

ACL2 for Freshmen: First Experiences

Carl Eastlund Dale Vaillancourt Matthias Felleisen
'(cce dalev matthias)@ccs.neu.edu

Northeastern University
Boston, MA

ACL2 Workshop 2007

Outline

- ① Background
- ② Conjecture
- ③ Experiment
- ④ Evaluation

Outline

① Background

② Conjecture

③ Experiment

④ Evaluation

Freshman Year

Fall Semester

Fundamentals I:

Functional programming
and the Design Recipe

Discrete Structures:

Discrete math, e.g. sets,
functions, and induction

Spring Semester

Fundamentals II:

Object-oriented
programming

Symbolic Logic:

Propositional and
predicate logic

The Six-Step Design Recipe

```
;; A LoN is either:  
;; - nil, or  
;; - (cons Number LoN)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?**
Use cond.
 - Compound data?**
Apply accessors.
 - Inductive data?**
Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:  
;; - nil, or  
;; - (cons Number LoN)  
  
;; sum : LoN -> Number  
;; Add all numbers in a list.
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
Use cond.
 - Compound data?
Apply accessors.
 - Inductive data?
Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
```

```
(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
 - Use cond.
 - Compound data?
 - Apply accessors.
 - Inductive data?
 - Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)

)

(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
 - Use cond.
 - Compound data?
 - Apply accessors.
 - Inductive data?
 - Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)
  (cond
    ((endp ns) 0)
    (t
     (+ (first ns)
        (sum (rest ns)))))

(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?**
Use cond.
 - Compound data?**
Apply accessors.
 - Inductive data?**
Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)
  (cond
    ((endp ns) 0)
    (t (cons (car ns)
              (sum (cdr ns)))))

(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
 - Use cond.
 - Compound data?
 - Apply accessors.
 - Inductive data?
 - Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)
  (cond
    ((endp ns) )
    (t (car ns)
        (sum (cdr ns)) )))

(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?**
Use cond.
 - Compound data?**
Apply accessors.
 - Inductive data?**
Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)
  (cond
    ((endp ns) 0)
    (t (+ (car ns)
          (sum (cdr ns))))))

(equal (sum nil) 0)
(equal (sum '(1 2)) 3)
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
 - Use cond.
 - Compound data?
 - Apply accessors.
 - Inductive data?
 - Recur.
- 5 Write Code
- 6 Run Tests

The Six-Step Design Recipe

```
;; A LoN is either:
;; - nil, or
;; - (cons Number LoN)

;; sum : LoN -> Number
;; Add all numbers in a list.
(defun sum (ns)
  (cond
    ((endp ns) 0)
    (t (+ (car ns)
          (sum (cdr ns))))))

(equal (sum nil) 0)      ; => t
(equal (sum '(1 2)) 3) ; => t
```

- 1 Data Definition
- 2 Contract & Purpose
- 3 Examples
- 4 Template
 - Multiple clauses?
 - Use cond.
 - Compound data?
 - Apply accessors.
 - Inductive data?
 - Recur.
- 5 Write Code
- 6 Run Tests

Student Languages

The screenshot shows the DrScheme IDE window titled "fact.ss - DrScheme". The editor contains the following Scheme code:

```
(define (fact n)
  (if (zero? n)
      1
      (* n (fact (sub1 n)))))

(define (g x)
  (let ((y (fact (- x))))
    (+ y x)))
```

The output window shows the following interaction:

```
Welcome to DrScheme, version 371-svn18aug2007 [3m].
Language: Intermediate Student.
> (fact 3)
6
> (g "3")
-: expects argument of type <number>; given "3"
>
```

The bottom status bar displays "Programming language: Intermediate Student", a clock showing "7:17", a "GC" button, a memory usage indicator showing "63.02", and a "Read/Write" button.

Teachpacks

worm-student.ss - DrScheme

Test Results

Recorded 6 checks. 1 check failed.
[check failed at line 320 column 3](#)

Actual value `(make-worm 'left (make-posn 0 10) empty)` differs from `(make-worm 'left (make-posn 10 10) empty)`

DrScheme

