Recitation 1

Parenthesis Matching

1.1 Announcements

• Welcome to 15-210!

• The course website is http://www.cs.cmu.edu/~15210/. It contains the syllabus, schedule, library documentation, staff contact information, and other useful resources.

• We will be using Piazza (https://piazza.com/) as a hub for course announcements and general questions pertaining to the course. Please check it frequently to make sure you don’t miss anything.

• The first homework assignment, ParenLab, has been released! It’s due Monday at 11:59pm. There will be a second part released next Monday, which will be due the following Friday. Most homeworks will then have Friday due dates.

• Homeworks will be distributed through Autolab (https://autolab.cs.cmu.edu/). You will submit coding tasks on Autolab, and written tasks on Gradescope (https://gradescope.com/).

• ParenLab is conceptually difficult, so be sure to get started early.
1.2 Parentheses and Matched Sequences

Suppose you are given a sequence of parentheses. You want to determine if it is *matched*, meaning “properly nested”. Let’s begin by defining this more carefully.

**Definition 1.1.** A matched sequence of parentheses $p$ is defined inductively as

$$p ::= ⟨⟩ | p \cdot p | (p)$$

*In other words, a matched sequence is one of (a) the empty sequence, (b) the concatenation of two matched sequences, or (c) a pair of parentheses surrounding a matched sequence.*

To be consistent with ParenLab, we’ll implement parentheses as a custom datatype given in a structure `Paren`.

```plaintext
structure Paren =
  struct
    datatype t = L | R
    ...
  end
```

Our goal is to implement a function

```plaintext
val parenMatch : Paren.t Seq.t → bool
```

where `parenMatch S` determines whether or not $S$ is a matched sequence.

Note that you will need to familiarize yourself with the 210 library. Documentation can be found on the course website at http://www.cs.cmu.edu/~15210/docs/. In particular, you should look closely at the SEQUENCE interface and the ArraySequence implementation.
1.3 From Left to Right

**Task 1.2.** Implement `parenMatch` using the sequence function `iterate`.

1.4 Divide and Conquer

**Task 1.3.** Implement `parenMatch` with a divide-and-conquer approach. Your implementation should satisfy the following work and span recurrences where \( n \) is the length of the input.

\[
W(n) = 2 \cdot W\left(\frac{n}{2}\right) + O(1)
\]

\[
S(n) = S\left(\frac{n}{2}\right) + O(1)
\]

Also briefly justify that your implementation meets the cost bounds shown. You should assume `Seq = ArraySequence` for cost bounds.

**Hint:** to solve this problem, you’ll only need the sequence function `splitMid` and some basic arithmetic. Check out the documentation of `splitMid` on the website if you are not already familiar. You should also use `Primitives.par` for parallelism – the code `Primitives.par (fn () ⇒ e1, fn () ⇒ e2)` implements the parallel pair \((e_1 || e_2)\). It is logically equivalent to just writing \((e_1, e_2)\), except that the two expressions are evaluated in parallel.

1.5 Additional Exercises

**Exercise 1.4.** As implied by the name, the `ArraySequence` implementation of sequences lays out its elements in an array. Describe how to implement `splitMid` (and in general, `subseq`) in \(O(1)\) work and span.