ACL2 Workshop 2007

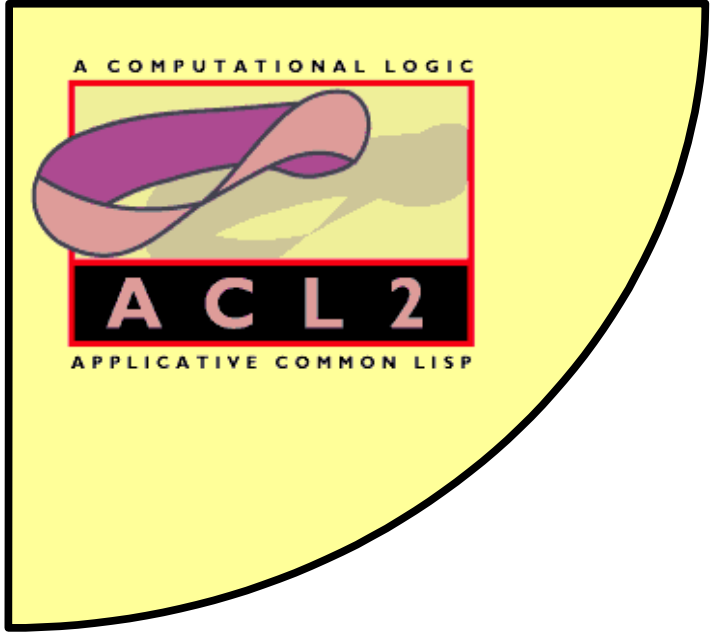
Hacking and Extending ACL2

Peter C. Dillinger

