

National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

R10 Streams

$\text{stream} := \text{null} \mid \text{pair}(\text{item}, () \Rightarrow \text{stream})$
 \rightarrow saves space / time by requesting as needed.
 e.g. video streaming.

$\text{pair}(\text{now}, () \Rightarrow \text{future})$

Stream Processing Functions

In addition to the [Stream Processing functions](#) of [Source §3](#) we further define the following functions and values:

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

function mul_streams(s1, s2) {
  if (is_null(s1)) {
    return s2;
  } else if (is_null(s2)) {
    return s1;
  } else {
    return pair(head(s1) * head(s2),
      () => mul_streams(stream_tail(s1),
        stream_tail(s2)));
  }
}
```

$S_1 \rightarrow x_1, x_2, \dots$
 $S_2 \rightarrow y_1, y_2, \dots$
 $S_1 + S_2 \rightarrow x_1 + y_1, x_2 + y_2, \dots$
 $\text{add_streams}(S_1, S_2) \rightarrow \text{pair}(\text{head}(S_1) + \text{head}(S_2), () \Rightarrow \text{add_streams}(\text{stream_tail}(S_1), \text{stream_tail}(S_2)))$

```
function scale_stream(s, f) {
  return stream_map(x => x * f, s);
}
```

```
const integers = integers_from(1);
const ones = pair(1, () => ones);
```

$S \rightarrow x_1, x_2, \dots$
 $\text{Stream_map}(f, S) \rightarrow f(x_1), f(x_2), \dots$
 $\text{Stream_map}(f, S) \rightarrow \text{pair}(f(\text{head}(S)), () \Rightarrow \text{Stream_map}(f, \text{stream_tail}(S)))$

f Stream-map (f, s)

base case:

$\hookrightarrow \text{pair}(f(\text{head}(s)), () \Rightarrow \text{Stream_map}(f, \text{stream_tail}(s)))$

Problems:*debug - remove stuff*

1. In order to take a closer look at delayed evaluation, we will use the function `display` which displays its argument and returns it. What is displayed in response to evaluating the following sequence?

```
const x = stream_map(display, enum_stream(0, 10)); → 0
stream_ref(x, 3); → 1, 2, 3.
stream_ref(x, 5); → 1, 2, 3, 4, 5.
```

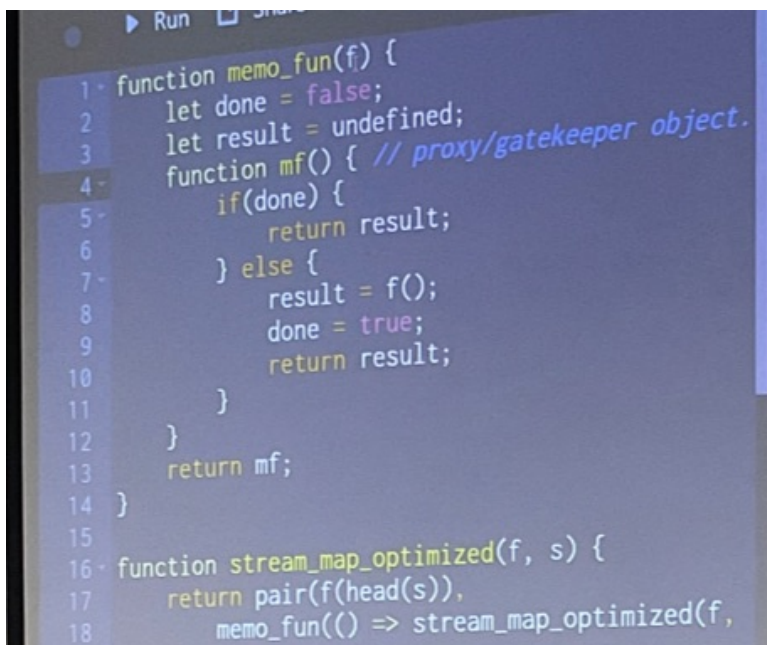
Consider the optimized version of `stream_map` covered in SICP JS, using the function `memo_fun` used in Lecture L10:

```
function stream_map_optimized(f, s) {
  return is_null(s)
    ? null
    : pair(f(head(s)),
          memo_fun( () => stream_map_optimized(
                        f, stream_tail(s)) ));
}
```

What will be displayed when evaluating the following sequence?