```

;; 'worm-change ;; this test case is wrong: worm-change also moves the worm
(check-expect (worm-change (make-worm 'up (make-posn 10 10) empty) 'left)
              (make-worm 'left (make-posn 10 10) empty))

```

Welcome to [DrScheme](#), version 371-svn18aug2007 [3m].
 Language: *Advanced Student custom*.
 Teachpacks: *world.ss* and *testing.ss*.
 true
 true
 true
 true
 true
 end of time: the worm ate itself (score: 23)
 >

Programming language: **Advanced Student cu...** GC 62.82 Read/Write

Dracula Language

The screenshot shows the DrScheme IDE window titled "fact.lisp - DrScheme". The editor contains the following code:

```
(defun fact (n)
  (if (zp n)
      1
      (* n (fact (1- n)))))

(defun g (x)
  (let ((y (fact (- x))))
    (+ y x)))
```

The error message in the console is:

```
Welcome to DrScheme, version 360.
Language: ACL2 Beginner (beta 11.2, 4/11/2007).
> (g 42)
 2:7: top-level broke the contract (-> natural-number/c any) on zp; expected
<natural-number/c>, given: -42
>
```

A red arrow points from the error message to the `zp` function call in the code. The error indicates that the `zp` function, which is expected to return a natural number, instead returned `-42` when called with `42`.

Dracula Proofs

The screenshot shows the DrScheme IDE with two windows. The main window, titled "worm.lisp - DrScheme", displays the following code:

```

;; When the worm moves, we drop the
;; This of course preserves that the
(defthm firstn-preserves-consecutive
  (implies (consecutive-segments-adj
            (consecutive-segments-adj
              (firstn n segs))))))

;; Finally, we prove an interesting
(defthm worm-move-preserves-well-for
  (implies (and (worm-p w) (worm-wel
                  (worm-well-formed? (worm-
:hints ("Goal" :in-theory (enable

```

The smaller window, titled "acl2.1184609263.txt - DrScheme", shows the proof's progress:

```

( DEFTHM WORM-MOVE-PRESERVES-WELL-FORMEDNESS ... )
Q.E.D.

(:TYPE-PRESCRIPTION CONSP-FIRSTN)
(:TYPE-PRESCRIPTION FIRSTN)
(:TYPE-PRESCRIPTION LEN)
(:TYPE-PRESCRIPTION LIST-OF-SEGMENTS?)
(:TYPE-PRESCRIPTION WORM-WELL-FORMED?)

Warnings: Non-rec

```

The IDE interface includes buttons for "Step", "Check Syntax", "Run", "Stop", "Start ACL2", "Admit Next", "Admit All", "Undo Last", "Save / Certify", "Reset ACL2", "Stop Prover", and "Previous Checkpoint / Next Checkpoint".

Dracula Teachpacks

The image shows a screenshot of the ACL2 environment with three windows:

- Test Results:** Shows 55 checks, with 3 failing.
 - Check failed at line 410 column 0: Actual value `(list 'worm 'right (list 'posn 10 0) empty)` differs from `(list 'worm 'right (list 'posn 0 0) empty)`, the expected value.
 - Check failed at line 444 column 0: Actual value `(list (list 'posn 0 0))` differs from `t`, the expected value.
 - Check failed at line 455 column 0: Actual value `()` differs from `t`, the expected value.
- worm.lisp - DrScheme:** Contains the following code:


```

worm.lisp
(defun ...)
;; worm-ate-self? : worm-p -> Boolean
;; Does the worm's head overlap with any segment in its body?
(defun worm-ate-self? (w)
  (let ((h (worm-head w)))
    (member-equal h (worm-body w)))) ;; instead of posn? ...

(check-expect (worm-ate-self?
  (worm 'up (make-posn 0 0)
    (list (make-posn 0 "DIAMETER")
          (make-posn "DIAMETER" "DIAMETER")
          (make-posn "DIAMETER" 0))
    nil)
  t)
(check-expect (worm-ate-self?
  (worm 'down (make-posn 0 0)
    (list (make-posn 0 "DIAMETER")
          (make-posn 0 0)))
  t)

