18: Streams

CS1101S: Programming Methodology

Martin Henz

October 24, 2012

Module Plan

- Wed, 24/10: Streams
- Fri, 26/10: Hari Raya Haji
- Wed, 31/10: Meta-circular evaluator (Part I)
- Fri, 2/11: Meta-circular evaluator (Part II)
- Wed, 7/11: Practical exam
- Fri, 9/11: Death Cube Contest
- Wed, 14/11: Java
- Fri, 16/11: Wrapping up CS1101S (party?)

The Basics

Remember Midterm Question 1

Represent E1 ? E2 : E3 as a function.

```
cond(E1, function() { return E2; },
         function() { return E3; })
```

where

```
function cond(x,y,z) {
    if (x) { return y(); } else { return z(); }
}
```

Delayed Evaluation

Main Idea

We delayed the evaluation of E2 and E3 until we had enough information to decide which one was needed.

Instrument of delay

Functions allow us to describe an activity without actually doing the activity.

Functions as Pickles

Like preserved vegetable or fruit, a function stores the activity and can be “opened” by applying it.

A Simple Example

```
function f(...) {
    var x = ...;
    return function() { return x + 10; };
}

// return y from computation
var y = f(...);

// two weeks later
var z = y();
```

Memoization with Lazy Evaluation

```
function memo_fun(fun) {
    var already_run = false;
    var result = undefined;
    return function() {
                if (! already_run) {
                    result = fun();
                    already_run = true;
                    return result;
                } else {
                    return result;
            }   };
}
```

Why don’t we always do this?

Nature of functions

Functions often have an *effect*, beyond the value they are computing

Side effects

When we are talking about lazy evaluation, these effects are called *side effects*

Pure functional programming

Programming without side-effects. All values can be memoized.

What does this function do?

```
function mystery(a, b) {
    function iter(count, sofar) {
        if (count > b) {
            return sofar;
        } else {
            if (prime(count)) {
                return iter(count + 1, count+sofar);
            } else {
                return iter(count + 1, sofar);
    }   }   }
    return iter(a, 0);
}
```

Sum of primes from a to b

```
function sum_primes(a, b) {
    function iter(count, sofar) {
        if (count > b) {
            return sofar;
        } else {
            if (prime(count)) {
                return iter(count + 1, count+sofar);
            } else {
                return iter(count + 1, sofar);
    }   }   }
    return iter(a, 0);
}
```

Also can?

```
function sum_primes(a, b) {
    return accumulate(function(x, y) {
                          return x + y;
                      },
                      filter(prime,
                             enum_list(a, b)));
}
```

Extreme example

```
head(tail(filter(prime,
                 enum_list(100, 1000000))));
```

What is wrong here?

Extreme example

```
head(tail(filter(prime,
                 enum_list(100, 1000000))));
```

What is wrong here?

We only need the first element of the list of 999901 numbers.

Idea of streams

Delayed lists

Our pairs contain a data item as head (as usual), but a function as tail that can be activated when needed.

Streams

A *stream* is either the empty list, or a pair whose tail is a nullary function that returns a stream.

Stream discipline

Like list discipline, now using streams.

Simple example

```
function ones_stream() {
    return pair(1, ones_stream);
}

var ones = ones_stream();
head(ones); // 1
head(tail(ones)()); // 1
head(tail(tail(ones)())()); // 1
```

Convenient function

```
function stream_tail(stream) {
    return tail(stream)();
}

var ones = ones_stream();
head(ones); // 1
head(stream_tail(ones));
head(stream_tail(stream_tail(ones)));
```

Streams are lazy lists

```
function stream_ref(s, n) {
    if (n === 0) {
        return head(s);
    } else {
        return stream_ref(stream_tail(s), n - 1);
    }
}
```

Everything still works

```
function stream_map(f, s) {
  if (is_empty_list(s)) {
    return [];
  } else {
    return pair(f(head(s)),
                function() {
                  return stream_map(
                           f, stream_tail(s));
                });
} }
```

Everything still works

```
function stream_filter(p, s) {
  if (is_empty_list(s)) {
      return [];
  } else if (p(head(s))) {
      return pair(head(s),
                  function() {
                      return stream_filter(
                                 p,stream_tail(s));
                  });
  } else {
      return stream_filter(p,
                           stream_tail(s));
} }
```

Extreme example

```
head(stream_tail(stream_filter(
                          prime,
                          enum_stream(100,
                                      1000000))));
```

General idea

Only compute what is needed. Be *lazy* about it!