Panagiotis (Pete) Manolios

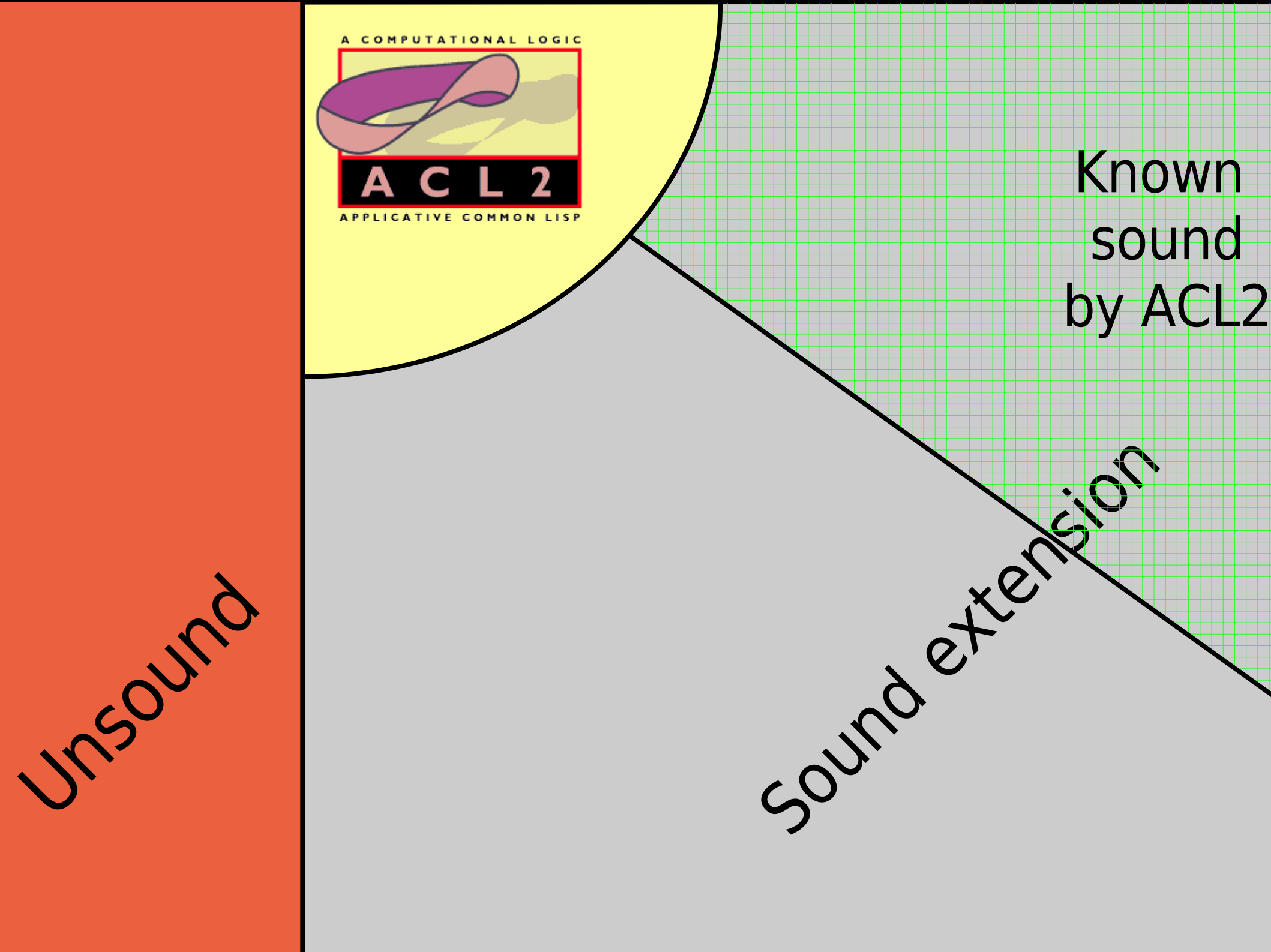
Matt Kaufmann

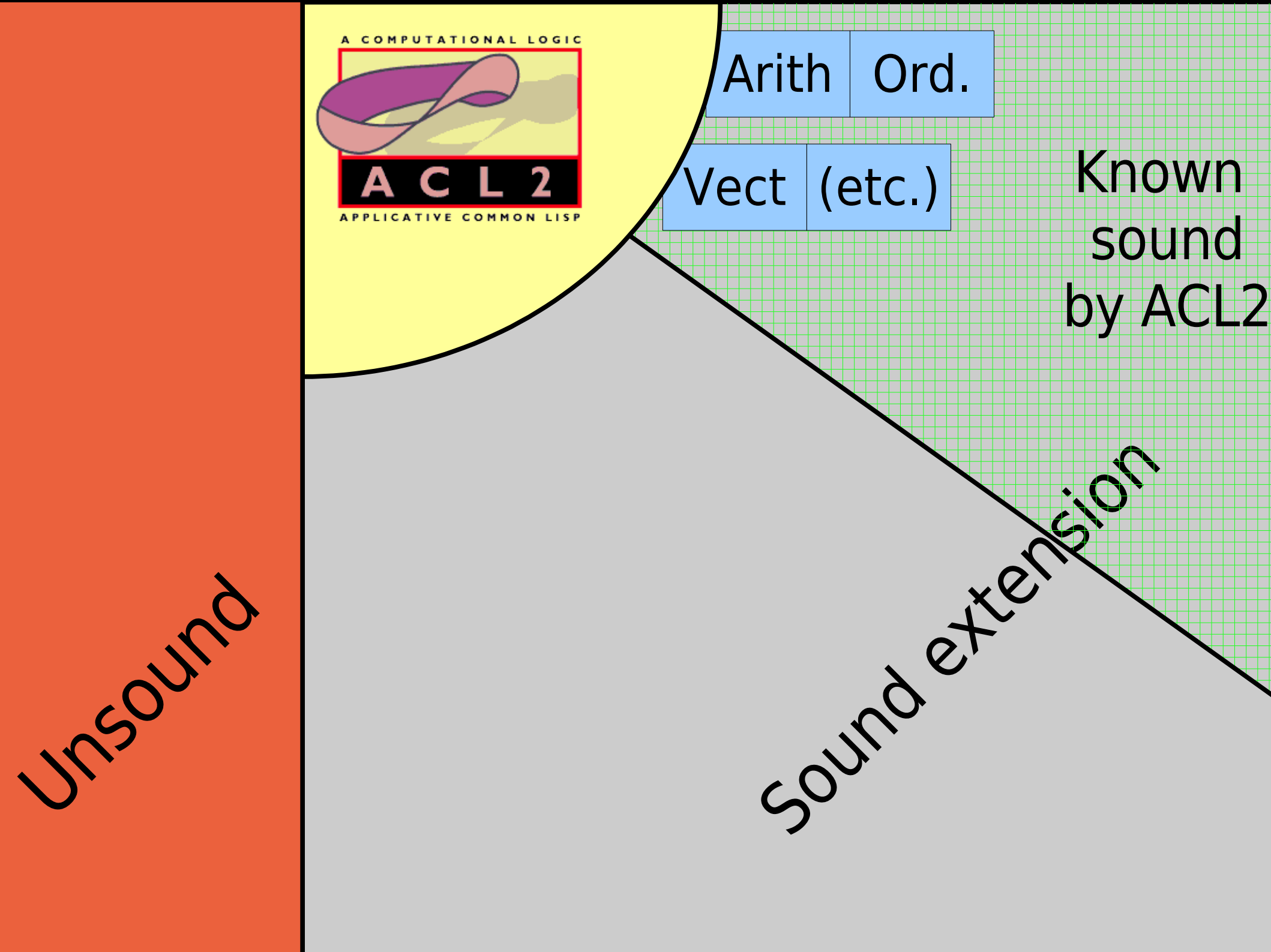
DANGER!

