CS 61A Discussion 10

Streams (and Interpreters)

Albert Xu

Kaavya Shah

Slides: albertxu.xyz/teaching/cs61a/

*part of these slides sourced from Jemmy and I's Final Review Fall '18

Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.
- This rest value is stored in something called a promise, which is essentially all the data of the rest!

Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.
- This rest value is stored in something called a promise, which is essentially all the data of the rest!
- A promise stored in the rest is not evaluated until we call cdr-stream on the stream.

Streams

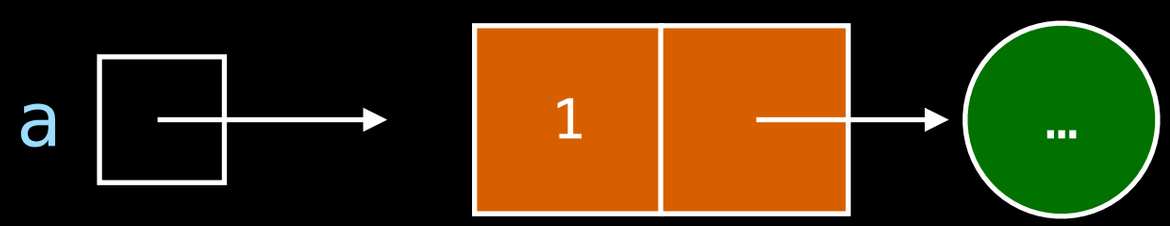
- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

```
>>> (define a (cons-stream 1 (cons-stream 2 nil)))
```

Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

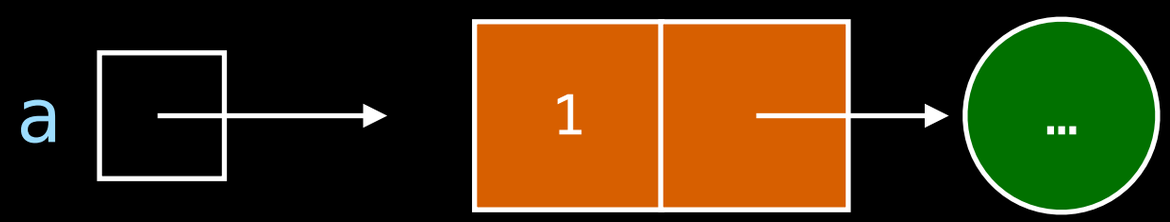
```
>>> (define a (cons-stream 1 (cons-stream 2 nil)))
```



Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

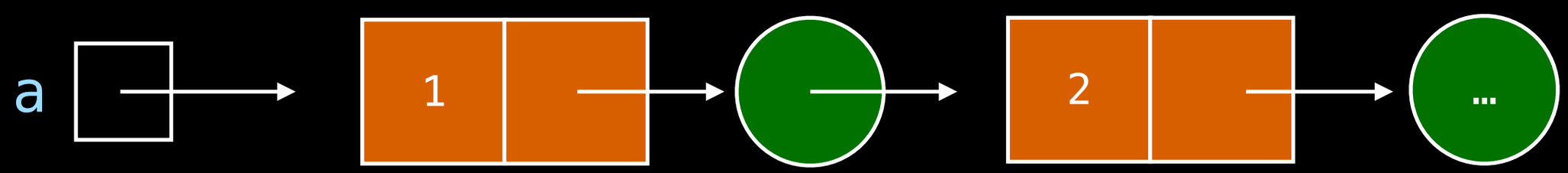
```
>>> (define a (cons-stream 1 (cons-stream 2 nil)))
```



Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

```
>>> (define a (cons-stream 1 (cons-stream 2 nil)))
```

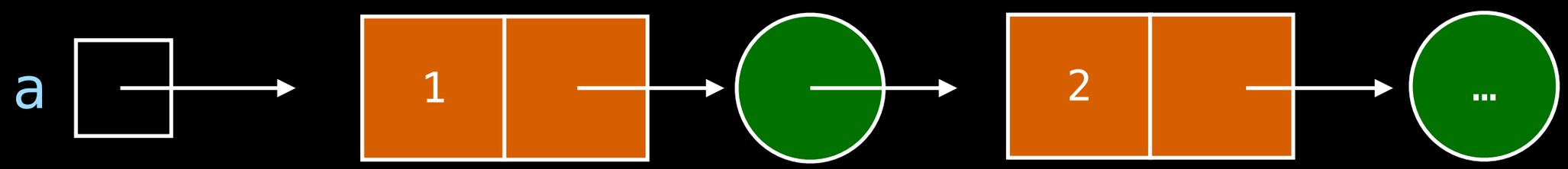



```
>>> (cdr-stream a)
(2 . #[promise(not forced)])
```

Streams

- Streams are lazy linked lists - the first element of each pair is calculated, but the rest is not calculated until we explicitly ask for it.

```
>>> (define a (cons-stream 1 (cons-stream 2 nil)))
```




```
>>> (cdr-stream a)
(2 . #[promise(not forced)])
```

Takeaway: notice how calling cdr-stream on a stream actually mutates that stream!

