Primitive ACL2 Datum Objects and Functions

Objects and Functions Talk

Warren A. Hunt, Jr.
hunt@cs.utexas.edu

Computer Science Department
University of Texas
2317 Speedway, M/S D9500
Austin, TX 78712-0233

January, 2026

Primitive ACL2 Functions

Here we investigate primitive ACL2 objects, constructors and access functions.

- The core of ACL2 is defined by 32 functions and the objects these functions manipulate.
- ACL2 has five data types: numbers, characters, strings, symbols, and pairs.
- ACL2 has 32 primitive functions that operate on ACL2 data objects.
- As a convention, ACL2 recognizes certain symbols – those that begin and end with a * character – as representing constant values.
- For example, the symbol `*three*` can be associated by the ACL2 theorem-proving system with the number 3 by presenting the ACL2 theorem-proving system with:

```
ACL2 !>(defconst *three* 3)
Summary
Form:  ( DEFCONST *THREE* ...)
Rules: NIL
Time:  0.00 seconds (prove: 0.00, print: 0.00, other: 0.00)
 *THREE*
ACL2 !>*three*
3
```

ACL2 Primitive Functions

Here are the ACL2 pre-defined function symbols:

```
ACL2 !>(strip-cars *primitive-formals-and-guards*)
( ACL2-NUMBERP                    ;; Recognizes all numbers, Later
  BAD-ATOM<=                      ;; Later
  BINARY-*
  BINARY-+
  UNARY--
  UNARY-/
  <
  CAR
  CDR
  CHAR-CODE
  CHARACTERP
  CODE-CHAR
  COMPLEX                          ;; Complex numbers, Later
  COMPLEX-RATIONALP                ;; Complex numbers, Later
  COERCE
  CONS
  CONSP
```

ACL2 Primitive Functions, continued

```
DENOMINATOR                      ;; Rational numbers, Later
EQUAL
IF
IMAGPART                         ;; Complex numbers, Later
INTEGERP
INTERN-IN-PACKAGE-OF-SYMBOL      ;; Packages, Later
NUMERATOR                        ;; Rational Numbers, Later
PKG-IMPORTS                      ;; Package (naming) system, Later
PKG-WITNESS                      ;; Package (naming) system, Later
RATIONALP                        ;; Rational numbers, Later
REALPART                         ;; Complex numbers, Later
STRINGP
SYMBOL-NAME                      ;; Packages, Later
SYMBOL-PACKAGE-NAME              ;; Packages, Later
SYMBOLP)
```

Initially, we will confine ourselves to the functions without comments.

Constructing a Logic is Subtle

Defining a logic is a subtle issue as it involved defining all of the object, functions, and their meanings.

Certainly, it is possible to create something *inconsistent* – meaning that through some operation(s), one can demonstrate something that should not be possible.

In ACL2, the upper-case symbols `T` and `NIL` are reserved; these distinct symbols represent *true* and *false*, respectively.

An axiom of ACL2 is that distinct symbols are not equal.

But, imagine that we could prove, by using the axioms (the given truths) and the rules of inference (the means to conclude new truths from existing truths), that `T` and `NIL` were the same.

This would be an inconsistency in the definition of the logic, and would render ACL2 useless.

We will assume that ACL2 is defined in a logically-consistent manner.

Demonstrating that ACL2 is inconsistent or somehow flawed would certainly raise one's grade!

"Incestuous" Core Logic Definitions

On *faith*, we accept that pairs are created by the `CONS` function.

On *faith*, we will accept that we have the natural numbers.

Instead of using *sticks* (or any other objects) to represent natural numbers, we will abbreviate them as 0, 1, 2, and so on.

Initially, one may think of ACL2 datum objects as being created from natural numbers, pairs, and *wrapper* functions (with restricted arguments) into other ACL2 datum objects.

For example, a character can be created by *wrapping* a natural number (between 0 and 127, inclusive) with the `code-char` primitive.

