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

Outline

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)   )
   (t
                              )))

(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)
              (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

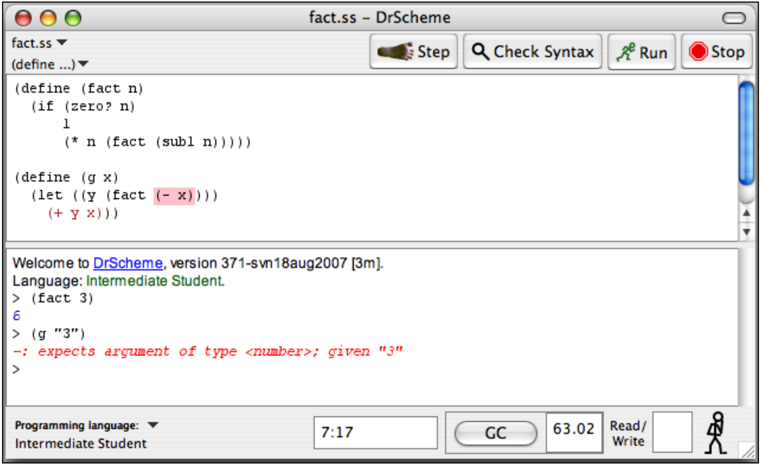
The Six-Step Design Recipe

```
;; 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
```

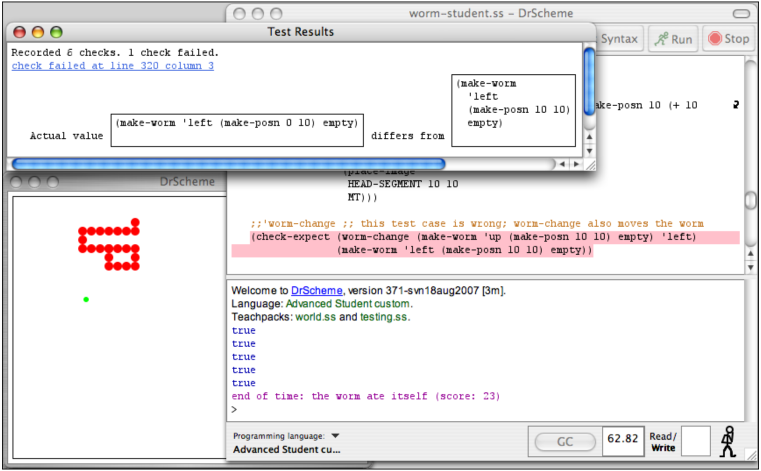
1. Data Definition
2. Contract & Purpose
3. Examples
4. Template

   Multiple clauses?
   Use cond.
   Compound data?
   Apply accessors.
   Inductive data?
   Recur.
5. Write Code
6. Run Tests

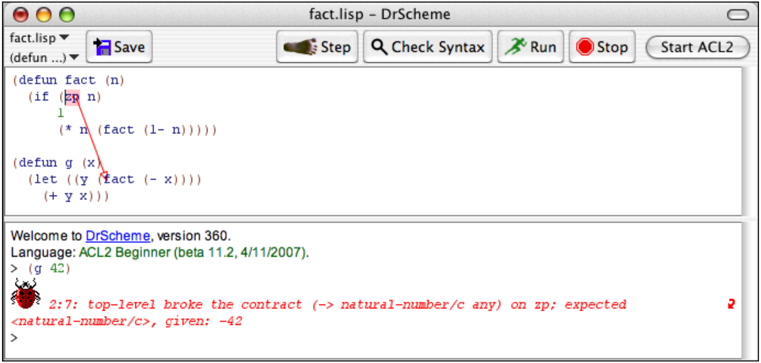
Student Languages



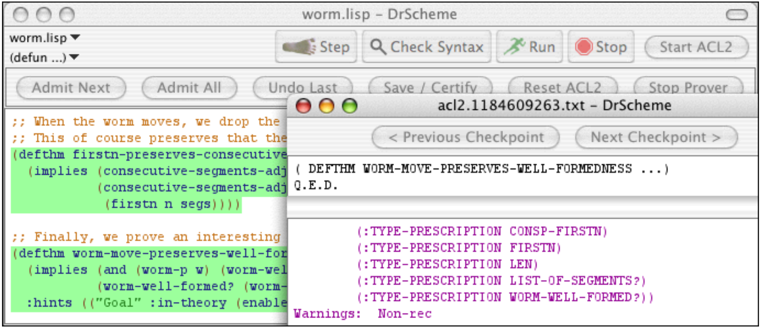
Teachpacks


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)
Syntax
Run
Stop
HEAD-SEGMENT 10 10
MT)))
;;'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))
DrScheme
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

