Set Theory and List Comprehension

Jamil Dhanani

March 2014

1 Introduction

In high school, you probably came across something called *set notation*. It's a useful invention related to the branch of mathematics that deals with, of course, these "objects" called *sets*. It turns out that some of the notation and concept related ot set theory can be extremely useful in programming. Indeed, most languages have some sort of attempt at imitating the basic notation of sets in order to allow you to write simple, readable, and powerful code.

2 A Simple Problem

Let's say I asked you to give me a list of the numbers, from 1 to 1000, that are divisible by 12 and 13. If you were to write a program to do this, it's most likely that you would come up with something along the lines of:

```
for i from 1 to 1000
  if i is divisible by 12 OR if i is divisible by 13
    add i to the list of numbers divisible by 12 or 13
```

This code tells you *how* to arrive at an answer, but it does not clearly express *what* you want. Of course, programming at the imperative level is more concerned about the "how" than the "what". This makes for simple readability and comprehension for the assembler, but not quite for the human brain.

Some programming languages, such as Haskell, have employed this syntax of list comprehension, which closely resembles the notation of sets. Consider the definition of the above problem in a set-based syntax:

$$\{x : 1 < x \leq 1000,\ \ x \bmod 12 = 0,\ \ x \bmod 13 = 0\} \tag{1}$$

Although this may seem obfuscating at first, reading it aloud is like plain and simple English: "x, such that x is in between one and a thousand, that x mod 12 is 0, and that x mod 13 is 0". This is, in fact, just a repetition of the original problem. In Haskell, this would be something like:

```
[x | x <- [1..1000], x % 12 == 0, x % 13 == 0]
```