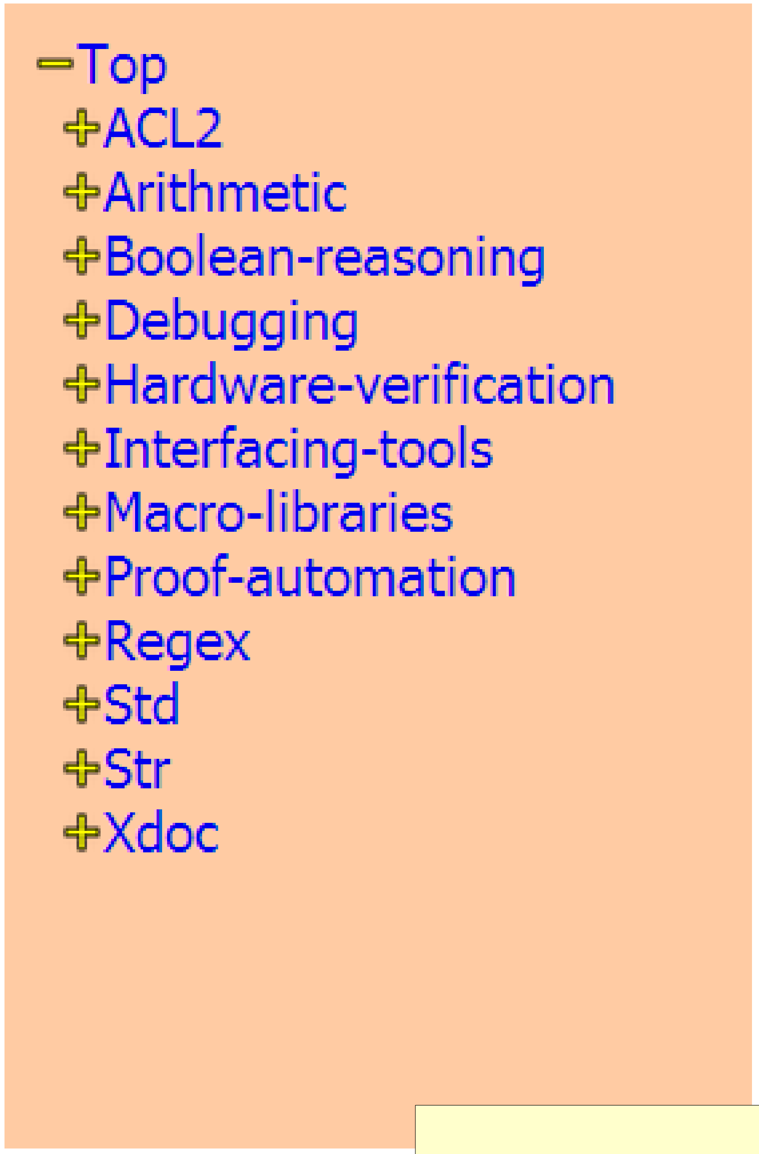
+4v
+ACL2
+Aig
+B*
+Bitops
+Bridge
+Clex
+Cutil
+Data-definitions
+Data-structures
+Esim
+Faig
+Gl
+Hacker
+Ihs
+Misc
+Osets
+Oslib
+Regex
+Std
+Str
+Testing
+Ubdds
+Undocumented
+Vl
· With-timeout
+Witness-cp
+Xdoc

```
(defmacro xdoc::fix-the-hierarchy ()
  `(progn
     (xdoc::change-parents ihs (arithmetic))

     (xdoc::change-parents b* (macro-libraries))
     (xdoc::change-parents data-definitions (macro-libraries))
     (xdoc::change-parents data-structures (macro-libraries))

     (xdoc::change-parents io (interfacing-tools))
     (xdoc::change-parents hacker (interfacing-tools))

     (xdoc::change-parents witness-cp (proof-automation))
     (xdoc::change-parents esim (hardware-verification))

     (xdoc::change-parents testing (debugging))

;; So I got started on that, and decided to move around a whole
;; bunch of ACL2 doc topics.  Much of this would probably make
;; more sense to do in ACL2 itself.

     (xdoc::change-parents copyright (about-acl2))
     (xdoc::change-parents version (about-acl2))
     (xdoc::change-parents release-notes (about-acl2))
     (xdoc::change-parents bibliography (about-acl2))
```



Full Index

+4v
+ACL2
+Aig
+B*
+Bitops
+Bridge
+Clex
+Cutil
+Data-definitions
+Data-structures
+Esim
+Faig
+Gl
+Hacker
+Ihs
+Misc
+Osets
+Oslib
+Regex
+Std
+Str
+Testing
+Ubdds
+Undocumented
+Vl
· With-timeout
+Witness-cp
+Xdoc



−Top
+ACL2
+Arithmetic
+Boolean-reasoning
+Debugging
+Hardware-verification
+Interfacing-tools
+Macro-libraries
+Proof-automation
+Regex
+Std
+Str
+Xdoc

Demo

Events

Add-custom-keyword-hint

Add a new custom keyword hint

Assert-event

Assert that a given form returns a non-`nil` value

Comp

Compile some ACL2 functions

Def-functional-instance

Functionally instantiate a pre-existing theorem to prove a new one.

