1 Source Specification

Predefined in global env: `undefined`, `NaN`, `Infinity`

fn	description
`Math.*`	Functions dealing with Math. (eg. `Math.pow`, `Math.sqrt`, `Math.E`, `Math.PI`)
`parseInt(string)`	Takes a string and parses it into an integer.
`equal(x,y)`	Returns `true` if `x` and `y` have the same structure, and the nodes are all identical. `false` otherwise.
`is_number(x)`	Returns `true` if `x` is a number, `false` otherwise.

1.1 List

fn	description	I/R	Time	Space
`is_pair(x)`	Returns `true` if `x` is a pair, `false` otherwise.			
`is_list(x)`	Returns `true` if `x` is a pair, `false` otherwise.			
`is_empty_list(x)`	Returns `true` if `x` is an empty list, false otherwise.			
`pair(x,y)`	Returns a pair of `x` and `y`.			
`head(p)`	Returns the head of a pair.			
`tail(p)`	Returns the tail of a pair.			
`set_head(p, n)`	Sets the head of pair `p` to `n`.			
`set_tail(p, n)`	Sets the tail of pair `p` to `n`.			
`list(x1,x2,...,xn)`	Returns a list of `n` elements.		O(n)	O(n)
`length(xs)`	Returns the length of the list `xs`.	I	O(n)	O(1)
`map(f,xs)`	Returns a list with each element having `f` applied to it.	R	O(n)	O(n)
`build_list(n,f)`	Make a list of elements with unary function `f` applied to numbers `0` to `n-1`.	R	O(n)	O(n)
`for_each(f,xs)`	map, but use only for side effects. Returns `true`.	I	O(n)	O(1)
`list_to_string(xs)`	Returns string representation of list `xs`.			
`reverse(xs)`	Returns list `xs` in reverse order.	I	O(n)	O(n)
`append(xs,ys)`	Returns a list with list `ys` appended to list `xs`.	R	O(n)	O(n)
`member(x, xs)`	Returns first postfix sublist whose head is identical to `x`.	I	O(n)	O(1)
`remove(x, xs)`	Returns a list by removing the first item identical to `x`.	R	O(n)	O(n)
`remove_all(x,xs)`	Returns a list by removing all elements identical to `x`.	R	O(n)	O(n)
`filter(pred,xs)`	Returns a list containing only elements in `xs` for which `pred` returns `true`.	R	O(n)	O(n)
`enum_list(start,end)`	`enum_list(0, 4)` becomes `[0, [1, [2, [3, [4, []]]]]]`.	R	O(n)	O(n)
`list_ref(xs, n)`	Returns the element in `xs` at position `n`.	I	O(n)	O(1)
`accumulate(op, initial, xs)`	`op(x1, op(x2, op(x3, ... op(xn, initial))))`.	R	O(n)	O(n)

1.2 Stream

fn	description	Lazy?
`stream_tail(s)`	returns result of applying nullary function at tail.	Y
`is_stream(s)`	returns `true` if `s` is a stream, `false` otherwise.	N
`stream(x1,x2,...,xn)`	Returns a stream with `n` elements.	N
`list_to_stream(xs)`	Transforms a list into a stream.	Y
`stream_to_list(s)`	Transform a stream into a list.	N
`stream_length(s)`	Returns the length of the stream `s`.	N
`stream_map(f,s)`	Returns a stream from stream `s` by element-wise application of `f`.	Y
`build_stream(n,f)`	Makes a stream of `n` elements, by applying the unary function `f` to numbers `0` to `n-1`.	Y
`stream_for_each(f,s)`	Applies f to every element of the stream `s`, and returns `true`.	N
`stream_reverse(s)`	Returns a finite stream `s` in reverse order.	N
`stream_append(xs,ys)`	Returns a stream that results from appending `ys` to `xs`.	Y
`stream_member(x,s)`	Returns first postfix substream whose head is identical to `x`.	P
`stream_remove(x,s)`	Returns a stream that results from removing the first element identical to `x` from stream `s`.	Y
`stream_remove_all(x,s)`	Returns a stream that results from removing all elements identical to `x` from stream `s`.	Y
`stream_filter(pred,s)`	Returns a stream that contains only elements which return `true` on unary predicate `pred`.	Y
`enum_stream(start,end)`	Similar to `enum_list`.	Y
`integers_from(n)`	Constructs an infinite stream of integers starting at `n`.	Y
`eval_stream(s,n)`	Constructs a list of the first `n` elements of s.	P
`stream_ref(s,n)`	Returns the element of stream `s` at position `n`.	P

