

CS101

Ramkumar R

September 30, 2009

Introduction

What is a prime number?

- ▶ A number whose only factors are one and itself
- ▶ A natural number that is not a multiple of any natural number except 1 and the number itself
- ▶ A (natural number) that is (not a multiple of) (any (natural number) except 1 and (the number itself))
- ▶ ((natural number) (not a multiple of) (any (natural number) except 1 and (the number itself)))

Enter: Literals and symbols

Problem 1: Assert if 27 is a prime number. Start off by specializing the previous expression to fit this problem statement.

- ▶ (27 (not a multiple of) (any (natural number) except 1 and 27))
- ▶ Now, we must *evaluate* this *expression*. (3 + 2) evaluates to a natural number, 5. What kind of data do we expect the above expression to evaluate to? **True** or **False**
- ▶ 1 and 27 are constants or *literals*
- ▶ (natural number) is a variable. Let us then denote it by a *symbol*, say M
- ▶ (27 (not a multiple of) M), where M is **any** natural number except 1 and 27
- ▶ Analyze (not a multiple of). It's some condition that requires two entities to operate. When this condition is satisfied, the complete expression evaluates to **True**, otherwise **False**

Enter: Operators

- ▶ How do we represent (not a multiple of)? Think about the operation itself. What do we mean when we say (3 (not a multiple of) 2)? It means that when 3 is divided by 2, it leaves a non-zero remainder (provided $2 \leq 3$ of course, by definition of multiple).
- ▶ ((remainder of $(27 / M)$) (is not equal to) 0), where M is a **any** natural number except 1 and 27
- ▶ Analyze $(27 / M)$... / is an operator that accepts two numbers (natural numbers in this case), 27 and M, as operands. $(27 / M)$ evaluates to another number (unless $M = 0$ of course, in which case the operation is undefined), which is, in the general case, a decimal number. Let us define a new operator % which magically divides the first operand by the second, and evaluates to the resulting remainder. Notice that **any** is also an operator that picks one item from a set of items.
- ▶ $((27 \% M) != 0)$ is our new expression, where we have replaced (is not equal to) by the operator !=

- ▶ Now, we have to tell the machine what M is.
- ▶ “where M is a any natural number except 1 and 27” must now be modified to fit the definition of multiple. $27 \% M = 0$ is the condition for 27 being a multiple of M , provided that $M \leq 27$.
- ▶ So M is a variable number, whose value can be picked up from the set $[1, 2, 3 \dots 27]$ - $[1, 27] = [2, 3 \dots 26]$

Putting it all together

- ▶ $((27 \% M) \neq 0)$ where $M = (\mathbf{any} [2, 3 .. 26])$
- ▶ $(\mathbf{all} ((27 \% M) \neq 0))$ where $M = [2, 3 .. 26]$
- ▶ $\mathbf{all} (\backslash m \rightarrow 27 \text{ 'mod' } m \neq 0) [2, 3 .. 26]$

In case you haven't noticed, the last line is a valid Haskell program!

Where to go from here

- ▶ Problem 2: Assert if 17 is a prime number. Now we have to change 27 to 17 and 26 to 16. This is a pain, especially in larger programs. So let's just replace it by a symbol, say N. Then
all $(\ n \rightarrow 27 \text{ 'mod' } m \neq 0) [2, 3 .. n-1]$ where $n = 17$
- ▶ Now reuse this code to find all the prime numbers ≤ 100