The ACL2s Language

Pete Manolios
Northeastern

Logic and Computation, 1/14/2019

Objectives

- Basic Data Types
- Expressions
- Syntax and Semantics of atomic data and primitives

ACL2 Universe

- We are done with the review of programming
- Now, we start a careful examination of the ACL2s language
- Programs manipulate objects from the ACL2 universe
- What's in the universe?

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

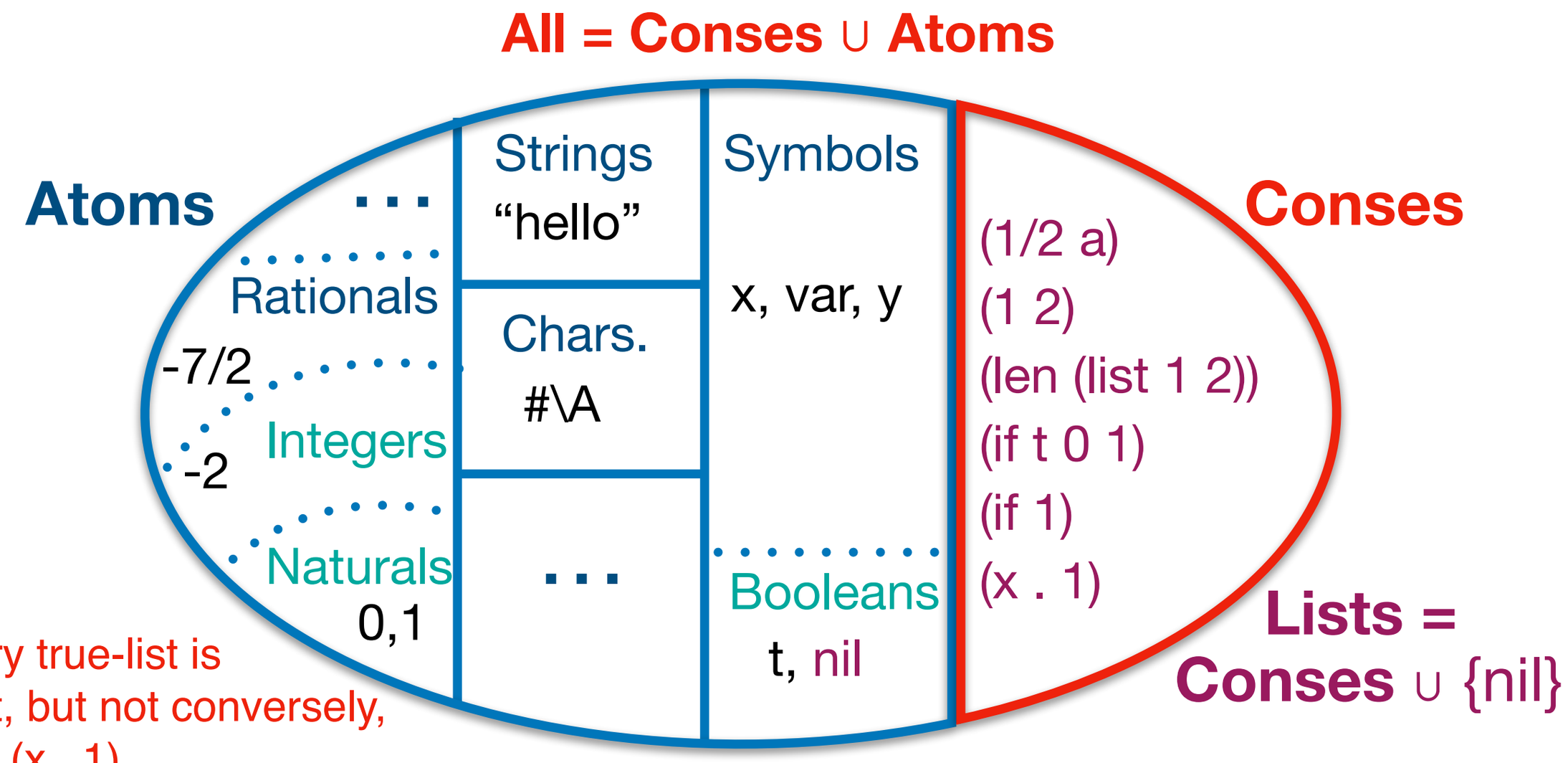
Quiz 1

- t is a
  - A. symbol
  - B. atom
  - C. boolean
  - D. A & B
  - E. B & C
  - F. A & B & C

Always pick the best answer

For example, if A, B and C are true, pick F

ACL2 Universe



True-Lists = $\cup_{i \in \mathbb{N}} TL_i$

$TL_0 = \{ () \}$, $TL_{i+1} = TL_i \cup \{(\texttt{cons x l}): \texttt{x}\in\texttt{All}, \texttt{l}\in TL_i\}$

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

Quiz 2

- (if ( ) ( ) 4) is
  - A. an expression
  - B. an atom
  - C. a list
  - D. A & C
  - E. B & C
  - F. A & B & C

Expressions

- ⟦*expr*⟧ denotes the semantics of *expr*
  - or what *expr* evaluates to at the REPL
- Constants are expressions that evaluate to themselves
  - `⟦t⟧ = t`
  - `⟦nil⟧ = nil`
  - `⟦6⟧ = 6`
  - `⟦-21⟧ = -21`