1.3 Metacircular

fn	description
`evaluate(stmt,env)`	Classifies `stmt` and directs the evaluation. Handles primitive forms, special forms and combinations.
`apply(fun, args)`	Primitive functions: calls `apply_primitive_function`. Compound functions: sequentially eval exps in new env created.
`lookup_variable_value(var,env)`	returns value bound to the symbol `var`, or signals an error if unbound.
`define_variable(var,value,env)`	adds to the first frame of `env` a binding of `var` to `value`.
`extend_environment(variables,values,base_env)`	returns a new environment, with a new frame extended from `base_env`, with the corresponding `variables` and `values`.
`set_variable_value(var,value,env)`	changes the binding of `var` in `env` to `value`, signals error if unbound.

2 Mutations

```
function mutable_reverse(xs) {
  function helper(prev,xs) {
    return prev;
  } else {
    var rest = tail(xs);
    set_tail(xs, prev);
    return helper(xs,rest);
  }

  return helper([],xs);
}

function mutable_reverse(xs) {
  if (is_empty_list(xs) ||
      is_empty_list(tail(xs))) {
    return xs;
  } else {
    var temp = mutable_reverse(tail(xs));
    set_tail(tail(xs), xs);
    set_tail(xs,[]);
    return temp;
  }
}
function make_circular_copy(xs) {
  function helper(rem,front_ptr) {
    if (is_empty_list(rem)) {
      return front_ptr;
    } else {
      return pair(head(rem),
                  helper(tail(rem),
                         front_ptr));
    }
  }

  if (is_empty_list(xs)) {
    return [];
  } else {
    var ys = pair(head(xs), []);
    set_tail(y, helper(tail(xs),ys));
    return ys;
  }
}
function mergeB(xs,ys) {
  if (is_empty_list(xs) && is_empty_list(ys)) {
    return [];
  } else if (is_empty_list(xs) ||
             head(xs) <= head(ys)) {
    set_tail(ys, mergeB(xs, tail(ys)));
    return ys;
  } else if (is_empty_list(ys) ||
             head(xs) >= head(ys)) {
    set_tail(xs, mergeB(tail(xs), ys));
    return xs;
  }
}
```

3 Permutations and Combinations

3.1 permutations

```
function permutations(xs) {
  if (is_empty_list(xs)) {
    return list([]);
  } else {
    return accumulate(function(e, acc) {
      return append(map(function(x) {
        return pair(e, x);
      }, permutations(remove(e, xs))), acc);
    }, [], xs);
  }
}
```

3.2 n_permutations

```
function n_permutations(xs, n) {
  if(n === 0) {
    return list([]);
  } else {
    return accumulate(function(e, acc) {
      return append(
        map(function(x) {
          return pair(e, x);
        }, n_permutations(remove(e, xs),
                          n - 1)),
        acc);
    }, [], xs);
  }
}
```

3.3 n_combinations

```
function n_combinations(xs, n) {
  if (n === 0) {
    return list([]);
  } else if (is_empty_list(xs)) {
    return [];
  } else {
    return append(
      map(function(e) {
        return pair(head(xs), e);
      }, n_combinations(tail(xs), n-1)),
      n_combinations(tail(xs), n));
  }
}
```

4 OOP

```
function Vector2D (x,y) {
  this.x = x;
  this.y = y;
}

Vector2D.prototype.length = function() {
  return Math.sqrt(this.x * this.x +
                   this.y * this.y);
}

function Thrust (x,y, tag) {
  Vector2D.call(this,x,y);
  this.tag = tag;
}

Thrust.Inherits(Vector2D);
```