```
const x = stream_map_optimized(display, enum_stream(0, 10));
stream_ref(x, 3);
stream_ref(x, 5);
```

2. Consider the function `zip_stream` from Path P10 that takes as parameters two infinite streams and returns an infinite stream in which the elements of the given streams are interleaved. For example if s_1 is the infinite stream 1, 11, 111,... and s_2 is the infinite stream 2, 22, 222,... then `zip_stream(s_1, s_2)` returns the stream 1, 2, 11, 22, 111, 222,.... Declare the function `zip_list_of_streams` that takes a non-empty list of infinite streams as argument and zips up them all, in the given order. For example, if s_1 and s_2 are as above and s_3 is the infinite stream 3, 33, 333,..., then `zip(list(s_1, s_2, s_3))` should return the stream 1, 2, 3, 11, 22, 33,....
3. Consider the function `partial_sums` from Path P10 that takes as parameter an infinite stream of numbers s and returns the infinite stream whose elements are $s_0, s_0+s_1, s_0+s_1+s_2, \dots$. For example, `partial_sums(integers)` should be the stream 1, 3, 6, 10, 15, Declare the function `partial_sums` using the function `add_streams` above. Does your function also work on finite streams as argument? How about the empty stream?



```
1 function memo_fun(f) {
2   let done = false;
3   let result = undefined;
4   function mf() { // proxy/gatekeeper object.
5     if(done) {
6       return result;
7     } else {
8       result = f();
9       done = true;
10      return result;
11    }
12  }
13  return mf;
14 }
15
16 function stream_map_optimized(f, s) {
17   return pair(f(head(s)),
18             memo_fun(() => stream_map_optimized(f,
```

Problems:

1. In order to take a closer look at delayed evaluation, we will use the function `display` which displays its argument and returns it. What is displayed in response to evaluating the following sequence?

```
const x = stream_map(display, enum_stream(0, 10));
stream_ref(x, 3);
stream_ref(x, 5);
```

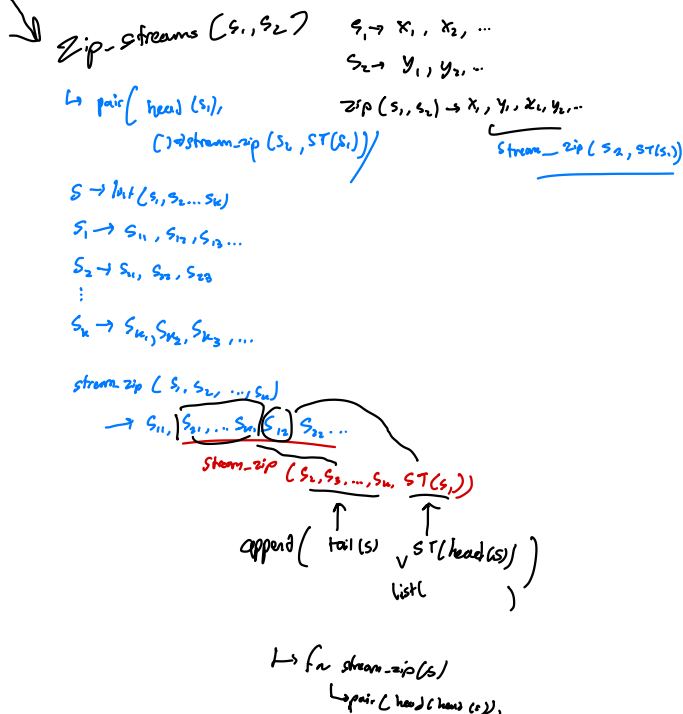
Consider the optimized version of `stream_map` covered in SICP JS, using the function `memo_fun` used in Lecture L10:

```
function stream_map_optimized(f, s) {
  return is_null(s)
    ? null
    : pair(f(head(s)),
          memo_fun( () => stream_map_optimized(
                      f, stream_tail(s)) ));
}
```

What will be displayed when evaluating the following sequence?