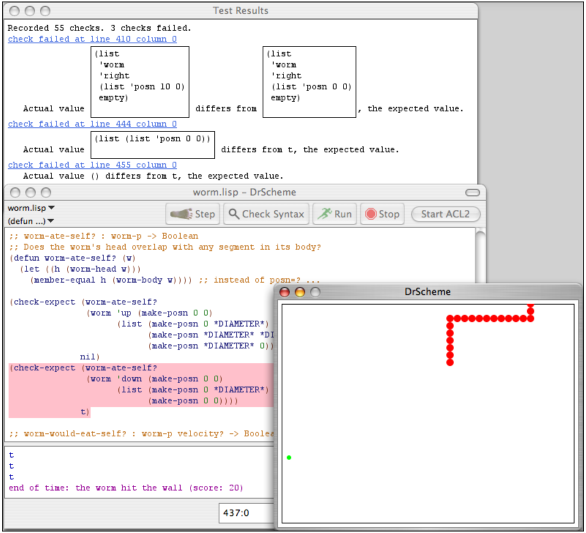
Dracula Language



Dracula Proofs



Dracula Teachpacks



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

Remember the S.A.T.?

Logic : Computing :: Analysis : Physics

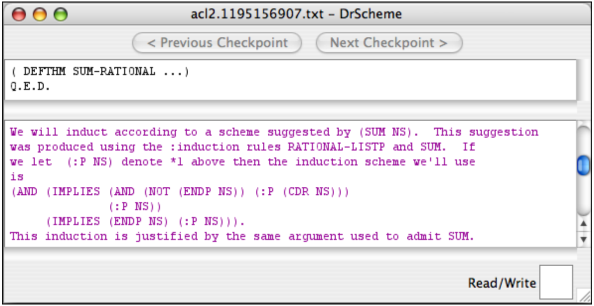
Preparing Freshmen for ACL2

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

Multiple clauses?
Use cond.

Compound data?
Apply accessors.

Inductive data?
Recur.



Outline

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	**Homework**
ACL2 Syntax	Simple programs
Propositional logic	Validity checker

2. Structural Induction
3. Automated Theorem Proving
4. Expanding on Induction
5. Final Project

Syllabus

1. Introduction
2. Structural Induction

Lecture Topic	Homework
Structural induction principles	Examples by hand
Inductive proofs	Examples by hand

3. Automated Theorem Proving
4. Expanding on Induction
5. Final Project

Structural Induction Principles

```
LoN = nil | (cons Number LoN)
```

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

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

Structural Induction Principles

```
LoN = nil | (cons Number LoN)
```

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

$$
\begin{array}{rl}
\text{if} & P(\texttt{nil}) \\
\text{and} & \\
 & \qquad\qquad P((\texttt{cons n l})) \\
\text{then} & \forall \texttt{l} \in \texttt{LoN}.\ P(\texttt{l})
\end{array}
$$

Structural Induction Principles

`LoN = nil | (cons Number LoN)`

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

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

Structural Induction Principles

```
LoN = nil | (cons Number LoN)
```

Multiple kinds of data? Add hypotheses.

Structures with fields? Add quantifiers.

Inductive data definition? Add inductive hypothesis.

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

Syllabus

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

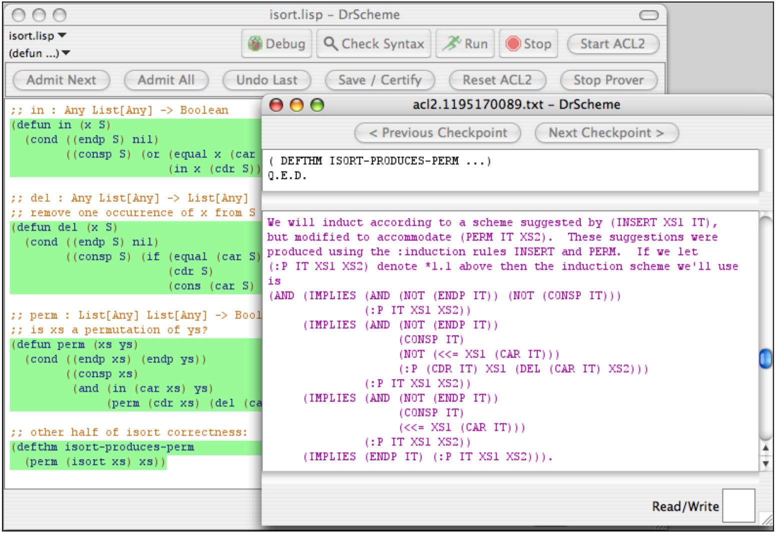
Lecture Topic	Homework
ACL2 strategies	Verify binary tree insert
Proof theory	Proof checker, ACL2 proofs

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



Syllabus

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

Lecture Topic	**Homework**
Generalized induction	Essay: quicksort
Proof about quicksort	Lemmas by hand
Proofs w/accumulators	Verify quicksort, accumulators

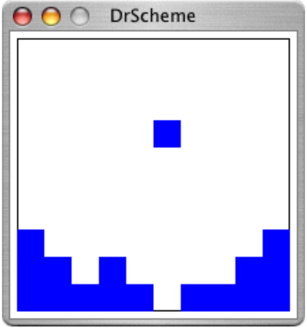
5. Final Project

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

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!