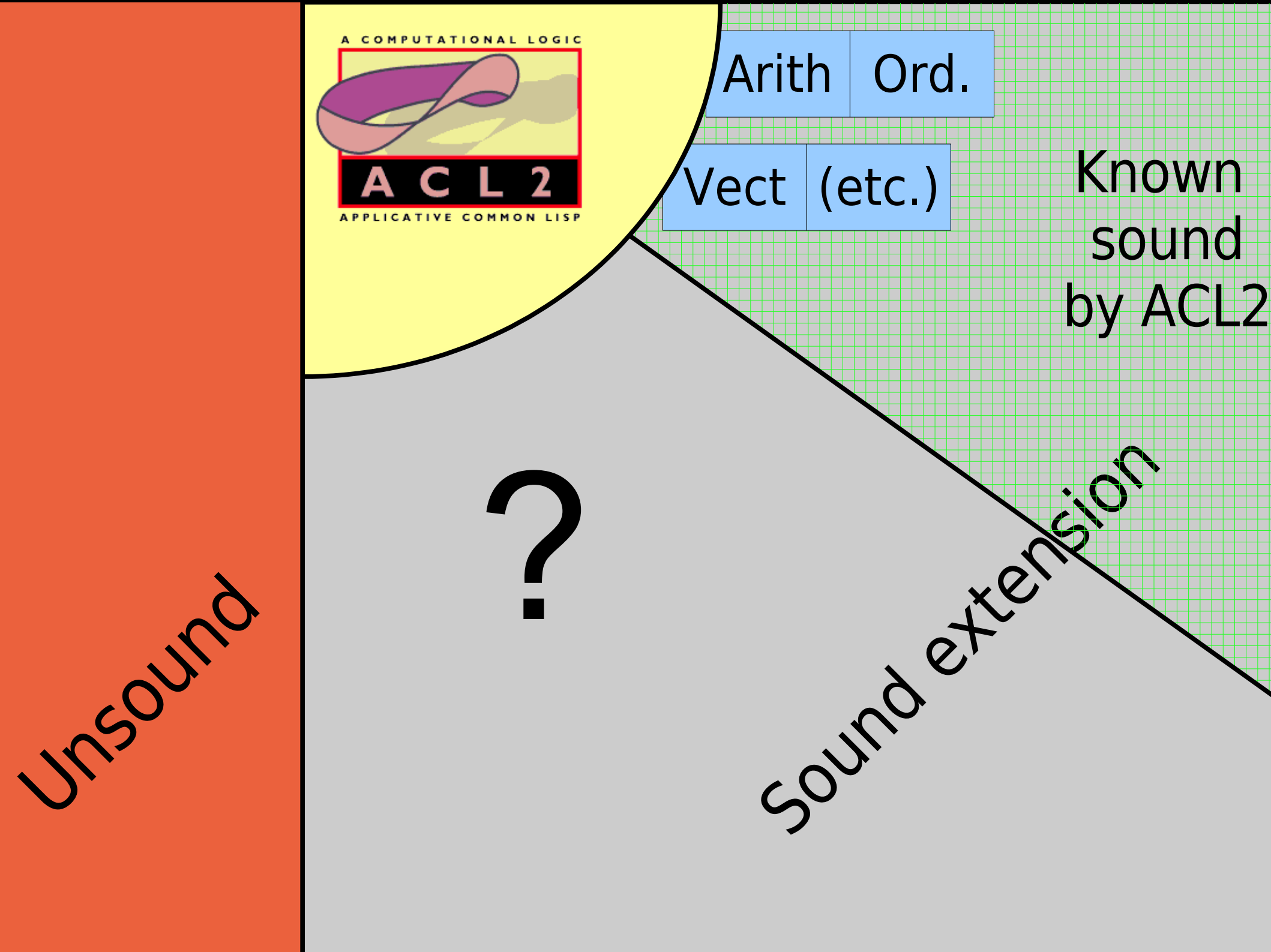

A COMPUTATIONAL LOGIC
A C L 2
APPLICATIVE COMMON LISP

Unsound

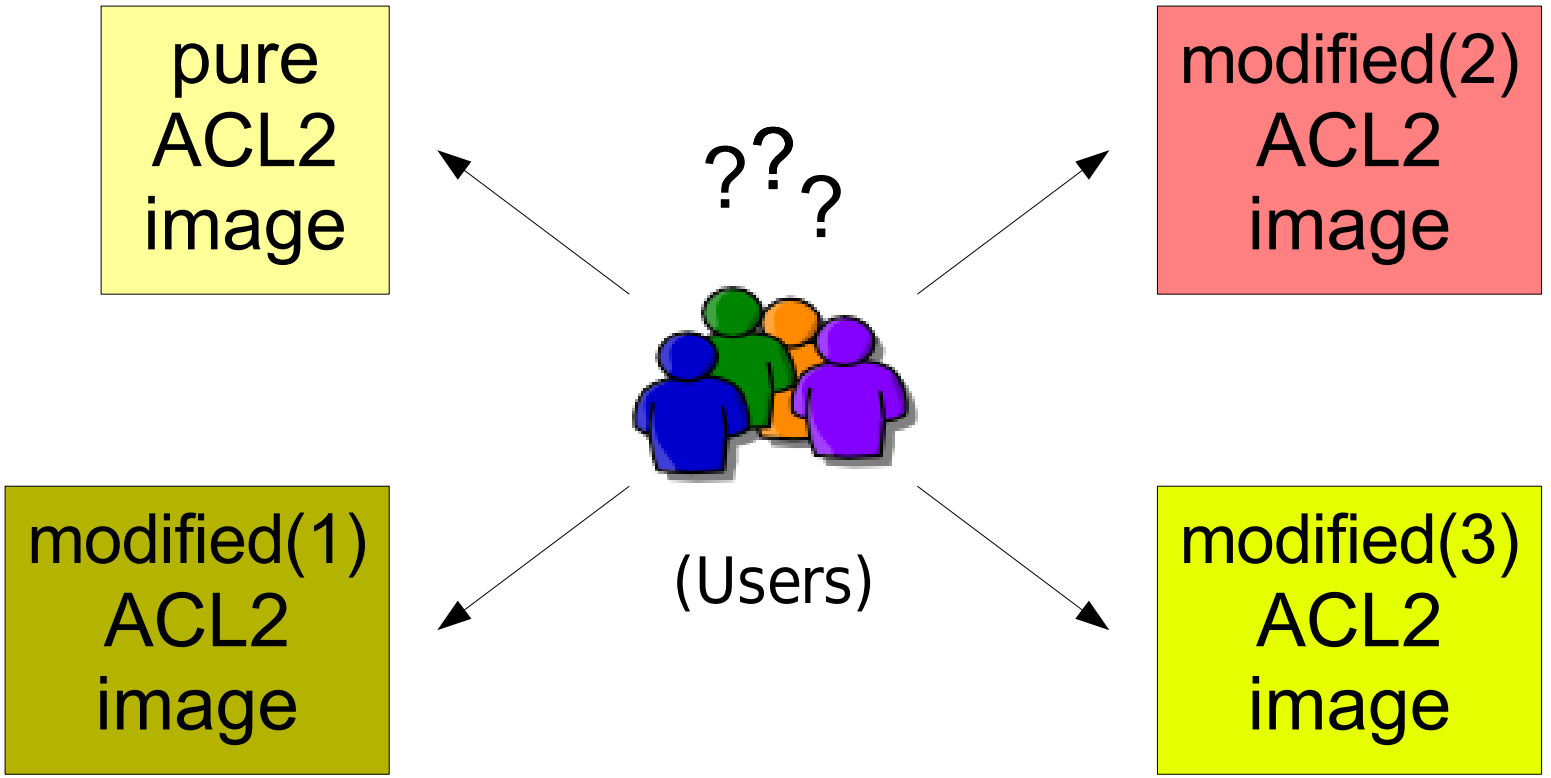
Sound extension


A COMPUTATIONAL LOGIC
ACL2
APPLICATIVE COMMON LISP
Known sound by ACL2
Sound extension
Unsound