Defabsstobj-missing-events

Obtain the events needed to admit a defabsstobj event

Defattach

Execute constrained functions using corresponding attached functions

Defaxiom

Add an axiom

Defchoose

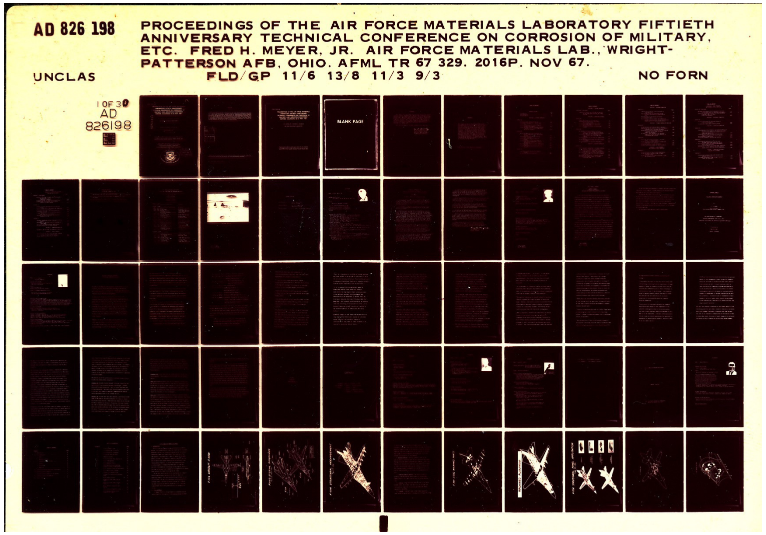
Define a Skolem (witnessing) function

Defcong

Prove congruence rule

Defconst

Define a constant


AD 826 198
PROCEEDINGS OF THE AIR FORCE MATERIALS LABORATORY FIFTIETH ANNIVERSARY TECHNICAL CONFERENCE ON CORROSION OF MILITARY, ETC. FRED H. MEYER, JR. AIR FORCE MATERIALS LAB., WRIGHT-PATTERSON AFB, OHIO. AFML TR 67 329. 2016P. NOV 67.
UNCLAS
FLD/GP 11/6 13/8 11/3 9/3
NO FORN
1 OF 30
AD
826198
BLANK PAGE

Switches-Parameters-and-Modes

Add-binop
Associate a function name with a macro name

Add-default-hints
Add to the default hints

Add-default-hints!
Add to the default hints non-locally

Add-dive-into-macro
Associate proof-checker diving function with macro name

Add-include-book-dir
Link keyword for `:dir` argument of ld and include-book

Add-invisible-fns
Make some unary functions invisible to the loop-stopper algorithm

Add-ld-keyword-alias
See ld-keyword-aliases.

Add-ld-keyword-alias!
See ld-keyword-aliases.

