1 Texts and their Denotations

The denotation of a text is the data object that it represents. Some examples include:

- 7.5 represents/denotes the number seven and a half
- 17 denotes the number seventeen
- CS17 denotes the symbol CS17
- cs17 denotes the symbol cs17 (which is different from CS17)
- + denotes the symbol “+”
- (a b c) denotes a list of three symbols: a, b and c
- (+ 3 7) denotes a list consisting of the symbol + and the numbers 3 and 7
2 (Abbreviated) Rules of Evaluation

- The value of a number is itself.
- The value of a Boolean is itself.
- The value of a symbol is the data object it is bound to.
- The value of a list is obtained as follows: evaluate each item in the list. The first item should evaluate to a procedure. Apply the procedure to the values of the rest of the items. The result of the procedure application is the value of the list.

Examples (in arithmetic)

- The symbol + is bound to the addition procedure
- The symbol * is bound to the multiplication procedure
- The symbol / is bound to the division procedure
- The symbol - is bound to the subtraction procedure

In Racket, there is no text to denote a procedure. Rather, the text “+” denotes a symbol that is bound to a procedure.

3 Expressions

3.1 Recursive/Inductive Definitions

A recursive definition is one that defines an element in a set in terms of other elements in the set. Some examples of this include:

- One of the listed ingredients for beef stock is beef stock
- GNU, which stands for “GNU is not Unix”
- LISP data object, which is a symbol, number, Boolean, or a list whose items are LISP data objects
- Foo, a made-up definition where a foo is either “#” or “*” followed by a foo followed by “-”

3.2 Defining an Expression

An expression is either a:

- Number
- Boolean
• Symbol

• List consisting of 1 or more items, where each item is an expression and the value of the first item is a procedure. (Note: Each of the items in a list is a subexpression)

This definition for an expression is recursive because one of the possible expressions, a list, is itself defined in terms of expressions.

3.3 Quiz

For today’s quiz, Professor Klein asked students how one can tell whether “********#——–” is a foo using the definition provided. We start at the first symbol, “*”, and see that this will be a foo only if followed by a foo followed by a “-”. We continue to the next symbol to see if the first is followed by a foo. Continuing to do this, we find that the term “********#——–” is a foo only if the number of *’s is equal to the number of -’s with the “#” directly in the middle.

3.4 Examples of Expressions (and Non-Expressions)

In class, we came up with various expressions using our recursive definition. Some expressions include:

• 4
• true
• x
• cs17
• four
• (+ 2 1)

We also discussed things that did not fit our definition of an expression:

• (3 4 5)
• (7 + 3)
• (+ (1 2) 1)
• ()

The first two examples are not expressions because the definition states that in order for a list to be an expression, the value of the first item must be a procedure. For both of these lists, the value of the first item is a number, so the lists are not expressions. In the third example, the value of the first item in the list is a procedure, but (1 2) is not an expression because its first item evaluates to a number. Therefore (+ (1 2) 1) is not an expression because it violates the rule that each item in the list must itself be an expression. Finally, the empty list is not an expression because our definition states that a list cannot be an expression unless it contains one or more items.
4 Evaluation vs. a Function

4.1 Errors when Evaluating

It is important to realize that not every valid expression evaluates successfully!

There are multiple ways that evaluation can fail on a valid expression:

- a symbol that is not bound to anything
- a procedure itself could fail, causing evaluation to fail

For example, let’s look at (* false 3). This is a valid expression because it is a list with one or more items, where each item is itself an expression and whose first item evaluates to a procedure. What would happen if we tried to evaluate this expression? We already know that the first item in the list evaluates to the * procedure. This procedure expects to be applied to two numbers. However, here we attempt to apply it to a boolean. This yields an error.

The division procedure requires that all but the first of the numbers that it is applied to are nonzero. It will fail if this rule is violated.

4.2 Functions as Machines

Think of a function as a “function box”. We supply this box with an input and it returns an output. For example, when we input the number 7 to the Add 3 function box, the output will be the number 10.

By the definition of a function, for any legal input (defined by the domain of the function), there is an output. Because of this we say that a function is \textit{total}.

5 HEALTH

This is important!!!

- Get sleep.
- Get exercise.
- Treat others in the community with respect and kindness.
- If you are sick, take care of yourself and others (and don’t go to Professor Klein’s office hours!!)
- The University has resources to help you if you feel that you need it!

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: \url{http://cs.brown.edu/courses/csci0170/feedback}.