`tag` — Version 1.0 ***due:*** *wed, sep 5 @ 11:59pm*

1 Introduction

This assignment is meant to help you familiarize yourself with the hand-in mechanism for 15-210 and to get you thinking about proofs and coding again. To that end, you will answer some questions about the mechanics and infrastructure of the course, then implement two solutions to the parentheses distance problem and perform some analysis of your solutions. Finally, you will do some exercises involving Big-*O* notation and prove an important identity.

1.1 Submission

This assignment is distributed in a number of files in our `git` repository. Instructions on how to access that repository can be found at `http://www.cs.cmu.edu/~15210/resources/git.pdf`. This is how assignments will be distributed in this course.

This assignment requires that you submit both code and written answers.

All of your code for this assignment must be in the files `paren.sml` and `test.sml`. Submit your solutions by placing these files in your handin directory, located at

`/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn1/`

Do not submit any other files. Name the files exactly as above. You can run the check script located at

`/afs/andrew.cmu.edu/course/15/210/bin/check/01.sh`

to make sure that you've handed in appropriate files.

Your written answers must be in a file called `hw01.pdf` and must be typeset. You do not have to use LaTeX, but if you do, we have provided the file `defs.tex` with some macros you may find helpful. This file should be included at the top of your `.tex` file with the command `\input{/<file>/<path>/defs.tex}`.

Your `paren.sml` file must contain all the code that you want to have graded for this assignment and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.2 Naming Modules

The questions below ask you to organized your solutions in a number of modules written from scratch. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

1.3 The SML/NJ Build System

This assignment includes a substantial amount of library code spread over several files. The compilation of this is orchestrated by CM through the file `sources.cm`. Instructions on how to use CM can be found at on the website at `http://www.cs.cmu.edu/~15210/resources/cm.pdf`.

2 Mechanics

The following questions are intended to make sure that you have read and understood the various policies for the course, as well as found the tools that we've set up to communicate with you.

Task 2.1 (1%). Describe the picture posted as an instructor's note on the course's Piazza board.

Task 2.2 (3%). In each of the following situations, have the students followed or broken the collaboration policy? Briefly justify your answers with a sentence or two.

1. Ludmilla, Joaquin, and Alexander have lunch together after 210 lecture. Over lunch, they discuss the homework assignment released earlier in the week, including their progress so far. After lunch, all three go to different classes and don't think about the 210 homework until that evening.

2. While working on 213 homework near his friends from 210, Jon has a moment of insight into the 210 homework assignment. He becomes excited, and tells his friends what he thought of. Ishmael hasn't gotten that far in the homework, so he doesn't quite understand what Jon is talking about. Nonetheless, Ishmael copies down what he thinks are the key bits of what he heard to look at when he gets that far.

3. Yelena has been working on the 210 homework but can't figure out why her solution isn't compiling. She asks Abida to work through a couple of toy examples of functor syntax with together, and they do so with a text editor and the SML REPL. Afterwards, Yelena gets her homework to compile and keeps working towards a solution.

Task 2.3 (1%). If you hand in your homework 25 hours after the posted due date, and you have no late days remaining, what is your maxiumum possible score?

3 The Parentheses Distance Problem

A string, s, is considered 'closed' if and only if it contains only '(' and ')' characters and is one of the following:

- The concatenation of two closed strings, $s_1 s_2$.
- A single closed string surrounded by a pair of 'matched' parentheses, (s_0).
- A single pair of matched parentheses, ().

More colloquially, s is closed if it could be compiled without generating unmatched-parentheses errors.

The distance between a pair of matched parentheses is the number of characters between the parentheses (exclusive). The maximum parentheses distance problem is to find the largest distance between a pair of matched parentheses in a closed string. More formally, for the closed string s, let M_s be the set of index pairs such that for $(i, j) \in M_s$, $i < j$ and (s_i, s_j) is a pair of matched parentheses. The maximum parentheses distance is

$$\max\{j - i - 1 | (i, j) \in M_s\}.$$

For example, for the string '**(**()()**)**()', the maximum parentheses distance would be 4.

When solving this problem, instead of interacting directly with strings, you will work with `paren` sequences, where the type `paren` is defined in a structure that ascribes to `PAREN_PACKAGE` and is given by

```
datatype paren = OPAREN | CPAREN
```

with `OPAREN` corresponding to a left parenthesis and `CPAREN` corresponding to a right parenthesis.

You will implement the function `parenDist : paren seq -> int option` such that `parenDist` S `= NONE` if S is not closed and `parenDist` S `= SOME` max if S is closed, where max is the maximum parentheses distance in S.

You will implement two solutions to this problem in the file `paren.sml`. Each solution will be a functor ascribing to the signature `PAREN`, defined in `PAREN.sig`. Each functor will take a structure ascribing to `PAREN_PACKAGE` as its only argument. The signature `PAREN` is defined in terms of the `SEQUENCE` signature from our library, which is documented in http://www.cs.cmu.edu/~15210/resources/docs.pdf. For testing, you should use the structure `ArrayParenPackage`, which uses the `ArraySequence` implementation of `SEQUENCE`.

