

Recitation 1

Scan

1.1 Announcements

- *SkylineLab* has been released, and is due **Friday afternoon**. It's worth 125 points.
- *BignumLab* will be released on Friday.

1.2 What is scan?

In the SEQUENCE library, there is a symmetry among certain aggregation functions:

Sequential	Parallel
iterate	reduce
iteratePrefixes	scan
iteratePrefixesIncl	scanIncl

We can see this symmetry in their types...

iterate	$(\beta * \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha \text{ seq} \rightarrow \beta$
reduce	$(\alpha * \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \text{ seq} \rightarrow \alpha$
iteratePrefixes	$(\beta * \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha \text{ seq} \rightarrow \beta \text{ seq} * \beta$
scan	$(\alpha * \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \text{ seq} \rightarrow \alpha \text{ seq} * \alpha$
iteratePrefixesIncl	$(\beta * \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha \text{ seq} \rightarrow \beta \text{ seq}$
scanIncl	$(\alpha * \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \text{ seq} \rightarrow \alpha \text{ seq}$

...as well as their output behavior: each of the parallel functions has output identical to its sequential analog under the condition that the first two arguments are an *associative function* and a corresponding *identity*, respectively.

Definition 1.1. A function f is associative if for every x, y, z ,

$$f(f(x, y), z) = f(x, f(y, z)).$$

Definition 1.2. A value b is an identity of a binary function f if for every x ,

$$f(b, x) = x = f(x, b).$$

So, for now, you can think of `scan` as a magical function which performs iteration of an associative function in parallel. If the function is constant-time, then an application of `scan` has linear work and logarithmic span.

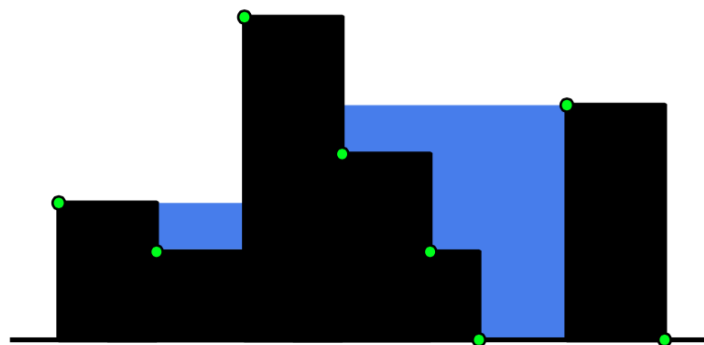
Remark 1.3. In reality, we can relax the constraint on the identity. It only needs to be an identity for the values encountered during the execution of the `reduce`, `scan`, or `scanIncl`. For example, if S is a sequence of non-negative integers, then $(\text{scan Int.max } 0 S)$ will still be logically equivalent to $(\text{iteratePrefixes Int.max } 0 S)$, despite the fact that, in general, 0 is not an identity for `Int.max`.

1.3 Skyline-Fill

For this example, we'll use the same conventions given in *SkylineLab*:

- Skylines are sequences of points (x, y) sorted by x -coordinate,
- all x -coordinates are unique and non-negative, and
- all y -coordinates (heights) are non-negative.

Imagine pouring water on a skyline. How much water can it hold?



Task 1.4. Implement the function

```
val fill : (int * int) Seq.t → int
```

where $(fill\ S)$ returns the area of water which can fill the skyline S . Your implementation should have $O(|S|)$ work and $O(\log |S|)$ span.

1.4 A Group at Dinner

A group of n friends sit around a circular table at a restaurant. Some of them know what they want to order; some of them don't. The ones who don't know what to order decide to pick the same thing as the person on their left.

Task 1.5. *Implement the function*

```
val groupOrder : (int → α option) → int → α Seq.t
```

where $(\text{groupOrder } f \ n)$ returns the sequence of orders of a group of n people. $f(i)$ is either the preferred order of the i^{th} person, or `NONE` if they don't know what they want. Assume the people are labeled 0 to $n - 1$ counter-clockwise, and that at least one person originally knows what they want to order. Your implementation should have $O(n)$ work and $O(\log n)$ span.

1.5 Bonus Exercises

Exercise 1.6. Implement `parenMatch` (from the previous recitation) using `scan` such that it has linear work and logarithmic span. Try adapting the iterative approach.

Exercise 1.7. Implement `parenDist` (from `ParenLab`) using `scan` such that it has linear work and logarithmic span.

Exercise 1.8. Did you know that you can calculate the first n Fibonacci numbers in $O(n)$ time and $O(\log n)$ span? We claim that if we extend the Fibonacci sequence as so...

$$\begin{aligned} F_{-1} &= 1 \\ F_0 &= 0 \\ F_1 &= 1 \\ &\vdots \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

...that the following holds for $n \geq 0$ (easily provable via induction):

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Using this fact, implement a function

```
val fibs : int → int Seq.t
```

which returns the first n Fibonacci numbers in $O(n)$ work and $O(\log n)$ span. Use `scan` to compute prefixes of matrix multiplications.

Exercise 1.9. Implement a function

```
val parenPairs : Paren.t Seq.t → (int * int) Seq.t
```

where `(parenPairs S)` returns a sequence of index-pairs where each pair contains the index of a parenthesis as well as its matching partner. For example the input `(()) ()` should yield some permutation of $\langle (0, 3), (4, 5), (1, 2) \rangle$. Your implementation should have $O(|S| \log |S|)$ work and $O(\log^2 |S|)$ span.

Hint: try marking each parenthesis with how many other parentheses are enclosing it. You might also need to `sort` at some point...

