PLC: Homework 7 [75 points]

Due date: Wednesday, March 22, 10:30pm
3 point bonus if you turn in by Tuesday, March 21st, 10:30pm

About This Homework

The goal of this homework is to explore the basics of context-free grammars and parsing. As usual, the first thing you should do for this homework is copy the files from the `hw/hw7` subdirectory of the class repository, to a new `hw7` subdirectory of your personal repository, similarly to previous homeworks. You will just be modifying two of these files, namely `html.agda` and `html-thms.agda`, but we will refer to others of them in the instructions below.

There is extra credit possible for this assignment, so you can earn over 75 points (but 75 is a perfect score).

Partners Allowed

For this homework, you may work by yourself or with one partner (no more). See the instructions for `hw2` for how to create the `ack.txt` and `partner.txt` files that are required if you work with a partner

All files should go in a hw7 subdirectory (which you create) of your personal repository. Please call the subdirectory exactly that. Do not call it `Homework 4` or `Hw7` or other variations, or we will penalize you 5 points.

How to Turn In Your Solution

Make sure you have done a "Subversion add" on the `hw7` directory you created in your personal repository, and on the `.agda` files you have to submit for this assignment, and then do a "Subversion commit" on your personal repository directory to submit them. You can commit as many times as you want up to the deadline. If you commit after the deadline, we will use the last existing version before the deadline. Work submitted after the deadline will not count.

How to Check Your Solution

As for previous homeworks, you might want to make sure you have correctly submitted your solution via Subversion. Again, just go to the URL for your personal repository using a web browser, log in with your HawkId and password, and you can see what has been submitted.

Make sure each of the files you submit can be checked by Agda without any holes, yellow highlighting, or red highlighting. If any file does not check this way, we will penalize you 5 points total (if multiple files do not check, we will still just penalize you 5 points). Any

problems you do not solve should simply be removed from the `.agda` files, so that there are no holes (question marks) in the files.

How To Get Help

You can post questions in the `hw7` section on Piazza, or elsewhere on Piazza. See the course's Google Calendar, linked from Piazza, for the locations and times for office hours, including evening Skype office hours for Prof. Stump.

1 Reading

Please read Chapter 7 of the book.

Using gratr

This homework requires that you install and use `gratr`. Please follow the directions in Section 7.2 of the book for how to do this. I am including in the `hw7` directory the helper `.agda` files that need to be present when compiling a parser generated by `gratr`.

2 Understanding grammars [40 points]

For each of the following grammars (included with the `hw7` files I am providing to you), craft an input file for which you will see the parse tree shown when you run the generated parser with the `--showParsed` command-line option, on the input file. [10 points each]

1. For the grammar `plustimes.gr`, find an input that will generate the following parse tree (it will print on one line with gratr, but I am adding newlines just to fit it on the page here). Save your input in a file called `input.plustimes`.

```
(Strt (Plus (Id (id x))
        (Times (Parens (Plus (Id (id y)) (Num (num 3))))
               (Id (id z)))))
```

2. For the grammar `hpn.gr`, find an input that will generate the following parse tree. Save your input in a file called `input.hpn`.

```
(Strt (CmdsStart
         (ToDecimal
            (SeqStart (Pow (SeqNext (Pow (SeqNext (Pow Zero) Zero))
                                    (SeqNext (Pow Zero) Zero)))) (posnum 7))))
```

3. For the grammar `xml.gr`, find an input that will generate the following parse tree. Save your input in a file called `input.xml`.

```
(Strt (Nested (StartTag (id Simple) AttrsNil)
              (StartEnd (Startendtag (id Nested)
                 (AttrsCons (id id) (strlit "3") AttrsNil)))
              (EndTag (id Simple))))
```

4. For the grammar `lst.gr`, find an input that will generate the following parse tree. Save your input in a file called `input.lst`.

```
(Strt (Parens (App (Cons (Var (var hi)) (Var (var bye)))
                   (App (Var (var good)) Nil))))
```

3 Writing grammars [35 points]

Write a `gratr` grammar named `tp` in a file called `tp.gr`, for simple typing declarations. Your grammar should accept files like `test1.tp` consisting of sequences of typing declarations. Each typing declaration starts with a variable, which can be any sequence of lowercase letters. Then there is optional whitespace, then a colon, then more optional whitespace, and then a type. A type is either a variable (this is for base types like `bool` or `nat`), a parenthesized type (with optional whitespace between the parentheses and the type), or an arrow type (again with optional whitespace).