Homework 11: Context Free Grammars and Parsing

CIS 352: Programming Languages

6 April 2018, Version 1

Administrivia

- Trade ideas with another student? Note it in your cover sheet.
- Turn in Part I in the CIS 352 submissions box.
- Turn in Part II via Blackboard. Include: *(i)* the source files, *(ii)* the transcripts of test runs, and *(iii)* your cover sheet.
- Let me know if any of my QuickCheck tests seem dodgy.

Grading Criteria

- The homework is out of 100 points.
- Unless otherwise stated, each problem is $\approx$ 60% correctness and $\approx$ 40% testing.
- Omitting your name(s) in the source code looses you 5 points.

Mogensen (2010, §3.1–§3.5) gives some background for this assignment. (At the very least, skim these sections.)

Part I: Written Problems

❖ *Problem 1 (10 points)* ❖

Consider the grammar

$$
\begin{aligned}
\langle \text{sentence} \rangle &::= \langle \text{noun} \rangle \langle \text{predicate} \rangle \\
\langle \text{predicate} \rangle &::= \langle \text{verb} \rangle \langle \text{object} \rangle \mid \langle \text{verb} \rangle \langle \text{object} \rangle \langle \text{verb} \rangle \\
\langle \text{object} \rangle &::= \langle \text{adjective} \rangle \langle \text{noun} \rangle \mid \langle \text{noun} \rangle \\
\langle \text{adjective} \rangle &::= \textbf{her} \\
\langle \text{noun} \rangle &::= \textbf{I} \mid \textbf{her} \mid \textbf{duck} \\
\langle \text{verb} \rangle &::= \textbf{saw} \mid \textbf{duck}
\end{aligned}
$$

Non-terminals have enclosing pointy brackets (e.g., ⟨verb⟩) and terminals are in **bold**. Show that the grammar is ambiguous by constructing two non-equivalent parse trees for: "**I saw her duck**".

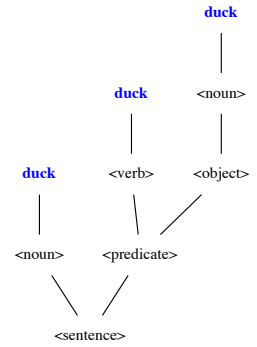

Figure 1: A parse of "**duck duck duck**"

❖ *Problem 2 (10 points)* ❖

Using the (abmiguous) grammar with start nonterminal S:

$$S ::= AS \mid \epsilon \qquad A ::= A\mathbf{1} \mid \mathbf{0}A\mathbf{1} \mid \epsilon$$

draw two distinct parse trees for the string **011**.

❖ *Problem 3 (18 points)* ❖

The grammar (with start nonterminal S):

$$S ::= (P)S \mid \# \qquad P ::= (P)P \mid \epsilon$$

describes the language of balance parentheses—with a '#' tacked on the end of the string. Construct a parse tree for each of:

(a) *(6 points)* $(())\#$

(b) *(6 points)* $()(())\#$

(c) *(6 points)* $((())())\#$

The '#' forces our parsers to read upto the '#'.

Part II: Problems based on Hutton, Chapter 8

See `http://www.cis.syr.edu/courses/cis352/code/Hutton/` for the starting files for this part. Add your code to `hw11.hs`.

❖ *Problem 4 (16 points)* ❖
Build a parser for the language of Problem 3. If your parser is working correctly, running `test3` should return `True`.[1]

[1] For ideas of how to proceed, take a look at Sample Parsers 1 and 2 in `hw11.hs`.

❖ *Problem 5 (15 points)* ❖
Problem 6 in Hutton (2007). Modifying `expr'` and `term'` in `hw11.hs` to include $-$ and `/` in the grammar. You can use the QuickCheck property `prop_parse1` to test your code.

❖ *Problem 6 (15 points)* ❖
The parser of the previous problem treats $+$, $-$, $*$, and `/` as right-associative. Complete the definition of `leftExpr` and `leftTerm` in `hw11.hs` to parse the expression grammar treating $+$, $-$, $*$, and `/` as left-associative. The function `leftTerm` is partially defined for you.[2] You can use the QuickCheck property `prop_parse2` to test your code.

[2] It handles $*$ left-associatively, but doesn't handle `/` at all.

❖ *Problem 7 (16 points)* ❖
BACKGROUND: The standard format used by the Unix `date` command to print dates is:

DayOfTheWeek Month Day Time TimeZone Year

For more information, see `http://www.unix.com/man-page/freebsd/1/date/`.

For example:

```
Tue Feb 29 03:18:00 GMT 2000
Thu Apr 14 14:21:33 EDT 2016
Mon May 2 14:40:00 EDT 2016
```

The Haskell package `Data.Time.Calendar` defines a type `Day` for calendar dates.
YOUR JOB: Define a parser

```
after100 :: Parser Day
```

such that `(parse after100` *inp*`)` parses an output from Unix's date command and returns the `Day` that is 100 days after that date.

For more information, see `http://hackage.haskell.org/package/time-1.6/docs/Data-Time-Calendar.html`. The function `addDays` will be handy.

A sample use of `Data.Time.Calendar`

```
ghci> :m Data.Time.Calendar
ghci> let april1 =
    fromGregorian 2018 4 1
ghci> april1
2018-04-01
ghci> addDays 45 april1
2018-05-16
```

EXAMPLES:

```
parse after100 "Tue Feb 29 03:18:00 GMT 2000"  ⇝  [(2000-06-08,"")]
parse after100 "Thu Apr 14 14:21:33 EDT 2016"  ⇝  [(2016-07-23,"")]
parse after100 "Mon May 2 14:40:00 EDT 2016"   ⇝  [(2016-08-10,"")]
```

References

G. Hutton. *Programming in Haskell*. Cambridge University Press, 2007.

T. Æ. Mogensen. *Basics of Compiler Design*. lulu.com, 2010. URL `http://www.diku.dk/~torbenm/Basics`.