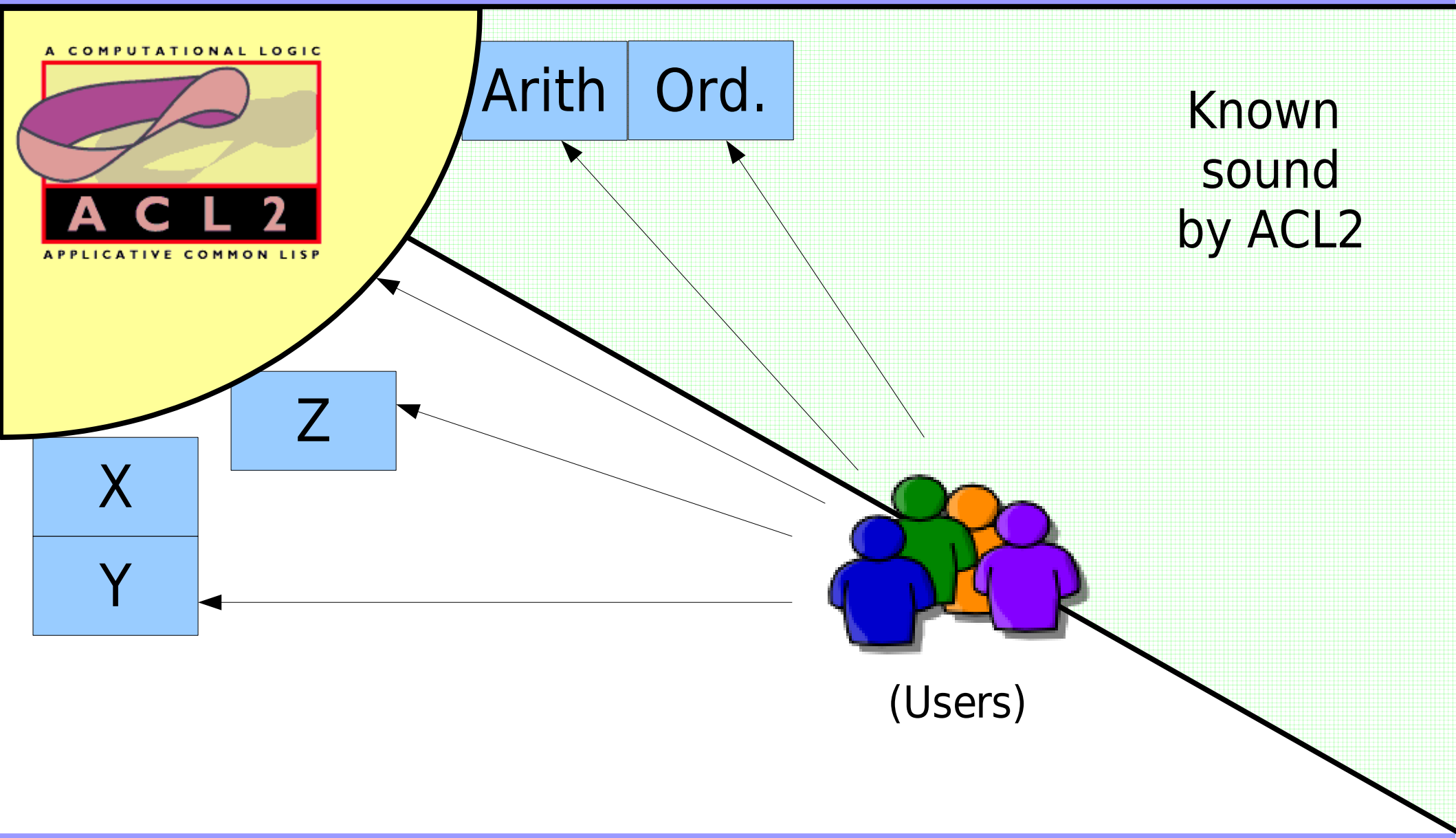

A COMPUTATIONAL LOGIC
A C L 2
APPLICATIVE COMMON LISP
Arith
Ord.
Vect
(etc.)
Known sound by ACL2
Sound extension
Unsound


A COMPUTATIONAL LOGIC
ACL2
APPLICATIVE COMMON LISP
Arith
Ord.
Vect
(etc.)
Known sound by ACL2
?
Unsound
Sound extension

Old Solution



Goal


A COMPUTATIONAL LOGIC
A C L 2
APPLICATIVE COMMON LISP
Arith
Ord.
Known sound by ACL2
Z
X
Y
(Users)

Goal

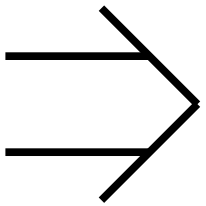
Facilitate ACL2 extensions of unproven soundness,

- Only when authorized
- In a trackable way
- That behave like books
- Easy to get right

Goal

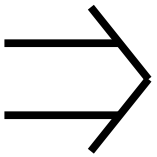
Facilitate ACL2 extensions of unproven soundness,