;; worm-would-eat-self? : worm-p velocity? -> Boolean

t
t
t
end of time: the worm hit the wall (score: 20)
437:0
      
```
- DrScheme:** A graphical window showing a red worm shape on a white background. The worm is composed of red dots forming an L-shape.

Sample Logic Exercise

Prove the conclusion from the premises or provide an interpretation which establishes invalidity.

- 1 My shirt is under the bed. Your shirt is on the table. If your shirt is on the table, then it's not under the bed. Therefore, my shirt is not your shirt.
- 2 If Tom votes, he will vote Democratic unless the party reverses its position on gun-control. The party will not reverse its position on gun-control. So, either Tom doesn't vote or he will vote Democratic.
- 3 I will do well in this course and I will study the material. So, I will do well in this course if and only if I will study the material.

Outline

① Background

② **Conjecture**

③ Experiment

④ Evaluation

Remember the S.A.T.?

Logic : Computing :: Analysis : Physics

Preparing Freshmen for ACL2

```
(defun sum (ns)
  (cond
    ((endp ns) ...)
    (t ... (car ns)
      ... (sum (cdr ns)) ...))))
```

Multiple clauses?

Use cond.

Compound data?

Apply accessors.

Inductive data?

Recur.

The screenshot shows a DrScheme window titled "acl2.1195156907.txt - DrScheme". At the top, there are two buttons: "< Previous Checkpoint" and "Next Checkpoint >". Below these buttons, the code editor contains the following text:

```
( DEFTHM SUM-RATIONAL ... )
Q.E.D.
```

Below the code editor, there is a scrollable area containing the following text:

```
We will induct according to a scheme suggested by (SUM NS). This suggestion
was produced using the :induction rules RATIONAL-LISTP and SUM. If
we let (:P NS) denote *1 above then the induction scheme we'll use
is
(AND (IMPLIES (AND (NOT (ENDP NS)) (:P (CDR NS)))
                (:P NS))
      (IMPLIES (ENDP NS) (:P NS))).
This induction is justified by the same argument used to admit SUM.
```

At the bottom right of the window, there is a "Read/Write" button and a small input field.

Outline

① Background

② Conjecture

③ Experiment

④ Evaluation

ACL2-based Logic Course

Purpose: Replacement for *Symbolic Logic*

Target: Students from *Fundamentals I*

Curriculum: Formal logic and ACL2 Verification

Trial Run: Spring 2007

Format: Half-credit class

Size: 6 freshmen, high A to mid B

Syllabus

- 1 Introduction
- 2 Structural Induction
- 3 Automated Theorem Proving
- 4 Expanding on Induction
- 5 Final Project

Syllabus

1 Introduction

Lecture Topic

ACL2 Syntax

Propositional logic

Homework

Simple programs

Validity checker

2 Structural Induction

3 Automated Theorem Proving

4 Expanding on Induction

5 Final Project

Syllabus

① Introduction

② Structural Induction

Lecture Topic

Structural induction principles
Inductive proofs

Homework

Examples by hand
Examples by hand

③ Automated Theorem Proving

④ Expanding on Induction

⑤ Final Project

Structural Induction Principles

$\text{LoN} = \text{nil} \mid (\text{cons Number LoN})$

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

$\forall l \in \text{LoN}. P(l)$

Structural Induction Principles

$$\text{LoN} = \text{nil} \mid (\text{cons Number LoN})$$

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

```
if  $P(\text{nil})$ 
and
       $P((\text{cons } n \ 1))$ 
then  $\forall l \in \text{LoN}. P(l)$ 
```

Structural Induction Principles

$$\text{LoN} = \text{nil} \mid (\text{cons Number LoN})$$

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

$$\begin{array}{l} \text{if } P(\text{nil}) \\ \text{and } \forall l \in \text{LoN}. \forall n \in \text{Number}. \\ \quad P((\text{cons } n \ l)) \\ \text{then } \forall l \in \text{LoN}. P(l) \end{array}$$

Structural Induction Principles

$$\text{LoN} = \text{nil} \mid (\text{cons Number LoN})$$

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

```

if P(nil)
and  $\forall l \in \text{LoN}. \forall n \in \text{Number}.$ 
    $P(l) \Rightarrow P((\text{cons } n \ l))$ 
then  $\forall l \in \text{LoN}. P(l)$ 

```

Syllabus

- 1 Introduction
- 2 Structural Induction
- 3 Automated Theorem Proving

Lecture Topic	Homework
ACL2 strategies	Verify binary tree insert
Proof theory	Proof checker, ACL2 proofs

- 4 Expanding on Induction
- 5 Final Project

ACL2 Strategies

- Work out proofs by hand.
- Compare ACL2 output to hand proof.
- Read early checkpoints.
- Guide ACL2 with lemmas.

Fragile Solutions

The screenshot shows two windows in DrScheme. The left window, titled 'isort.lisp - DrScheme', contains the following code:

```

;; in : Any List[Any] -> Boolean
(defun in (x S)
  (cond ((endp S) nil)
        ((consp S) (or (equal x (car S))
                       (in x (cdr S)))))

;; del : Any List[Any] -> List[Any]
;; remove one occurrence of x from S
(defun del (x S)
  (cond ((endp S) nil)
        ((consp S) (if (equal (car S) x)
                       (cdr S)
                       (cons (car S)
                             (del x (cdr S))))))

;; perm : List[Any] List[Any] -> Boolean
;; is xs a permutation of ys?
(defun perm (xs ys)
  (cond ((endp xs) (endp ys))
        ((consp xs)
         (and (in (car xs) ys)
              (perm (cdr xs) (del (car xs) ys)))))

;; other half of isort correctness:
(defthm isort-produces-perm
  (perm (isort xs) xs))

```

The right window, titled 'acl2.1195170089.txt - DrScheme', shows the proof output:

```

(DEFTHM ISORT-PRODUCES-PERM ...)
Q.E.D.

We will induct according to a scheme suggested by (INSERT XS1 IT),
but modified to accommodate (PERM IT XS2). These suggestions were
produced using the :induction rules INSERT and PERM. If we let
(:P IT XS1 XS2) denote *l.1 above then the induction scheme we'll use
is
(AND (IMPLIES (AND (NOT (ENDP IT)) (NOT (CONSP IT)))
             (:P IT XS1 XS2))
      (IMPLIES (AND (NOT (ENDP IT))
                    (CONSP IT)
                    (NOT (<=< XS1 (CAR IT)))
                    (:P (CDR IT) XS1 (DEL (CAR IT) XS2)))
             (:P IT XS1 XS2))
      (IMPLIES (AND (NOT (ENDP IT))
                    (CONSP IT)
                    (<=< XS1 (CAR IT)))
             (:P IT XS1 XS2))
      (IMPLIES (ENDP IT) (:P IT XS1 XS2))).

```

At the bottom right of the right window, there is a 'Read/Write' button.

Syllabus

- 1 Introduction
- 2 Structural Induction
- 3 Automated Theorem Proving
- 4 Expanding on Induction

Lecture Topic

Generalized induction

Proof about quicksort

Proofs w/accumulators

Homework

Essay: quicksort

Lemmas by hand

Verify quicksort, accumulators

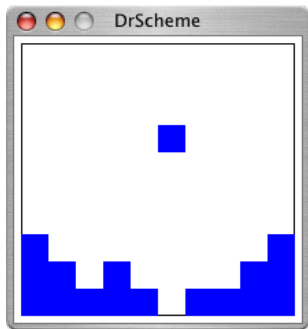
- 5 Final Project

Syllabus

- 1 Introduction
- 2 Structural Induction
- 3 Automated Theorem Proving
- 4 Expanding on Induction
- 5 Final Project

Lecture Topic	Homework
First-order logic	Tetris program

Final Project



Assignment: *Tetris*-like game

Given: One block, falling endlessly

In-class goal: Fix program; prove blocks hit bottom and stop falling

Final goal: 2-3 new, verified *Tetris* features

Student Performance

In Class: Contributed frequently, presented well

Logic: Proficient at induction with occasional prompting

Programming: “Forgot” the Design Recipe

ACL2: Could prove some theorems; gave up on the rest

Outline

① Background

② Conjecture

③ Experiment

④ Evaluation

Exit Interviews

- Fast-paced and challenging
- Overwhelmed by ACL2 output
- Underprepared for proof strategies, logic notation
- **Liked the class enough to stay late on Friday afternoon.**

Student Accomplishments

- Systematic structural induction
- Presentation skills
- Write, verify ACL2 programs
- All in half a regular course

Future Directions

- Stress the Design Recipe
- Begin ACL2 proofs earlier
- Unified proof strategy
- Simplified readout from ACL2
- Canon of robust proof exercises
- Fix, extend, document Dracula

Success

Northeastern adopted the course.

The End

Thank You!