How to indicate parallel calls? As seen in recitation, you can use the function `par` (inside the structure `Primitives`) to express calls that will be run in parallel. Parallel operations can also be expressed in terms of operations on sequences such as `map` or `reduce`. *In your code, be explicit about what calls are being made in parallel.*

3.1 Brute Force Implementation

Task 3.1 (5%).

Implement a brute-force solution to the maximum parentheses distance problem in a functor called `ParenBF`. You may use `match` from recitation 1 as part of your solution. You might also find `subseq` of the `sequence` library helpful.

Task 3.2 (5%).

What is the work and span for your brute-force solution? You can assume `subseq` has $O(m)$ work and $O(1)$ span, where m is the length of the resulting subsequence, and `match` has $O(n)$ work and $O(\log^2 n)$ span where n is the length of the sequence.

3.2 Divide and Conquer

Task 3.3 (25%).

Implement a solution to the maximum parentheses distance problem by divide-and-conquer recursive programming. If the work of showing any tree view of a sequence is denoted W_{showt}, the work of your solution must be expressed by the recurrence

$$W_{parenDist}(n) = 2\left(W_{parenDist}\left(\frac{n}{2}\right)\right) + W_{showt} + O(1)$$

with

$$W_{parenDist}(0) \in O(1)$$

A solution with correct input-output behavior but with work that is not described by this recurrence will not receive full credit. Write your implementation in a functor called `ParenDivAndConq`. On this assignment we are not requiring you to write a proof of correctness. But we advice that you work out proof by mathematical induction for your solution.

Task 3.4 (10%).

The specification in Task 3.2 stated that the work of your solution must follow a recurrence that was parametric in the work it takes to view a sequence as a tree. Naturally, this changes with the sequence implementation.

1. Solve the recurrence with the assumption that $W_{showt} \in O\left(\lg n\right)$.
2. Solve the recurrence with the assumption that $W_{showt} \in O(n)$ where n is the length of the input sequence.
3. In two or three sentences, describe what data structure you would use to implement the sequence α `seq` so that `showt` would actually have $O(\lg n)$ work.
4. In two or three sentences, describe what data structure you would use to implement the sequence α `seq` so that `showt` would actually have $O(n)$ work.

3.3 Testing Your Code

Task 3.5 (10%).

Write a structure called `ParenTest` ascribing to the signature `TESTS`. Your structure should thoroughly and carefully test both your implementations of the `PAREN` signature. This should include both edge cases and more general test cases on specific sequences.

You should write `ParenTest` in the file `test.sml`. At submission time there should not be any testing code in the same file as your various `PAREN` implementations, as it can make it difficult for us to test your code when you turn it in.

Your brute force implementation will likely be simple enough that it will have few edge cases and require a relatively small number of tests. Once you have confirmed that your brute-force solution works, you can write more exhaustive tests by comparing the faster solution against the slower one on many

sequences including those whose solution would be inconveinent to calculate by hand. **No tests should evaluate at compile time.**

To aid in your tests, we have provided the function `strToParens` which will convert a string containing only the '(' and ')' characters into the corresponding `paren seq`. We have also provided a basic test framework to serve as an example of the type of code we are expecting.

4 Asymptotics

For this problem, let's recall the definition of big-O:

Definition 4.1. A function $f : \mathbb{N} \to \mathbb{R}_+$ is in $O(g)$ if and only if there exist constants $N_0 \in \mathbb{N}$ and $c \in \mathbb{R}_+$ such that for all $n \geq N_0$, $f(n) \leq c \cdot g(n)$.

Task 4.1 (5%). Rearrange the list of functions below so that it is ordered with respect to O—that is, for every index i, all of the functions with index less than i are in Big-O of the function at index i. You can just state the ordering; you don't need to prove anything.

1. $f(n) = n^{\lg(2n)}$
2. $f(n) = 108^n$
3. $f(n) = n^{1.5}$
4. $f(n) = (2n)!$
5. $f(n) = 23n^{42} + 15n^{16} + 4n^8$
6. $f(n) = \lg(n)$
7. $f(n) = n^n$

Task 4.2 (15%). Carefully **prove or disprove** each of the following statements. Remember that verbose proofs are not necessarily careful proofs: your answers for these questions will likely fit on one page.

1. O is a transitive relation on functions. That is to say, for any functions f, g, h, if $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.
2. O is a symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$, then $g \in O(f)$.
3. O is an anti-symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$ and $g \in O(f)$, then $f = g$.

Task 4.3 (10%). Show that for $a \in \mathbb{R}$, $a \neq \pm 1$, $1 + a^2 + a^4 + \cdots + a^{2n} = \frac{a^{2n+2}-1}{a^2-1}$.