Designing Programs
Introduction to ACL2s

Pete Manolios
Northeastern

Logic and Computation, 1/9/2019

Objectives

- Course Webpages
- Designing programs, review
- Introduction to the ACL2s language

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Course Webpages

- Review course webpages

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Overview of the Class

- I will be releasing sparse lecture notes every so often
- Read the lecture note before class
  - The point of college is to learn how to learn on your own
  - Expect to get lots of practice with that in this course
  - Related issues such as security, efficiency, applications will show up in class
- Homeworks will introduce new concepts and applications (learning to learn)
- We will be using Piazza (should all have gotten invitations)
- No recordings allowed
- No electronics in class without prior approval (phones, laptops)
  - Just turn off or mute your phones before class
  - We only meet for ~3 hours a week
  - Each of those hours costs you about $370, so make the most of it

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Quiz

```
;; rl: List x Nat -> List
;; Given a list, l, and a natural number, n, rl rotates the list
;; to the left n times

;; Primary consideration: correctness
;; No need to worry about efficiency
```

```
(check= (rl (list 1 2 3) 1) (list 2 3 1))
(check= (rl (list 1 2 3) 2) (list 3 1 2))
(check= (rl (list 1 2 3) 3) (list 1 2 3))
```

Designing Program

```
;; len: List -> Nat
;; Given a list, len returns the length of the list
```

1. Identify data definitions
2. Write a description

```
(definec len (l :tl) :nat
  (if (endp l)
      . . .
      (. . . (len (rest l)) . . .)))
```

4. Formalize contracts

5. Data-driven definition template

```
(check= (len (list) 0))
(check= (len (list 1 2)) 2)
```

3. Test Cases

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Designing Program

```
;; len: List -> Nat
;; Given a list, len returns the length of the list
```

1. Identify data definitions
2. Write a description

```
(definec len (l :tl) :nat
  (if (endp l)
      0
      (+ 1 (len (rest l)))))
```

4. Formalize contracts
5. Data-driven definition template
6. Complete data-driven definition

```
(check= (len (list) 0))
(check= (len (list 1 2)) 2)
```

3. Test Cases

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

ACL2s vs Racket

```
;; len: List -> Nat
;; Given a list, len returns the length of the list
```

```
(definec len (l :tl) :nat
  (if (endp l)
      0
      (+ 1 (len (rest l)))))
```

```
(define (len l)
  (if (empty? l)
      0
      (+ 1 (len (rest l)))))
```

```
(check= (len (list) 0))
(check= (len (list 1 2)) 2)
```

```
(check-expect (len (list) 0))
(check-expect (len (list 1 2)) 2)
```

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

ACL2s vs Racket

```
;; len: List -> Nat
;; Given a list, len returns the length of the list
```

```
(definec len (l :tl) :nat
  (if (endp l)
      -1
      (+ 1 (len (rest l)))))
```

```
(define (len l)
  (if (empty? l)
      -1
      (+ 1 (len (rest l)))))
```

ACL2s will not accept the above definition, but Racket will.
Contracts allow ACL2s to check function signatures.

ACL2s

- We will use ACL2s, based on ACL2, which consists of
  - a LISP-based language with contracts
  - a logic that makes it clear how to state properties and prove theorems
  - a theorem prover that automates much of the tedious effort involved and guarantees that we did not make mistakes.
- ACL2 won the software system award from the ACM.
  - It is used in industry, eg, AMD, Rockwell, IBM, Intel, GE, Centaur, …
- We show how to use logic to formalize the syntax and semantics of the core ACL2s language
- We then use the ACL2s language to
  - formally reason about programs
  - to model systems at various levels of abstraction
  - to design and specify interfaces between systems
  - to reason about such composed systems

Why is len Well-Defined?

- Why does this definition make sense?
- Because it terminates; we'll cover that later
- A key idea every time you define a program is to convince yourself that on every recursive call, some parameter decreases in a well-founded way
- Hmm, can lists be circular? then what?
- Lists are non-circular in ACL2s, which is why this works
- Termination is one of the key ideas in CS
- Note that data driven definitions always terminate
- That's why it is a good idea to use the template

```
(definec len (l :tl) :nat
   (if (endp l)
    0
    (+ 1 (len (rest l)))))
```

```
(definec len (l :tl) :nat
   (if (endp l)
    (+ 1 (len (rest l)))
    0))
```

What if I wrote this?

Next Time

- Tour of ACL2s: install ACL2s and experiment with it
- rl: Define and test the function using Racket or …
- Skim the lecture notes
- Lecture notes release announcement coming on Piazza

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019