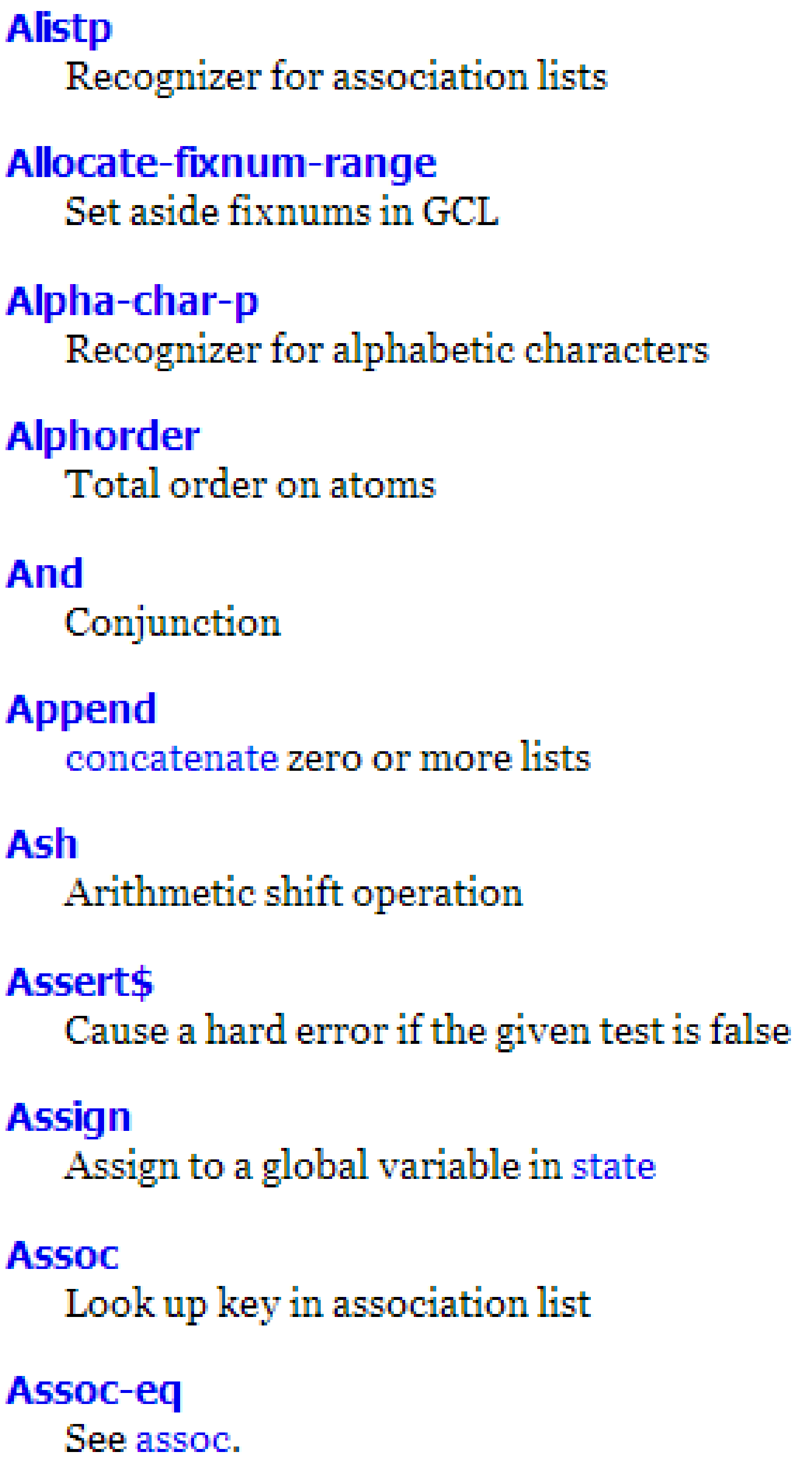
Add-macro-alias
Associate a function name with a macro name

Add-macro-fn
Associate a function name with a macro name

Add-match-free-override
Set `:match-free` value to `:once` or `:all` in existing rules

Add-nth-alias
Associate one symbol with another for printing of nth/update-nth terms