- Only when authorized
- In a trackable way



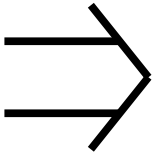
Trust tags

- That behave like books



`DEFCODE`

- Easy to get right

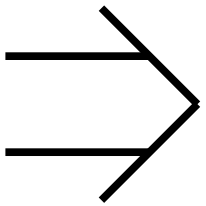


Elaborate extension

Outline

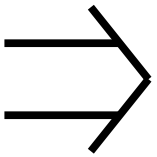
➔ Motivating examples/cases

- Only when authorized
- In a trackable way



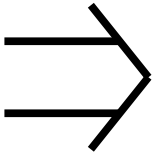
Trust tags

- That behave like books



`DEFCODE`

- Easy to get right

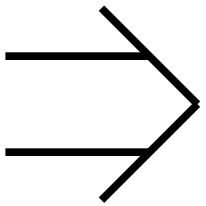


Elaborate extension

Outline

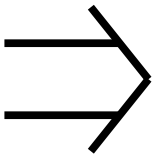
➔ Motivating examples/cases

- Only when authorized
- In a trackable way



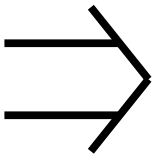
Trust tags

- That behave like books



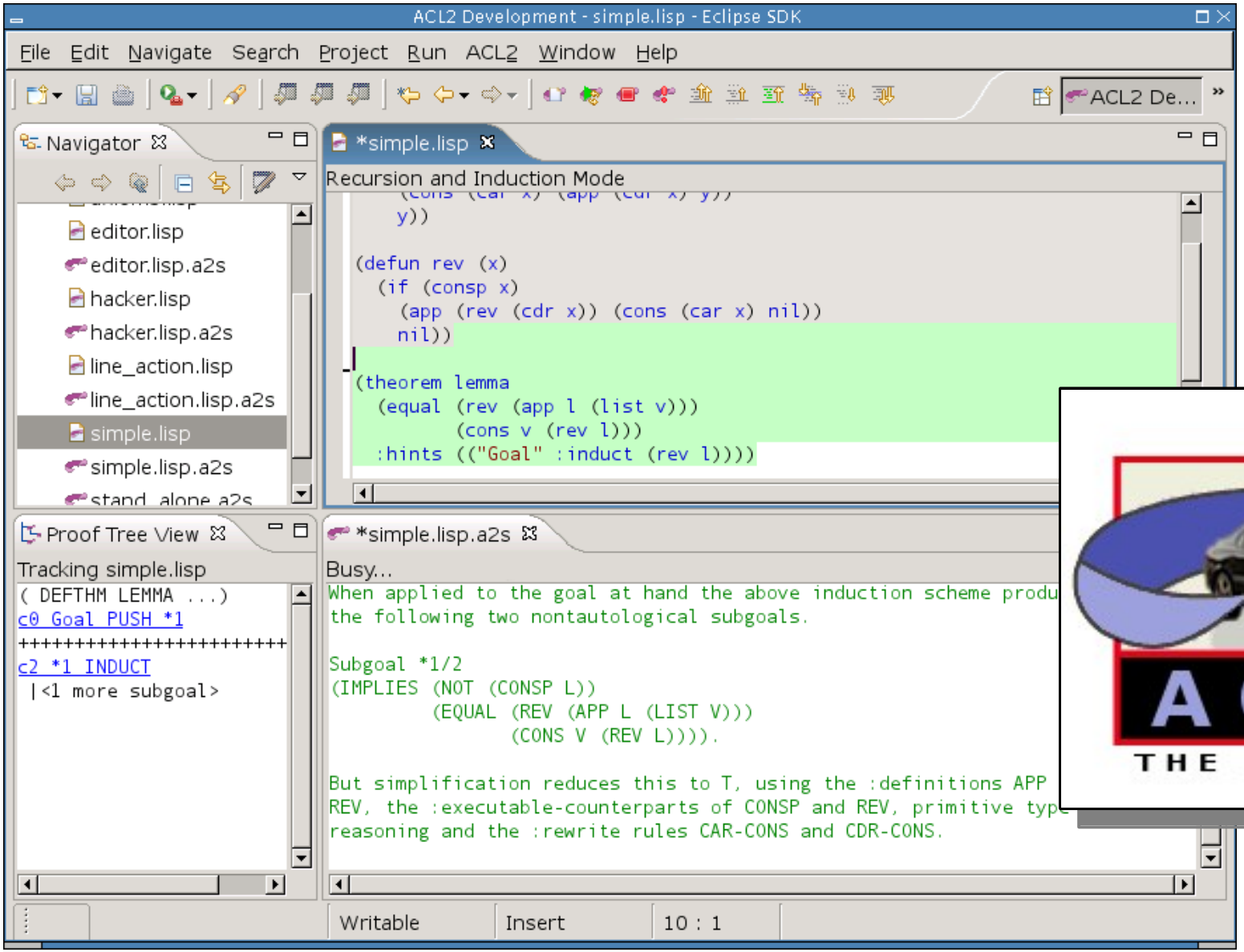
`DEFCODE`

- Easy to get right



Elaborate extension

Case 1: ACL2 Sedan



ACL2 Core Changes for ACL2s

- User Interface support:
  - Metadata output
  - Specialized UNDO/REDO mechanism
- Pedagogy support:
  - e.g. no autoinduct (Moore)
- Experimental improvements:
  - e.g. CCG Termination analysis (Vroon & Manolios)

Case 2: External reasoning

- E.g. external decision procedures
  - e.g. SULFA (Reeber & Hunt)
  - e.g. ACL2-SMT (Srinivasan & Manolios)
- See Kaufmann/Moore/Ray/Reeber paper

  “Integrating external deduction tools...”
- (Uses trust tags & “clause processors”)

Case 3: make-event problems

- “Macros with state”
- Introduced in Version 3.0
- Arbitrary code in books
- Save & restore the world (& other things)

  → *should* be safe

make-event problems

```
(defconst *nil.lisp*
          '((in-package "ACL2")
            (defthm bad nil :rule-classes nil)))

(defconst *nil.cert*