Booleans

- Built-in functions & signatures
  - `if: All × All × All → All`
- Expressions?
  - `(if nil 0 1)` Yes; signature satisfied
  - `(if nil 0)` No; arity of if is 3, not 2
  - `(if 1 2 3)` Yes; signature satisfied
- Semantics of `if`
  - ⟦`(if` *test then else*`)`⟧ = ⟦*then*⟧ , when ⟦*test*⟧ ≠ `nil`
  - ⟦`(if` *test then else*`)`⟧ = ⟦*else*⟧ , when ⟦*test*⟧ = `nil`
- We specify semantics only for expressions (signature is satisfied)
- Examples
  - ⟦`(if t nil t)`⟧ = `nil`
  - ⟦`(if (if t nil t) 1 2)`⟧ = ⟦`(if nil 1 2)`⟧ = `2`

Lazy vs Strict

- Semantics of `if`
  - ⟦(`if` *test then else*)⟧ = ⟦*then*⟧ , when ⟦*test*⟧ ≠ `nil`
  - ⟦(`if` *test then else*)⟧ = ⟦*else*⟧ , when ⟦*test*⟧ = `nil`
- `if` is lazy:
  - first ACL2s evaluates *test*, i.e., it computes ⟦*test*⟧
  - if ⟦*test*⟧ ≠ `nil` then ACL2s returns ⟦*then*⟧
  - otherwise, it returns ⟦*else*⟧
- So, *test* is always evaluated, but only one of *then*, *else* is
- All other functions are strict
  - ACL2s evaluates all of the arguments to the function
  - Then ACL2s applies the function to evaluated results

Equal

- Built-in functions & signatures
  - `if: All × All × All → All`
  - `equal: All × All → Boolean`
- Semantics of `equal`
  - ⟦`(equal x y)`⟧ = `t` iff ⟦`x`⟧ = ⟦`y`⟧
  - i.e., ⟦`(equal x y)`⟧ = `t` if ⟦`x`⟧ = ⟦`y`⟧ and `nil` otherwise
- Examples
  - ⟦`(equal 3 nil)`⟧ = `nil`
  - ⟦`(equal (if (if t nil t) 1 2) 2)`⟧ = `t`

Booleanp

- The “p” is for *predicate:* `booleanp` is a *recognizer* for Booleans
- A recognizer takes anything as input and returns a Boolean

This is weird; the only time we will see this

```
(definec booleanp (x :all) :boolean
  (if (equal x t)
      t
    (equal x nil)))
```

Semantics of Defined Functions Example

⟦(booleanp 3)⟧

= { Semantics of `booleanp` }

⟦(if (equal 3 t) t (equal 3 nil))⟧

= { Semantics of `equal`, `if`}

⟦(equal 3 nil)⟧

= { Semantics of equal }

nil

Defined Functions, v1

How would you define and (conjunction)?

```
(definec and (a :bool b :bool) :bool
  (if a b nil))
```

Not the way "and" is really defined! We'll see why soon.

Numbers

- Built-in functions & signatures
  - `integerp: All → Boolean`
  - `rationalp: All → Boolean`
- Semantics
  - ⟦`(integerp x)`⟧ is `t` iff ⟦`x`⟧ is an integer
  - ⟦`(rationalp x)`⟧ is `t` iff ⟦`x`⟧ is a rational
- In ACL2s, we get “real” numbers, not approximations (Java & C)
- Remember integers are rationals

Numeric Functions

- Built-in functions & signatures
  - `+, *: Rational × Rational → Rational`
  - `<: Rational × Rational → Boolean`
  - `unary--: Rational → Rational`
  - `unary-/: Rational \ {0} → Rational`
- What is wrong with this definition?

```
(definec posp (a :all) :bool
  (and (integerp a) (< 0 a)))
```

Contract violation!
How do we fix?

```
(definec posp (a :all) :bool
  (if (integerp a) (< 0 a) nil))
```

Maybe and should be lazy?
But functions are strict
Macros! (Abbreviation power)

Defined Functions, v2

```
(and)       → t
(and a)     → a
(and a b)   → (if a b nil)
(and a b c) → (if a (if b c nil) nil))
```

and, or are macros
macros are first expanded
then evaluation happens

```
(or)       → nil
(or a)     → a
(or a b)   → (if a a b)
(or a b c) → (if a a (if b b c))
```

Defined Functions

```
(definec implies (a :all b :all) :bool
  (if a (if b t nil) t))

(definec not (a :all) :bool
  (if a nil t))
```

```
(definec iff (a :all b :all) :bool
  (if a
      (if b t nil)
    (if b nil t)))

(definec xor (a :all b :all) :bool
  (if a
      (if b nil t)
    (if b t nil)))
```

Rationals

- Built-in functions & signatures
  - `numerator: Rational → Integer`
  - `denominator: Rational → Pos`
- `Examples`
  - ⟦2/4⟧ = 1/2
  - ⟦(/ 132 -765)⟧ = -44/255
- Rules
  - To simplify x/y, where x is an integer and y a natural number, divide both by the gcd(x,y) to obtain a/b. If b=1, return a, else return a/b
  - ACL2s simplifies rationals
  - Note that ⟦(equal 4/2 2)⟧ = t

Next Time

- Real quiz; install PollAnywhere
- Conses
- Contract violations
- Read to the end of section 2.8