5 Streams

5.1 Recursively defined streams

```
function fibgen(a,b) {
  return pair(a,b function() {
    return fibgen(b, a+b);
  });
}

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

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

// Visualization:
ones:     1 1 1 1 1 1
integers:   1 2 3 4 5
--------------------
integers: 1 2 3 4 5 6
```

5.2 Stream of primes

```
function sieve(s) {
  return pair(head(s), function() {
    return sieve(stream_filter(function() {
      return !is_divisible(x,head(s));
    }, stream_tail(s)));
  });
}

var primes = sieve(integers_from(2));
```

5.3 Iterations with streams

```
function improve_guess(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);
}

function sqrt_stream(x) {
  var guesses = pair(1, function() {
    return stream_map(function(guess) {
      return improve(guess,x);
    }, guesses);
  });

  return guesses;
}
```

5.4 Interleave

```
function interleave(s1,s2) {
  return pair(head(s1), function() {
    return pair(head(s2), function() {
      return interleave(stream_tail(s1),
                        stream_tail(s2));
    });
  });
}
```

5.5 Cartesian Product

```
function pairs(s1,s2){
  if (is_empty_list(s1) || is_empty_list(s2)) {
    return [];
  } else {
    return pair(
      pair(head(s1), head(s2)),
      function() {
        return interleave(
          stream_map(function(x) {
            return pair(head(s1),x);
          }, stream_tail(s2)),
          pairs(stream_tail(s1), s2));
      });
  }
}
```

6 Misc

6.1 Towers of Hanoi

```
function hanoi(disks, source,dest,aux) {
  if (disks === 0) {
    return [];
  } else {
    hanoi(disks-1,source,aux,dest);
    display("Move disk from " +
            source + " to " + dest);
    hanoi(disks-1,aux,dest,source);
  }
}
```

6.2 Count Change

```
// denoms is a list of coin denominations:
// eg. list(50,20,10,5)
function count_change(amt, denoms) {
  if (is_empty_list(denoms) || amt < 0) {
    return 0;
  } else if (amt === 0) {
    return 1;
  } else {
    return count_change(
      amt,
      tail(denoms)) +
      count_change(amt-head(denoms),
                   denoms);
  }
}
```

6.3 Power set

```
function power_set(xs) {
  if (is_empty_list(xs)) {
```

```
    return list([]);
  } else {
    // Either you pick the number,
    // or you don't
    var without_head = power_set(tail(xs));
    var use_head = map(function(l) {
      return pair(head(xs),l);
    }, without_head);

    return append(use_head,without_head);
  }
}
```

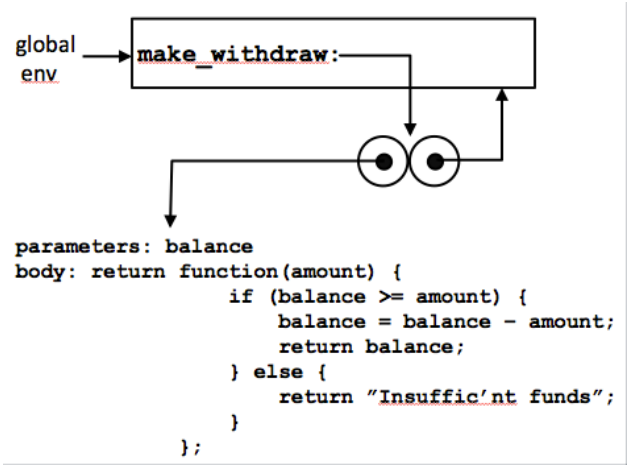
7 Memoization

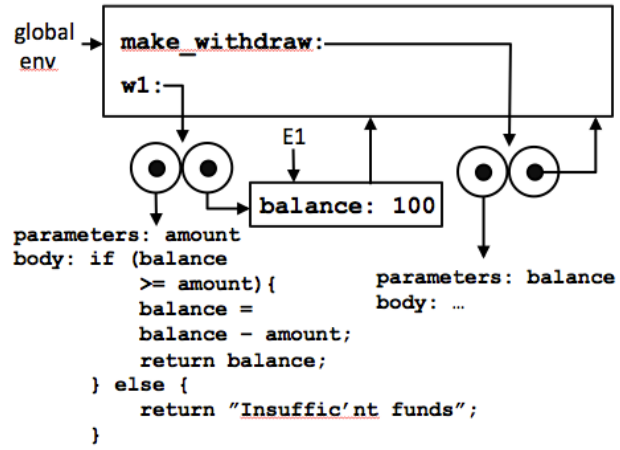
```
function memo_fib(n) {
  var res = {};
  res[1]=0;
  res[2]=1;
  function fib(n) {
    if (res[n] !== undefined) {
      return res[n];
    } else {
      res[n] = fib(n-2) + fib(n-1);
      return res[n];
    }
  }

  return fib(n);
}
```