          '(<evil .cert file contents>))

(make-event
  (er-progn (write-to-file *nil.lisp* "nil.lisp")
            (write-to-file *nil.cert* "nil.cert")
            (value '(value-triple :invisible))))

(local (include-book "nil" :load-compiled-file nil))

(defthm bad nil :rule-classes nil)
```

Potential evil inside make-event

- Output to files
- `sys-call`
  - Bob Boyer's gdb example
- `trace$` with custom code

Rejected solution: <whatever>-okp flags

(Now these require a trust tag)

Potential evil inside make-event

- Ou
- sy
  - –
- tr

Reje

Key Insight:

Which dangerous construct

doesn't matter as much as

How it's used

and

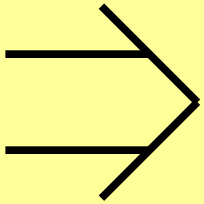
On behalf of whom

(Now these require a trust tag)

Outline

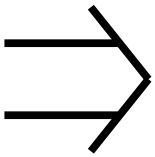
➔ Motivating examples/cases

- Only when authorized
- In a trackable way



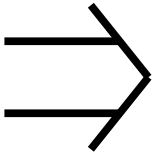
Trust tags

- That behave like books



`DEFCODE`

- Easy to get right



Elaborate extension

basic example (broken)

```
ACL2 !>(redef)
 (:QUERY . :OVERWRITE)
ACL2 !>(defun prove (term pspv hints ens wrld ctx state) (value nil))


ACL2 Error in ( DEFUN PROVE ...):  Redefinition of system functions
is not permitted unless there is an active trust tag (ttag).  See :DOC
defttag.


Summary
Form:  ( DEFUN PROVE ...)
Rules: NIL
Warnings:  None
Time:  0.00 seconds (prove: 0.00, print: 0.00, other: 0.00)

******** FAILED ********  See :DOC failure  ******** FAILED ********
ACL2 !>
```

basic example (fixed w/ttag)

```
********  FAILED  ********   See :DOC failure   ********  FAILED  ********
ACL2 !>(defttag <non-nil symbol>)

TTAG NOTE: Adding ttag <non-nil symbol> from the top level loop.
 T
ACL2 !>
```

