Laziness

Eric Seamons and Joshua Katuka

3/4/2016

Java

Java is eager in its evaluation because it evaluates a numeric expression immediately after assigning it to a variable or passing it into a function as a parameter.

The following line of code will throw an error in java:

```
int div_by_zero = 1 / 0;
```

On the other hand, languages with lazy evaluation will hold off the evaluation until the data is needed. In lazy racket, you can do the following

```
(define div-by-zero  (/ 1 0))
```

and it will not throw an error. However if you use div-by-zero in an arithmetic expression in racket

```
(+ 1 div-by-zero)
```

It will throw an error. In java you would get an error upon assignment, since java eagerly evaluates expressions.

The program below shows that java eagerly evaluates. As soon as 1/0 is assigned to a variable, an error is thrown. It doesn't wait until other calculations are performed. The error is caught and the program doesn't even make it into the isPrime function.

If Java were lazy the program would have printed out

> "Number was assigned but not evaluated since Java is lazy" followed by
>
> "Division by zero"

However the program just prints out

> "Division by zero"

Because Java is eagerly evaluating the expression 1 / 0 and it immediately throws an error

```
public class Primality {

	public static void main(String[] args) {

		boolean result = false;

		try {
			result = isPrime(1 / 0);
			System.out.println(result);
		} catch (ArithmeticException e) { //
				System.out.println("Division by zero");
		}

	}

	public static boolean isPrime(int candidate_number) {

		System.out.println("Java is lazy and hasn't throw the error yet");

		if(candidate_number < 2) {
			return false;
		}
		else {
			int sqrt = (int) Math.sqrt(candidate_number);
			return test_primality_below_sqrt(candidate_number,sqrt);
		}

	}

	public static boolean test_primality_below_sqrt(int candidate_number, int
sqrt) {

		if(sqrt == 1) {
			return true;
		}
		else {
			if(candidate_number % sqrt == 0) {
				return false;
			}
			else {
				return test_primality_below_sqrt(candidate_number,sqrt-1);
			}
		}
	}

}
```

Strictness Points

Doug's program will not produce the same result as the program with 3 strictness points. Here is an example of an expression that will evaluate with different results by each interpreter:

(if0 (+ 1 2) (+ 4 5) (+ 7 8))

Doug's interpreter only has a strictness point around the id's. This program has no id's. This causes a problem because the condition in the if0 statement needs to evaluate to a number, or else the interpreter throws an error. The other lazy interpreter has a strictness point around the condition in the if0 statement to prevent this problem.

State

Lazy languages don't have mutable state, because it can cause different results depending on when things are evaluated. Lazy languages evaluate things when they are needed. Laziness relies on the fact that evaluation order doesn't matter. However, with mutation, you need to pay attention to the order in which operations are performed. Laziness and mutation don't work well together. One of the reasons why functional programming lends itself well to laziness is because its functions don't have side effects. Below is an example of how side effects and mutable state can cause problems when things are evaluated lazily. Without mutable state and side effects, the program should print "My name is Eric and I am 24 years old" twice. However because of mutation, it prints out "My name is Eric and I am 24 years old" the first time, and "My name is Ryan and I am 24 years old" the second time.

```
public class ChangeState {

		public static String name = "Eric";

		public static void main(String[] args) {

			int age = 24;
			printStatement(24);

			name = "Ryan";

			printStatement(24);


		}

		public static void printStatement(int age) {
			System.out.println("My name is "+ name+ " and I am "+age+" years
old.");
		}
```

```
}
```

Application

Laziness has many programming applications. The benefits of lazy evaluation include:

· First of all, it can save memory in the computer. The ability to create an infinite list is useful because we can have a “list” of number in a particular sequence without actually having to store the numbers. We only use resources when we require them. It prevents storage from being wasted on resources that aren’t being used.

· Performance increases by avoiding needless calculations, and error conditions in evaluating compound expressions

· The ability to construct potentially infinite data structures

· The ability to define control flow (structures) as abstractions instead of primitives

In computer windowing systems, the painting of information to the screen is driven by expose events, which drive the display code at the last possible moment. By doing this, windowing systems avoid computing unnecessary display content updates.

Another example of laziness in modern computer systems is copy-on-write page allocation or demand paging, where memory is allocated only when a value stored in that memory is changed.

Laziness can be useful for high performance scenarios. An example is the Unix mmap function, which provides demand driven loading of pages from disk, so that only those pages actually touched are loaded into memory, and unneeded memory is not allocated.

MATLAB implements copy on edit, where arrays which are copied have their actual memory storage replicated only when their content is changed, possibly leading to an out of memory error when updating an element afterwards instead of during the copy operation.