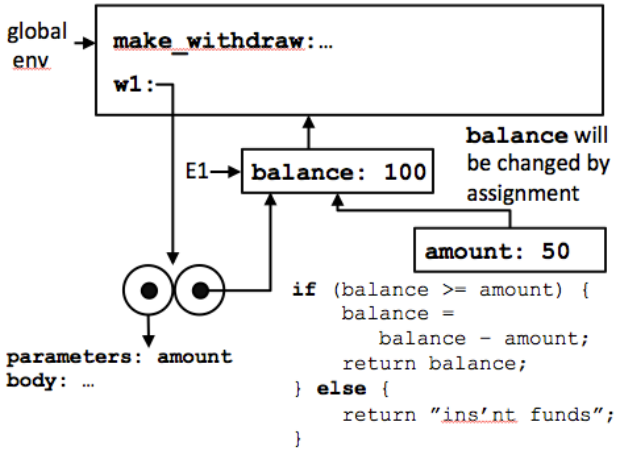
8 Environment Model

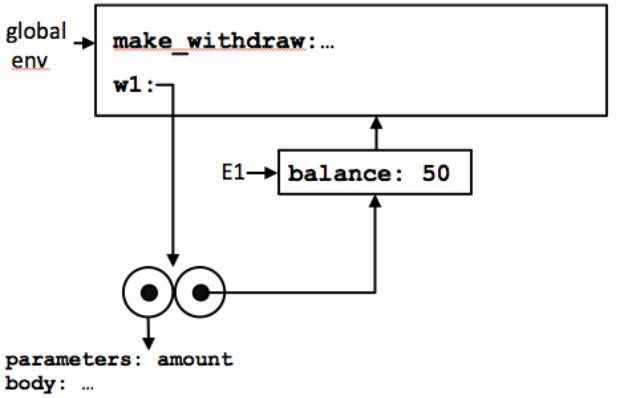
```
function make_withdraw(balance) {
  return function(amount) {
    if (balance >= amount) {
      balance = balance - amount;
      return balance;
    } else {
      return "Insufficient Funds";
    }
  };
}
```

```
// Pic 1
var w = make_withdraw(100);      // Pic 2
w(50);                           // Pic 3
// Pic 4
```









9 Metacircular Interpreter

9.1 Reverse Application Order

```
function list_of_values(exps.env) {
  if (no_operands(exps)) {
    return [];
  } else {
    var r = list_of_values(rest_operands(exps),
                           env);
    return pair(evaluate(first_operand(exps),
                         env),
                r);
  }
```

```
}
```

9.2 Thunking

```
function list_of_values(exps,env) {
  if (no_operands(exps)) {
    return [];
  } else {
    return pair(
      make_thunk(first_operand(exps), env),
      list_of_values(rest_operands(exps), env)
    );
  }
}

function make_thunk(expr,env) {
  return {
    tag: "thunk",
    expression: expr,
    environment: env
  };
}

function force(v) {
  if (is_thunk(v)) {
    return force(evaluate(thunk_expression(v),
                          thunk_environment(v)));
  } else {
    return v;
  }
}

function lookup_variable_value(variable,env) {
  function env_loop(env){
    if (is_empty_environment(env)) {
      error("Unbound Variable");
    } else if (has_binding_in_frame(
      variable,
      first_frame(env))) {
      var val = force(first_frame(env)[variable]);
      first_frame(env)[variable] = val;
      return val;
    } else {
      return env_loop(enclosing_environment(env));
    }
  }

  var val = env_loop(env);
  return val;
}
```