Why Streams?

- Streams can represent infinite sequences!

Why Streams?

- Streams can represent infinite sequences!

```
>>> (define (naturals n) (cons-stream n (naturals (+ n 1))))
```

Why Streams?

- Streams can represent infinite sequences!

```
>>> (define (naturals n) (cons-stream n (naturals (+ n 1))))
>>> (naturals 1)
(1 #[promise (nf)])
```

Why Streams?

- Streams can represent infinite sequences!

```
>>> (define (naturals n) (cons-stream n (naturals (+ n 1))))
>>> (naturals 1)
(1 #[promise (nf)])
>>> (cdr-stream (naturals 1))
(2 #[promise (nf)])
```

Why Streams?

- Streams can represent infinite sequences!

```
>>> (define (naturals n) (cons-stream n (naturals (+ n 1))))

            >>> (naturals 1)
            (1 #[promise (nf)])
            >>> (cdr-stream (naturals 1))
            (2 #[promise (nf)])
```

- So why doesn’t Python have streams?

Why Streams?

- Streams can represent infinite sequences!

```
>>> (define (naturals n) (cons-stream n (naturals (+ n 1))))

            >>> (naturals 1)
            (1 #[promise (nf)])
            >>> (cdr-stream (naturals 1))
            (2 #[promise (nf)])
```

- So why doesn't Python have streams?
    - Because generators can represent infinite sequences too!

Questions

1. How does a stream differ from a list in Scheme?

2. What arguments does `cons-stream` take in, and how does it differ from `cons`?

3. What's so special about `cdr-stream` - what's the difference between it and `cdr`?

Review Questions

1. How does a stream differ from a list in Scheme?

   The rest of a stream is only computed when we first access it, xa in a Scheme linked list we compute all the elements when it's created. This allows us to create infinite streams!

2. What arguments does `cons-stream` take in, and how does it differ from `cons`?

   Just like `cons`, `cons-stream` takes in a 1) a first element and 2) a rest which is another stream or nil, but instead of evaluating the second operand, it stores it as data in the promise!

Review Questions

3. What's so special about `cdr-stream` - what's the difference between it and `cdr`?

   On the first call to `cdr-stream` on that stream, the expression that was unevaluated and stored upon construction is now evaluated to return the rest of the stream. This **value is stored** so that on subsequent calls to `cdr-stream`, the rest no longer needs to be computed.

Review Questions

3. What's so special about `cdr-stream` - what's the difference between it and `cdr`?

On the first call to `cdr-stream` on that stream, the expression that was unevaluated and stored upon construction is now evaluated to return the rest of the stream. This **value is stored** so that on subsequent calls to `cdr-stream`, the rest no longer needs to be computed.

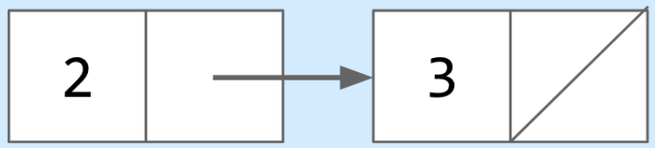
Takeaway: each element in the stream is only computed once, and anytime in the future we look up that element we use that stored value

Lists

```
scm> (define a (cons 1 (cons 2
(cons 3 nil))))
```

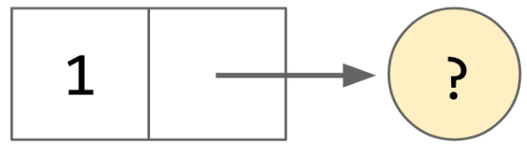

```
scm> (car a)
1

scm> (cdr a)
```



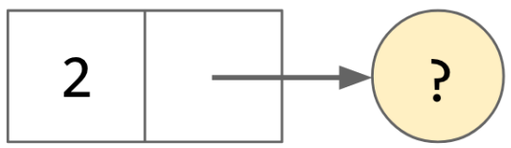
Streams

```
scm> (define a (cons-stream 1
(cons-stream 2 (cons-stream 3 nil))))
```




```
scm> (car a)
1

scm> (cdr-stream a)
```



Thanks for coming.

Have a great rest of your week! :)

Attendance: links.cs61a.org/albert-disc
Slides: albertxu.xyz/teaching/cs61a/