acl2-built-ins



Alistp
Recognizer for association lists

Allocate-fixnum-range
Set aside fixnums in GCL

Alpha-char-p
Recognizer for alphabetic characters

Alphorder
Total order on atoms

And
Conjunction

Append
concatenate zero or more lists

Ash
Arithmetic shift operation

Assert$
Cause a hard error if the given test is false

Assign
Assign to a global variable in state

Assoc
Look up key in association list

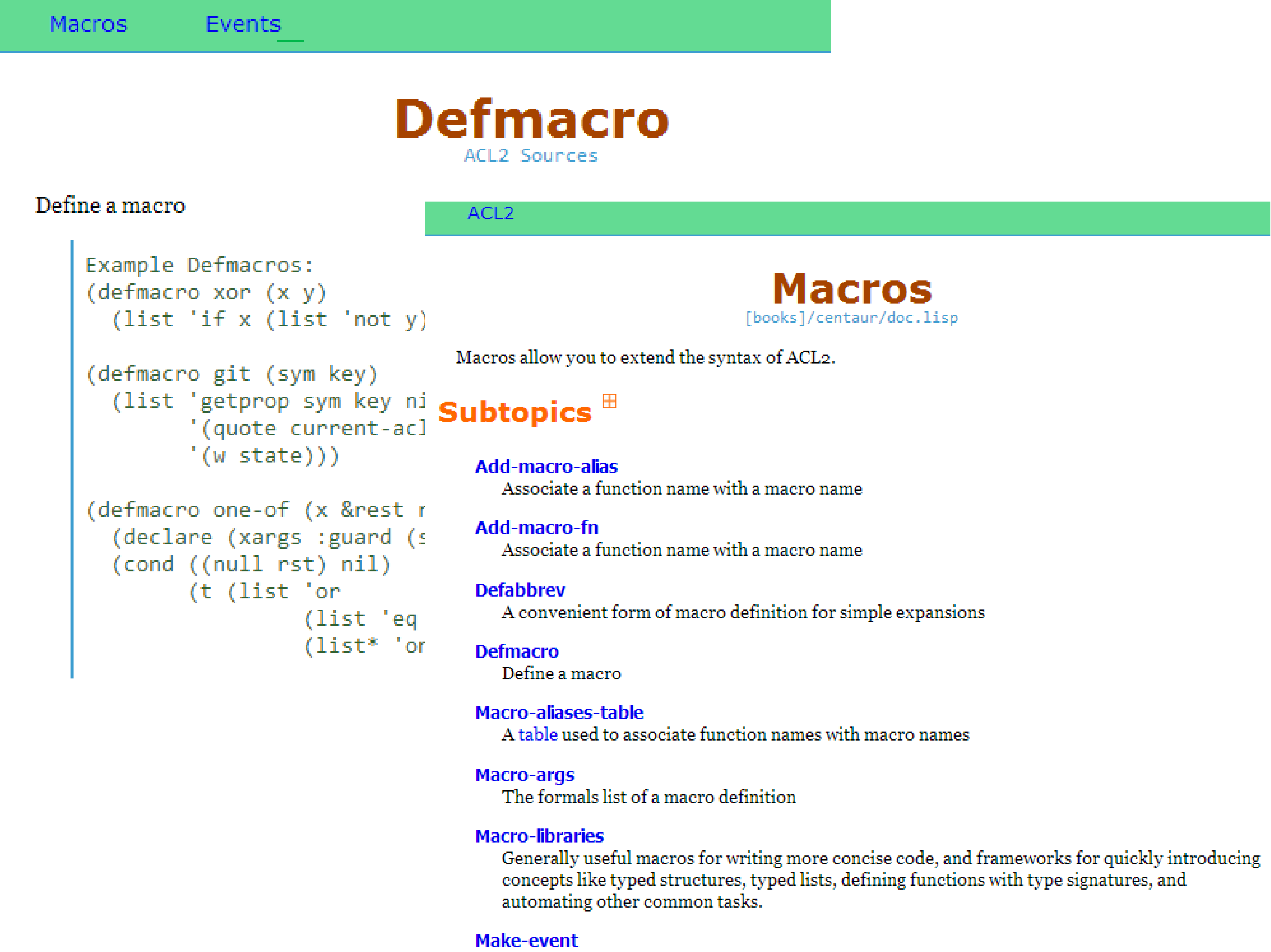
Assoc-eq
See assoc.

The New York Times.

ARMISTICE SIGNED, END OF THE WAR!
BERLIN SEIZED BY REVOLUTIONISTS;
NEW CHANCELLOR BEGS FOR ORDER;
OUSTED KAISER FLEES TO HOLLAND

A solution:

Multiple Parents



Defmacro

ACL2 Sources

Define a macro

```
Example Defmacros:
(defmacro xor (x y)
  (list 'if x (list 'not y)

(defmacro git (sym key)
  (list 'getprop sym key ni
        '(quote current-acl
        '(w state)))

(defmacro one-of (x &rest r
  (declare (xargs :guard (s
  (cond ((null rst) nil)
        (t (list 'or
                 (list 'eq
                 (list* 'or
```

Macros

[books]/centaur/doc.lisp

Macros allow you to extend the syntax of ACL2.