basic example (fixed w/ttag)

```
********  FAILED  ********   See :DOC failure   ********  FAILED  ********
ACL2 !>(defttag t) ;; Let me do what I want.

TTAG NOTE: Adding ttag T from the top level loop.
 T
ACL2 !>(defun prove (term pspv hints ens wrld ctx state) (value nil))

ACL2 Query (:REDEF):  PROVE is an ACL2 system function.  Its current
defun-mode is :program. Do you really want to redefine it?  (N, Y,
E, O or ?):  y
...

Summary
Form:  ( DEFUN PROVE ...)
Rules: ((:FAKE-RUNE-FOR-TYPE-SET NIL))
Warnings:  None
Time:  0.02 seconds (prove: 0.00, print: 0.00, other: 0.02)
 PROVE
ACL2 !>(defthm bad nil :rule-classes nil)...
```

ttags with books example

abc.lisp

```
(in-package "ACL2")
...
(defttag :foo)
<use of dangerous constructs>
(defttag nil)
...
```

abc.acl2

```
(certify-book "abc" ? t
  :ttags ((:foo)))
```

ttags with books example

abc.lisp

```
(in-package "ACL2")
...
(defttag :foo)
<use of dangerous constructs>
(defttag nil)
...
```

abc.acl2

```
(certify-book "abc" ? t
  :ttags ((:foo)))
```

xyz.lisp

```
...
(include-book "abc"
  :ttags ((:foo)))
...
```

xyz.acl2

```
(certify-book "xyz"
  :ttags ((:foo)))
```

ttags with books failure (1)

```
ACL2 !>(include-book "all")


ACL2 Error in ( INCLUDE-BOOK "all" ...):  The ttag DEFCODE associated
with file /home/peterd/eclipse-workspace/acl2s/hooks/hacker/all.lisp
is not among the set of ttags permitted in the current context, namely:
  NIL.
See :DOC defttag.


Summary
Form:  ( INCLUDE-BOOK "all" ...)
Rules: NIL
Warnings:  None
Time:  0.68 seconds (prove: 0.00, print: 0.00, other: 0.68)

******** FAILED ********  See :DOC failure  ******** FAILED ********
ACL2 !>
```

ttags with books failure (2)

```
ACL2 !>(include-book "all" :ttags ((defcode)))


ACL2 Error in ( INCLUDE-BOOK "all" ...):  The ttag TABLE-GUARD associated
with file /home/peterd/eclipse-workspace/acl2s/hooks/hacker/all.lisp
is not among the set of ttags permitted in the current context, namely:
  ((DEFCODE)).
See :DOC defttag.


Summary
Form:  ( INCLUDE-BOOK "all" ...)
Rules: NIL
Warnings:  None
Time:  0.15 seconds (prove: 0.00, print: 0.00, other: 0.15)

******** FAILED ********  See :DOC failure  ******** FAILED ********
ACL2 !>
```

success!

```
ACL2 !>(include-book "all" :ttags ((defcode) (table-guard)))

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/defcode.lisp.

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/defcode-macro.lisp.

TTAG NOTE (for included book): Adding ttag TABLE-GUARD from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/table-guard.lisp.

Summary
Form:  ( INCLUDE-BOOK "all" ...)
Rules: NIL
Warnings:  None
Time:  0.96 seconds (prove: 0.00, print: 0.00, other: 0.96)
 "/home/peterd/eclipse-workspace/acl2s/hooks/hacker/all.lisp"
ACL2 !>
```

success! (alternate)

```
ACL2 !>(include-book "all" :ttags :all)) ; Just do it!

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/defcode.lisp.

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/defcode-macro.lisp.

TTAG NOTE (for included book): Adding ttag TABLE-GUARD from file
/home/peterd/eclipse-workspace/acl2s/hooks/hacker/table-guard.lisp.

Summary
Form:  ( INCLUDE-BOOK "all" ...)
Rules: NIL
Warnings:  None
Time:  0.96 seconds (prove: 0.00, print: 0.00, other: 0.96)
 "/home/peterd/eclipse-workspace/acl2s/hooks/hacker/all.lisp"
ACL2 !>
```

“TTAG NOTE” justification (1/2)

```
ACL2 !>(ttags-seen)
<no ttags seen>
Warning: This output is minimally trustworthy (see :DOC TTAGS-SEEN).
ACL2 !>(include-book "defcode" :ttags :all)

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/acl2-hacking/code/defcode.lisp.

TTAG NOTE (for included book): Adding ttag DEFCODE from file
/home/peterd/acl2-hacking/code/defcode-macro.lisp.

 "/home/peterd/acl2-hacking/code/defcode.lisp"
ACL2 !>(ttags-seen)
(DEFCODE "/home/peterd/acl2-hacking/code/defcode-macro.lisp"
         "/home/peterd/acl2-hacking/code/defcode.lisp")
Warning: This output is minimally trustworthy (see :DOC TTAGS-SEEN).
ACL2 !>
```

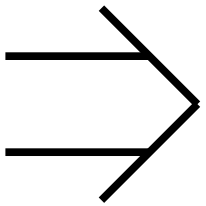
“TTAG NOTE” justification (2/2)

```
 "/home/peterd/acl2-hacking/code/defcode.lisp"
ACL2 !>(ttags-seen)
(DEFCODE "/home/peterd/acl2-hacking/code/defcode-macro.lisp"
         "/home/peterd/acl2-hacking/code/defcode.lisp")
Warning: This output is minimally trustworthy (see :DOC TTAGS-SEEN).
ACL2 !>(include-book "mischief" :ttags :all)

TTAG NOTE (for included book): Adding ttag MISCHIEF from file
/home/peterd/acl2-hacking/code/mischief.lisp.

 "/home/peterd/acl2-hacking/code/mischief.lisp"
ACL2 !>(ttags-seen)
<no ttags seen>
Warning: This output is minimally trustworthy (see :DOC TTAGS-SEEN).
ACL2 !>
```

Outline

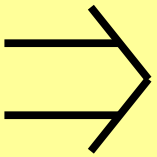
➔ Motivating examples/cases

- Only when authorized
- In a trackable way



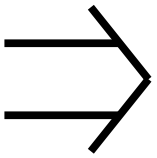
Trust tags

- That behave like books



`DEFCODE`

- Easy to get right



Elaborate extension

DANGER!

