

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

S11-in-class

Problems:

Stream of pairs

1. Given a stream s the following function returns a stream of pairs of elements from s :

```
function stream_pairs(s) {
  return is_null(s)
    ? null
    : stream_append(
      stream_map(
        sn => pair(head(s), sn),
        stream_tail(s)),
      stream_pairs(stream_tail(s)));
}
```

base case won't occur for inf. stream.

pair 2 lists together

change all elements of list(2, 3, 4, 5) to list(pair(1, 2), pair(1, 3), pair(1, 4), pair(1, 5))

do it again with list(2, 3, 4, 5).

- (a) Suppose that `ints` is the (finite) stream 1, 2, 3, 4, 5. What is `stream_pairs(ints)`?
~~list(pair(1, 2), pair(1, 3), pair(1, 4), pair(1, 5), pair(2, 3), pair(2, 4), pair(2, 5), pair(3, 4), pair(3, 5), pair(4, 5))~~
- (b) Give the clearest explanation that you can of how `stream_pairs` works.
~~all possibilities of SP_2 , no repeat.~~
- (c) Suppose that `integers` is the infinite stream of positive integers. What is the result of evaluating

`const s2 = stream_pairs(integers);` *infinite recursion?*

Hint: Note that the function `stream_append` is defined in Source §3 as follows:

```
function stream_append(xs, ys) {
  return is_null(xs)
    ? ys
    : pair(head(xs),
      () => stream_append(stream_tail(xs),
                           ys));
}
```

- (d) Consider the following variant of `stream_append`, called `stream_append_pickle` and the function `stream_pairs2` which makes use of it.

```

function stream_append_pickle(xs, ys) {
  return is_null(xs)
    ? ys() second stream
    : pair(head(xs),
           () => stream_append_pickle(stream_tail(xs),
                                       ys));
}
call it if xs is null

```

pair (n. fn &v, future => tail xs)

```

function stream_pairs2(s) {
  return is_null(s)
    ? null
    : stream_append_pickle(
      stream_map(
        s => pair(head(s), s),
        stream_tail(s)),
      () => stream_pairs2(stream_tail(s)));
}
base

```

pickle implementation

```

const s2 = stream_pairs2(integers);

```

Why does the function `stream_pairs2` solve the problem that arose in the previous question? *It is a lazy func.*

- (e) What are the first few elements of `stream_pairs2(integers)`? Can you suggest a modification of `stream_pairs2` that would be more appropriate in dealing with infinite streams?

Multiplying series

2. Multiplying two series is a lot like multiplying two multi-digit numbers, but starting with the left-most digit, instead of the right-most.

For example:

```

      11111
    x 12321
    -----
    11111
   22222
  33333
 22222
 11111
  -----
136898631

```

s1 x head(s2)
s1 x head(tail(s2))

interleave-stream-append(s1, s2)
ret. is-null(s1)

? s2

: pair(head(s1),

() => interleave-stream-append(s2, stream-tail(s1));

Now imagine that there can be an infinite number of digits, i.e., each of these is a (possibly infinite) series. (Remember that because each “digit” is in fact a term in the series, it can become arbitrarily large, without carrying, as in ordinary multiplication.)

Using this idea, complete the definition of the following function, which multiplies two series:

infinite = stream

```

function mul_series(s1, s2) {
  return pair(<E1>,
    () => add_series(<E2>, <E3>));
}

```

To test your function, demonstrate that the product of S_1 (from Problem 3) and S_1 is S_2 . What is the coefficient of x^{10} in the product of S_2 and S_2 ? Turn in your definition of `mul_series`. (Optional: Give a general formula for the coefficient of x^n in the product of S_2 and S_2 .)

John

```

func add-series (s1, s2) {
  ret = is-null (s1)
    ? s2
    : is-null (s2)
    ? s1
    : pair (head (s1) + head (s2),
      () => add-series (tail (s1) (), tail (s2) ());
}

```

```

func mult-series (s1, s2) {
  if (!is-null (s1) && !is-null (s2)) {
    ret = pair (head (s1) * head (s2),
      () => add-series (
        scale-series { stream-map (x => x * head (s2),
          tail (s1) ()),
          mul-series (s1, tail (s2) ());
    );
  }
  ret = null;
}

```

mul(s1, s2).

s1:

1	1	1	1	1	...
---	---	---	---	---	-----

1st.

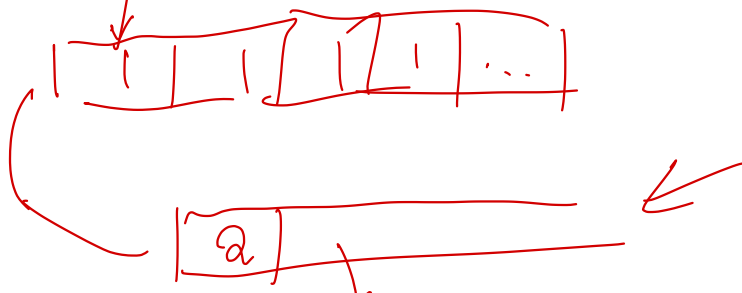
1	1	2	7
---	---	---	---

s2:

1	2	3	2	1
---	---	---	---	---

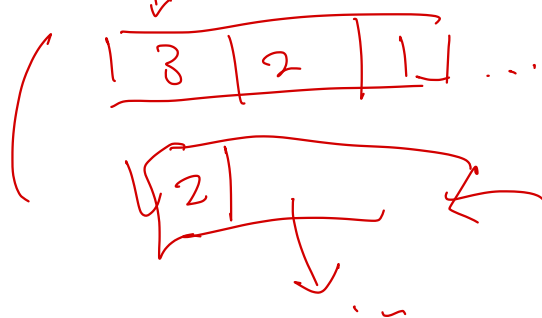
2nd.

add



3rd.

add



1 3 6 8 ...

s1:

1	2	3	2	1
---	---	---	---	---

s2:

1	1	1	1	1
---	---	---	---	---

s1:

1	1	1	1	1
---	---	---	---	---

s2:

2	3	2	1
---	---	---	---