Subtopics ⊞

Add-macro-alias
Associate a function name with a macro name

Add-macro-fn
Associate a function name with a macro name

Defabbrev
A convenient form of macro definition for simple expansions

Defmacro
Define a macro

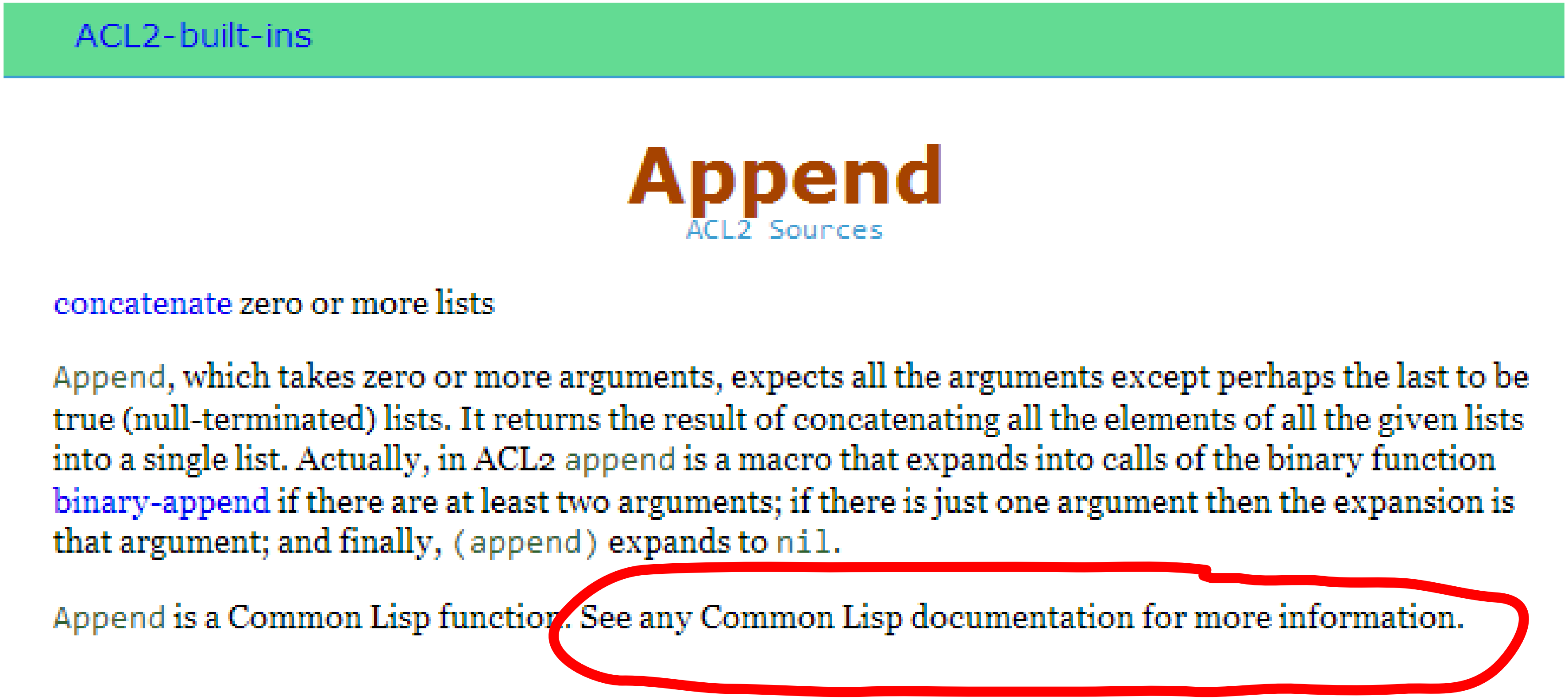
Macro-aliases-table
A table used to associate function names with macro names

Macro-args
The formals list of a macro definition

Macro-libraries
Generally useful macros for writing more concise code, and frameworks for quickly introducing concepts like typed structures, typed lists, defining functions with type signatures, and automating other common tasks.

Make-event

3. We should really link to external resources.



ACL2-built-ins

Append

ACL2 Sources

concatenate zero or more lists

Append, which takes zero or more arguments, expects all the arguments except perhaps the last to be true (null-terminated) lists. It returns the result of concatenating all the elements of all the given lists into a single list. Actually, in ACL2 append is a macro that expands into calls of the binary function binary-append if there are at least two arguments; if there is just one argument then the expansion is that argument; and finally, `(append)` expands to `nil`.