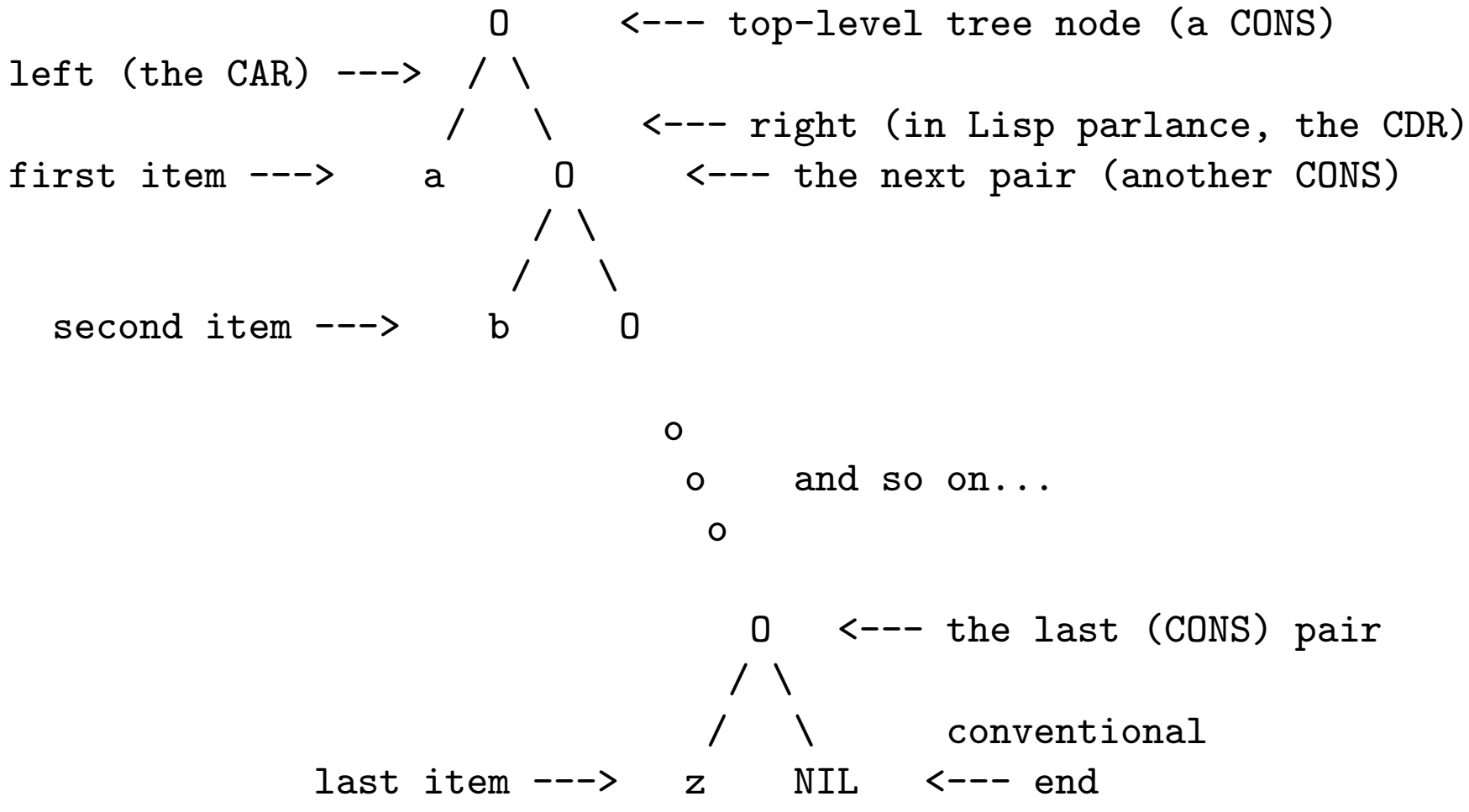
```
ACL2 !> (code-char 97)     ;; "Wrap" 97 into a character
#\a
ACL2 !>(char-code #\a)     ;; Peel off the wrapper
97
```

Strings can be created by *wrapping* a right-associated, list of characters with the `coerce` primitive function:

```
ACL2 !>(coerce (list #\a #\b #\c) 'string)
"abc"
```

Aggregating a Collection of ACL2 Objects into a Proper List

We will use CONS pairs to create collections of ACL2 objects, recognized by the ACL2 TRUE-LISTP function.



where each item (a, b, ..., z) are the elements.

Creating a String

Given a CHARACTER-LISTP object, we can create a string.

```
:pe character-listp
 V     -8355  (DEFUN CHARACTER-LISTP (L)
                 (DECLARE (XARGS :GUARD T :MODE :LOGIC))
                 (COND ((ATOM L) (EQUAL L NIL))
                       (T (AND (CHARACTERP (CAR L))
                               (CHARACTER-LISTP (CDR L))))))
```

```
ACL2 !>:pe characterp
CHARACTERP is built into ACL2 without a defining event.  See :DOC
CHARACTERP.  See :DOC ARGS for a way to get more information about
such primitives.  See :DOC primitive for a list containing each
built-in function without a definition, each associated with its
formals and guard.
ACL2 !>
```

Strings can be created by *wrapping* a right-associated, list of characters with the coerce primitive function:

```
ACL2 !>(coerce '(#\a #\b #\c) 'string)
"abc"
```