Recitation 4

Scan Reloaded

4.1 Announcements

- *BignumLab* has been released, and is due Friday afternoon. It’s worth 175 points.
- *RandomLab* will be released on Friday.
4.2 Implementation

Recall the implementation of \texttt{scan} for sequences of power-of-2 length. Note that we typically refer to line 7 as the \textit{contraction} step, line 8 as the \textit{recursive} step, and line 11 as the \textit{expansion} step.

\begin{algorithm}
\begin{algorithmic}
\Fun{scan} \Var{f} \Var{b} \Var{S} =
\Case{\Var{|S|}}
\Case{0} \Rightarrow (\langle \rangle , \Var{b})
\Case{1} \Rightarrow (\langle \Var{b} \rangle , \Var{S}[0])
\Case{n} \Rightarrow
\Let{\Var{S'} = \langle \Var{f}(\Var{S}[2\Var{i}], \Var{S}[2\Var{i}+1]) : 0 \leq \Var{i} < n/2 \rangle}
\Var{R}, \Var{t} = \Var{scan} \Var{f} \Var{b} \Var{S'}
\Fun{P}(\Var{i}) = \If{\text{even}(\Var{i})} \Var{R}[\Var{i}/2] \Else \Var{f}(\Var{R}[\lfloor \Var{i}/2 \rfloor], \Var{S}[\Var{i} - 1])
\End
defined
\end{algorithmic}
\end{algorithm}

A diagram should help clear up any confusion. Consider $\langle \text{scan } + 0 \ (1, 2, 3, 4, 5, 6, 7, 8) \rangle$.
4.3 Cost Analysis

Since we so commonly use \texttt{scan} with a constant-time function argument, it is helpful to memorize that it has $O(n)$ work and $O(\log n)$ span in this case. But what about more complex functions? Let’s try \texttt{merge} as an example.

\begin{quote}
\textbf{Task 4.2.} Analyze the work and span of
\begin{equation}
\texttt{scan (merge cmp)} \langle \rangle \ S
\end{equation}
assuming that $|S| = n$, $|x| \leq m$ for every $x \in S$, and \texttt{cmp} is constant-time. Give your answers as tight Big-O bounds in terms of $n$ and $m$.
\end{quote}

Recall that \texttt{(merge cmp (A, B))} requires $O(|A| + |B|)$ work and $O(\log |A| + \log |B|)$ span, and it produces a sequence of length $|A| + |B|$.
### 4.4 Bonus Exercise: Factorials with Bignums

In this section, we write \(*^*\) for bignum multiplication and \(\pi\) for the bignum representation of \(x\). We’ll be using the same conventions here as in *BignumLab*.

Factorials quickly become too large to represent in a single 32-bit or 64-bit unsigned integer.\(^1\) This makes them the perfect candidate for bignums, which can be arbitrarily large. Consider the following code, which computes the first \(n\) factorials (excluding \(0!\)):

```
fun factorials n = Seq.scanIncl ** 1 (i: 1 ≤ i ≤ n)
```

**Exercise 4.4.** Analyze the work of \(\text{factorials } n\). Note that you’ll first need to determine

1. The work of \(\pi \times n\), and
2. The bit width of \(\pi \times n\).

*The former is given by solving the recurrence given in BignumLab for multiplication, namely

\[
W(n) = 3 W \left( \frac{n}{2} \right) + O(n).
\]

*The latter can be determined via a little bit of algebra. Note that the bit width of a number \(\pi\) is \(1 + \lfloor \log_2 x \rfloor\), assuming \(x \geq 1\).

*Warning: this is pretty hard.*

---

\(^1\)With 32-bit unsigned integers, the largest factorial we can compute before encountering overflow is \(11!\). For 64-bits, it’s \(19!\).