These techniques should be considered a *last resort* by most users.

Basic Extension Mechanisms

- Unprotected code
- Raw Lisp code
- Untouchable fns & state global vars
- Redefine system fns
- External invocation
- File writing

- PROGN!
- SET-RAW-MODE
- SET-TEMP-TOUCHABLE-{FNS,VARS}
- (complicated)
- SYS-CALL
- OPEN-OUTPUT-CHANNEL!

PROGN! shortcomings

- Define a function in raw Lisp (in a book):

```
(progn!
   (in-raw-mode
      (defun f (x) (g x))))
```

- What about undoing?
- What about resurrection (`oops`)?
  - Cheap “solution”: `reset-prehistory`

DEFCODE: separate meanings

• What happens when it is processed by the ACL2 loop?	:LOOP
• What happens when any world extension it creates is installed?	:EXTEND
• What happens when any world extension it creates is retracted?	:RETRACT
• What does it look like to the Lisp compiler?	:COMPILE

DEFCODE Example Outline

- Define a function in raw Lisp:

```
(defcode
  :loop      If already defined, throw an error.
  :compile   <raw definition for compiler>
  :extend    Install raw definition.
  :retract   Remove raw definition. )
```

DEFCODE Actual Example

```
(defcode
  :loop ((in-raw-mode
          (when (or (fboundp 'f)
                    (macro-function 'f))
            (hard-error 'defcode
                        "F already defined."))))
  :compile ((defun f (x) (g x)))
  :extend ((in-raw-mode (defun f (x) (g x))))
  :retract ((in-raw-mode (fmakunbound 'f))))
```

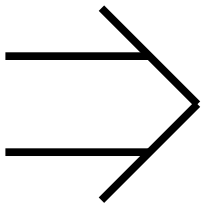
DEFCODE Possible Criticisms

- Verbose
  - Macros → not a problem!
- Not part of ACL2 proper
  - Hard to define DEFCODE without DEFCODE!
- Why (again)?
  - Enables books of extensions to behave like ordinary books (+ ttags)
- Low-level; hard to get right ...

Outline

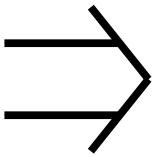
➔ Motivating examples/cases

- Only when authorized
- In a trackable way



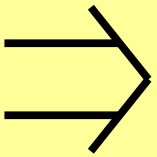
Trust tags

- That behave like books



`DEFCODE`

- Easy to get right



Elaborate extension

E.g. Remove Auto Induction

```
(redefun+rewrite
 induct
 (:carpat (select-x-cl-set %cl-set% %induct-hint%)
  :vars (%cl-set% %induct-hint%)
  :mult +
  :repl
  (select-x-cl-set (if (get-no-auto-induct state)
                       nil
                       %cl-set%)
                   %induct-hint%)))
```

Yes, it uses make-event. :)

E.g. Special Output

```
(modify-raw-defun
  read-object (chan st)
  :name-for-old-raw original-read-object
  :raw (progn (when (and (acl2::live-state-p st)
                         (eq chan *standard-oi*)
                         (acl2s-activep st))
                (format t "${ReAd-ObJeCt}$~%"))
              (original-read-object chan st)))
```

- Resembles AOP

Conclusion

Facilitate ACL2 extensions of unproven soundness,

- Only when authorized
- In a trackable way

⇒ Trust tags

- That behave like books

⇒ `DEFCODE`

- Easy to get right

⇒ Elaborate extension

Final Thoughts

- Trust tags for users
  - Know what they are & what they mean
  - Be picky
- Trust tags for hackers
  - Be nice
  - Be smart
- Should `DEFCODE` be in ACL2?
- Hacking is a last resort!

Thanks & Questions

See also

- Paper & accompanying code
- “Integrating External Deduction Tools with ACL2.” Kaufmann, Moore, Ray, and Reeber. *Journal of Applied Logic*.
- “ACL2s: The ACL2 Sedan.” Dillinger, Manolios, Vroon, and Moore. *Proc. UITP* 2006. ENTCS.
- http://acl2s.peterd.org

No ttags using a ttag

```
(in-package "ACL2")

(defttag no-ttags)

(progn!
 (set-raw-mode t)
 (defun notify-on-defttag (val active-book-name include-bookp state)
   (value (hard-error 'defttag "Trust tags have been disabled for
  this session." ()))))

(local
 (progn!
  (set-raw-mode t)
  (defun notify-on-defttag (val active-book-name include-bookp state)
    (value nil))))
```