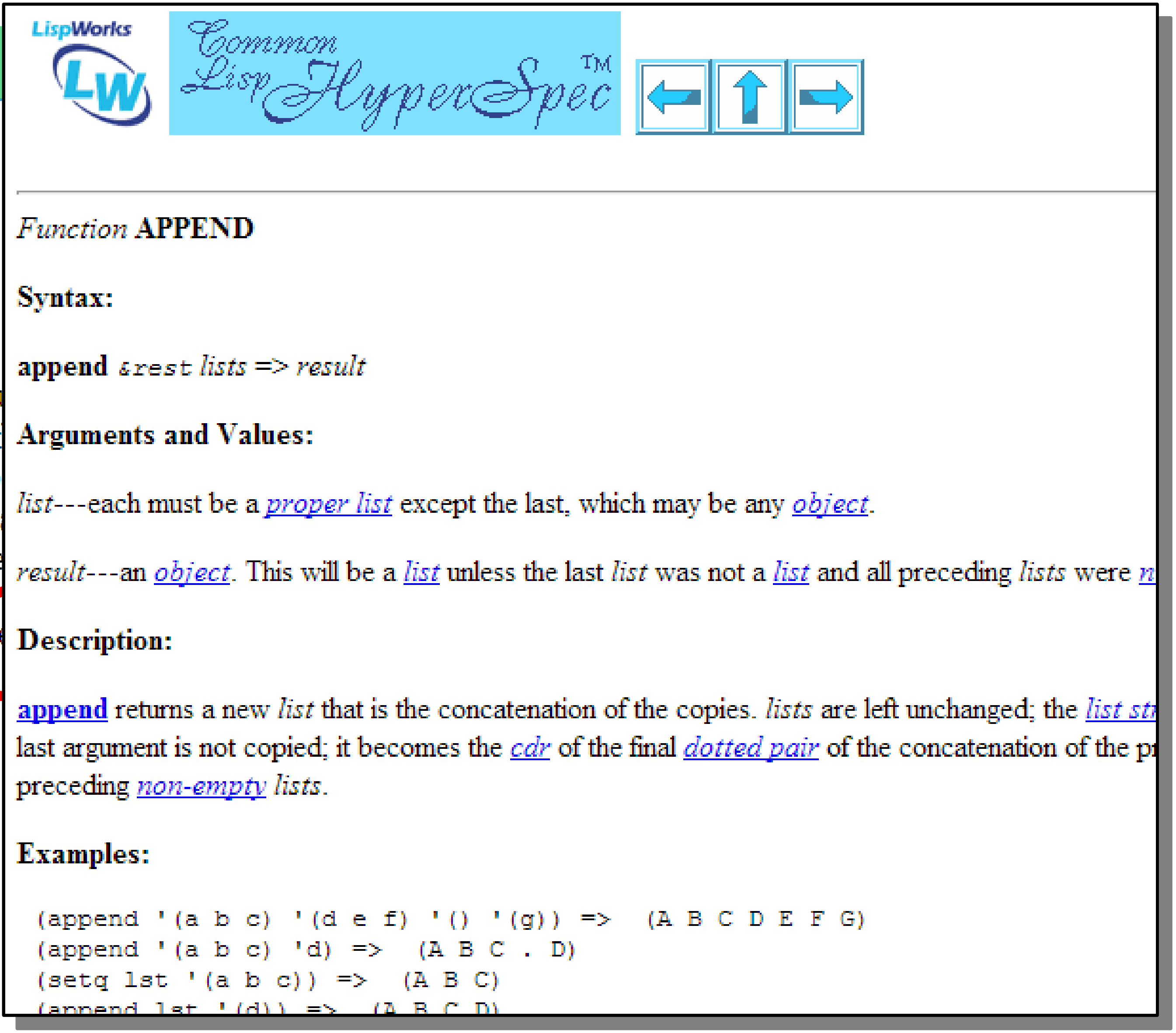
Append is a Common Lisp function. See any Common Lisp documentation for more information.

ACL2-built-ins

concatenate zero or more lists

Append, which takes zero or more argu
true (null-terminated) lists. It returns t
into a single list. Actually, in ACL2 app
binary-append if there are at least two
that argument; and finally, (append) e

Append is a Common Lisp function. See



Function **APPEND**

Syntax:

append `&rest` *lists* => *result*

Arguments and Values:

list---each must be a *proper list* except the last, which may be any *object*.

result---an *object*. This will be a *list* unless the last *list* was not a *list* and all preceding *lists* were *n*