```
const x = stream_map_optimized(display, enum_stream(0, 10));
stream_ref(x, 3);
stream_ref(x, 5);
```

2. Consider the function `zip_stream` from Path P10 that takes as parameters two infinite streams and returns an infinite stream in which the elements of the given streams are interleaved. For example if s_1 is the infinite stream 1, 11, 111,... and s_2 is the infinite stream 2, 22, 222,... then `zip_stream(s_1, s_2)` returns the stream 1, 2, 11, 22, 111, 222,... Declare the function `zip_list_of_streams` that takes a non-empty list of infinite streams as argument and zips up them all, in the given order. For example, if s_1 and s_2 are as above and s_3 is the infinite stream 3, 33, 333,..., then `zip(list(s_1, s_2, s_3))` should return the stream 1, 2, 3, 11, 22, 33,....
3. Consider the function `partial_sums` from Path P10 that takes as parameter an infinite stream of numbers s and returns the infinite stream whose elements are $s_0, s_0 + s_1, s_0 + s_1 + s_2, \dots$. For example, `partial_sums(integers)` should be the stream 1, 3, 6, 10, 15, Declare the function `partial_sums` using the function `add_streams` above. Does your function also work on finite streams as argument? How about the empty stream?



$\text{fun stream_zip}(s)$
 $\text{curr} \rightarrow \text{zip}(\text{tail}(s), \text{head}(s) \text{ tail}(s))$
 $\text{curr} \rightarrow \text{zip}$
 $S \rightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{3} \rightarrow \dots \rightarrow \boxed{n}$
 $\text{fn helper}(curr, s)$
 $\text{if } (\text{is_null } curr)$
 $\quad \rightarrow \text{const } \text{failed} = \text{map}(s, s)$
 $\quad \rightarrow \text{helper}(\text{failed}, \text{failed})$
 else
 $\quad \rightarrow \text{pair}(\text{head } curr),$
 $\quad \quad () \Rightarrow \text{helper}(\text{tail } curr, s)$
 $\rightarrow \text{helper}(s, s)$

2.
 $\text{prev} \rightarrow 0, x_1, \dots$
 $S \rightarrow x_1, x_2, \dots$
 $\text{ps}(s) \rightarrow x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots$
 $x_1 + x_2 \rightarrow x_3$
 x_k
 $\sum_{i=1}^{k-1} x_i$
 $\sum_{i=1}^k x_i$

$\text{fn ps}(s)$
 $\text{fn helper}(\text{prev}, s)$
 $\quad \rightarrow \text{pair}(\text{head}(s) + \text{prev},$
 $\quad \quad () \Rightarrow \text{helper}(\text{head}(s) + \text{prev}, \text{stream_tail}(s)))$

$() \Rightarrow \text{helper}(\text{head}(s) + \text{prev}, \text{stream_tail}(s))$
 $\rightarrow \text{helper}(0, s)$
 $\text{stream: pair}(\text{now}, () \Rightarrow \text{future})$
 $\text{const } (0) \rightarrow \text{pair}(1, () \Rightarrow \text{add_streams}(\text{stream_tail}(fibs), \text{fibs}))$
 $\text{1 1 2 3 5 8 13 ... = stream_tail}(fibs)$
 $\text{0 1 1 2 3 5 8 ... = fibs}$
 $\text{0 1 1 2 3 5 8 13 21 ... = fibs}$
 fck-g.

$$P_k^s = P_{k-1}^s + S_n$$

$\text{fn partial_sum}(s)$
 $\quad \rightarrow \text{pair}(\text{head}(s),$
 $\quad \quad () \Rightarrow \text{add_streams}(\text{partial_sum}(s), \text{stream_tail}(s)))$

```

34 // // stream_zip
35
36 // // x1
37
38
39 function add_streams(s1, s2) {
40   return pair(head(s1) + head(s2),
41     () => add_streams(stream_tail(s1), stream_tail(s2))
42 )
43
44 function partial_sum(s) {
45   return pair(head(s),
46     () => add_streams(stream_tail(s),
47       partial_sum(s)))
48 }
49
50 const ones = pair(1, () => ones);
51 eval_stream(partial_sum(partial_sum(ones)), 10);

```