More examples

```
function integers_from(n) {
    return pair(n,
                function() {
                    return integers_from(n + 1);
                });
}

var integers = integers_from(0);
head(integers);
head(stream_tail(integers));
head(stream_tail(stream_tail(integers)));
```

More examples

```
function divisible(x, y) {
    return x % y === 0;
}
no_fours =
stream_filter(function(x) {
                  return ! divisible(x, 4);
              },
              integers);
```

From streams to lists

```
function eval_stream(s, n) {
    if (n === 0) {
        return [];
    } else {
        return pair(head(s),
                    eval_stream(stream_tail(s),
                                n - 1));
    }
}
```

U still there?

```
stream_ref(no_fours,3);

stream_ref(no_fours,100);

eval_stream(no_fours,10);
```

Repeating sequence

Wanted

Stream containing 1, 2, 3, 1, 2, 3, 1, 2, 3,...

```
var rep123 =
pair(1,
     function() {
       return pair(2,
                   function() {
                     return pair(3,
                                 ???);
     });           });
```

Repeating sequence

Wanted

Stream containing 1, 2, 3, 1, 2, 3, 1, 2, 3,...

```
var rep123 =
pair(1,
     function() {
       return pair(2,
                   function() {
                     return pair(3,
                                 function() {
                                   return rep123;
                                 });
                   });
     });
```

More and more

Wanted

Stream containing 1, 1, 2, 1, 2, 3, 1, 2, 3, 4,...

```
function helper(a, b) {
    if (a > b) {
        return helper(1, 1 + b);
    } else {
        return pair(a,
                    function() {
                        return helper(a + 1, b);
                    });
    }
}
```

Stream Processing

Like lists except

- Wrap tail in function
- Use `stream_tail` instead of `tail`

Stream library

We need to write a stream library that provides stream versions of our list library functions: `stream_map`, `stream_filter` etc

Replace

Wanted

A function replace that creates a new stream by replacing in a given stream a particular value by another value.

Example

```
replace(more_and_more, 1, 0)
```

—> 0 0 2 0 2 3 0 2 3 4 0 2 3 4 5 ...

Replace

```
function replace(s, a, b) {
  return pair( (head(s) === a) ? b : head(s),
               function() {
                 return replace(stream_tail(s),
                                a, b);
               });
}
```

Challenge

Wanted

Write a function that when given a list returns a stream that repeats the list infinitely

Adding two streams

Wanted

Write a function that takes two streams and returns a stream that contains the pairwise sums

Example

adding
1 2 3 4 5 6...
1 1 2 2 3 3...
should return
2 3 5 6 8 9...

Adding streams

```
function add_streams(s1, s2) {
  if (is_empty_list(s1) {
    return s2;
  } else {
    return pair(head(s1) + head(s2),
                function() {
                  return add_streams(
                           stream_tail(s1),
                           stream_tail(s2));
                });
} }
```

Another example

```
var wat =
pair(0,
  function() {
    return pair(1,
                function() {
                  return add_streams(
                           wat,
                           stream_tail(wat));
                });
  });
```

Or another fib?

```
function fibgen(a, b) {
    return pair(a,
                function() {
                    return fibgen(b, a + b);
                });
}
var fibs = fibgen(0, 1);
```

Integers Revisited

```
var ones = pair(1, function() { return ones; });

var integers =
pair(1, function() {
          return add_streams(ones, integers);
        });
```

Iteration revisited

```
function improve(guess,x) {
    return average(guess,x / guess);
}
function sqrt_iter(guess,x) {
    if (good_enough(guess,x))
        return guess;
    else
        return sqrt_iter(improve(guess,x), x);
}
function sqrt(x) {
    return sqrt_iter(1.0,x);
}
```

Using streams for iteration

```
function sqrt_stream(x) {
  var guesses =
  pair(1.0,
       function() {
         return stream_map(
                  function(guess) {
                    return sqrt_improve(guess,x);
                  },
                  guesses)
       });
  return guesses;
}
```

Using streams for iteration

```
eval_stream(sqrt_stream(2), 6)
// 1
// 1.5
// 1.416666666666
// 1.414215686274
// 1.414213562374
// 1.414213562373
```