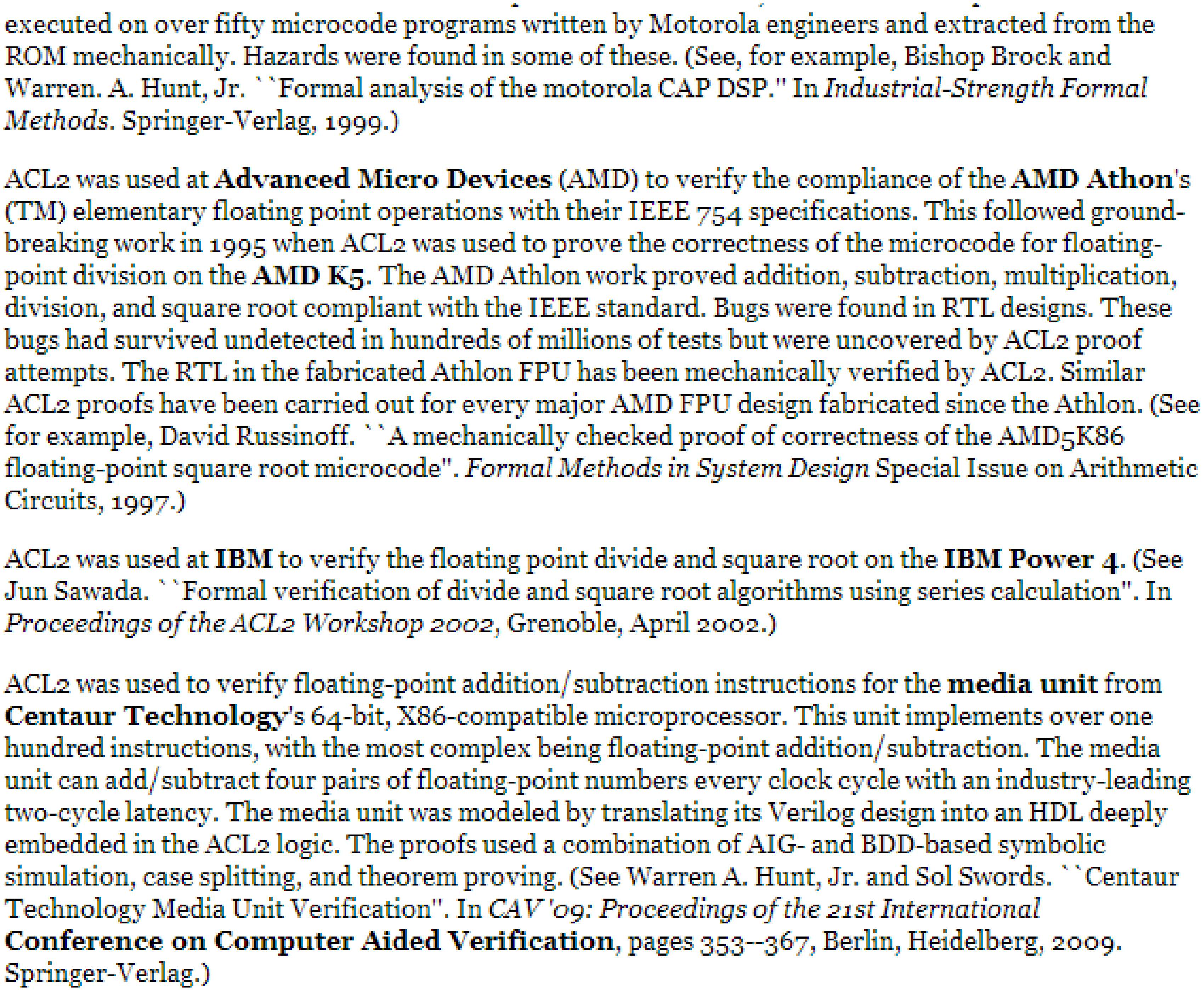
Description:

append returns a new *list* that is the concatenation of the copies. *lists* are left unchanged; the *list st*
last argument is not copied; it becomes the *cdr* of the final *dotted pair* of the concatenation of the pr
preceding *non-empty lists*.

Examples:

```
 (append '(a b c) '(d e f) '() '(g)) =>  (A B C D E F G)
 (append '(a b c) 'd) =>  (A B C . D)
 (setq lst '(a b c)) =>  (A B C)
 (append lst '(d)) =>  (A B C D)
```

interesting-applications



executed on over fifty microcode programs written by Motorola engineers and extracted from the ROM mechanically. Hazards were found in some of these. (See, for example, Bishop Brock and Warren. A. Hunt, Jr. ``Formal analysis of the motorola CAP DSP." In *Industrial-Strength Formal Methods*. Springer-Verlag, 1999.)

ACL2 was used at **Advanced Micro Devices** (AMD) to verify the compliance of the **AMD Athon**'s (TM) elementary floating point operations with their IEEE 754 specifications. This followed ground-breaking work in 1995 when ACL2 was used to prove the correctness of the microcode for floating-point division on the **AMD K5**. The AMD Athlon work proved addition, subtraction, multiplication, division, and square root compliant with the IEEE standard. Bugs were found in RTL designs. These bugs had survived undetected in hundreds of millions of tests but were uncovered by ACL2 proof attempts. The RTL in the fabricated Athlon FPU has been mechanically verified by ACL2. Similar ACL2 proofs have been carried out for every major AMD FPU design fabricated since the Athlon. (See for example, David Russinoff. ``A mechanically checked proof of correctness of the AMD5K86 floating-point square root microcode". *Formal Methods in System Design* Special Issue on Arithmetic Circuits, 1997.)

ACL2 was used at **IBM** to verify the floating point divide and square root on the **IBM Power 4**. (See Jun Sawada. ``Formal verification of divide and square root algorithms using series calculation". In *Proceedings of the ACL2 Workshop 2002*, Grenoble, April 2002.)

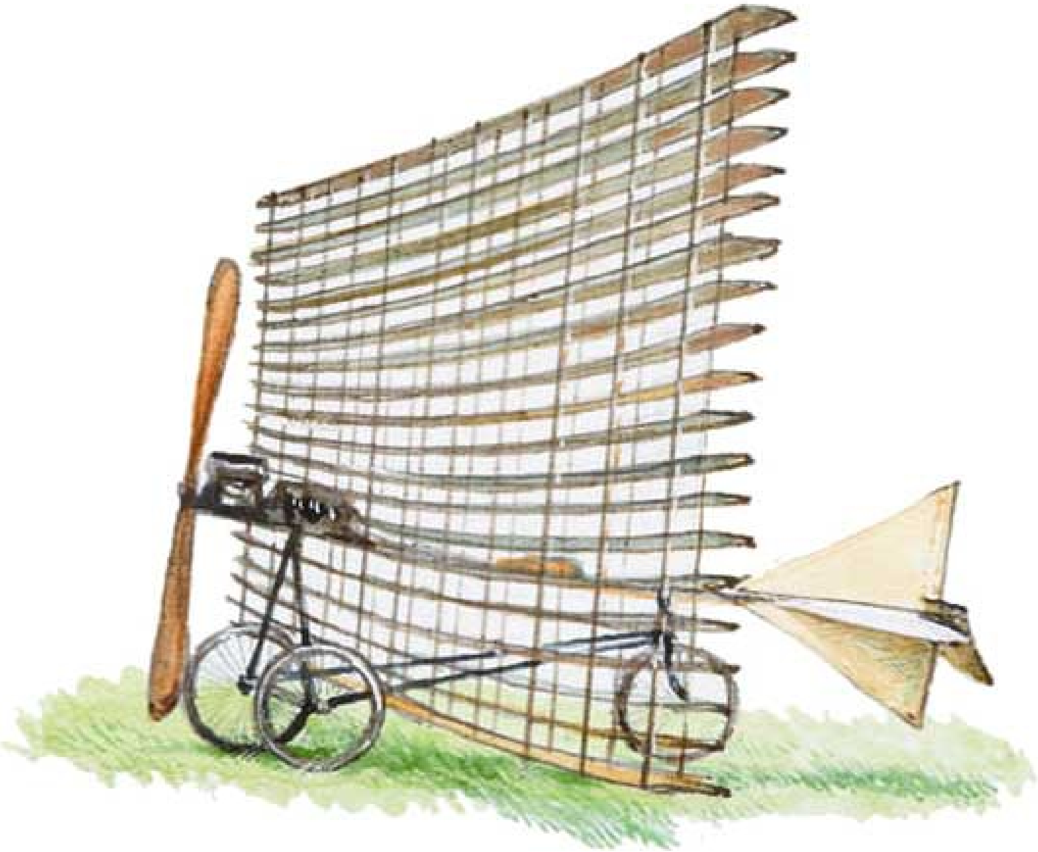
ACL2 was used to verify floating-point addition/subtraction instructions for the **media unit** from **Centaur Technology**'s 64-bit, X86-compatible microprocessor. This unit implements over one hundred instructions, with the most complex being floating-point addition/subtraction. The media unit can add/subtract four pairs of floating-point numbers every clock cycle with an industry-leading two-cycle latency. The media unit was modeled by translating its Verilog design into an HDL deeply embedded in the ACL2 logic. The proofs used a combination of AIG- and BDD-based symbolic simulation, case splitting, and theorem proving. (See Warren A. Hunt, Jr. and Sol Swords. ``Centaur Technology Media Unit Verification". In *CAV '09: Proceedings of the 21st International* **Conference on Computer Aided Verification**, pages 353--367, Berlin, Heidelberg, 2009. Springer-Verlag.)

A C L 2
PPLICATIVE COMMON LISP
Aided Reasoning
Studies
edited
Matt Kau
Moore
J Strother Moore
Comp
Scal
Techn
for Fo
Verifi
Proc
Applications
ACL2
WORKSHOP 2
Ruben Gamboa
Jared Davis
Interact
Theorem Pro
FLoC 2010
Computer Aided
Verification
Robert B. Jones
FMCAD
ACL2
Xdoc
???

We should convert ACL2's doc into xdoc and make it editable by the community.

Just need to spellcheck...

Thanks!



	XDOC	:DOC
Built into ACL2	no	yes
Docs in Latex	no	yes
Docs in Texinfo	no	yes
Docs in Terminal	yes	yes
Docs in Browser	yes+	yes
Standard markup	yes	no
DRY code insertion	yes	no
Do what you want	yes	no
Packages work	yes	no?
Custom manuals	yes	no?