Disc 8 - CFGs and Parsing

Tuesday, October 26, 2021 9:43 PM

1) Identify the terminals, non-terminals, production rules, and the start state of the following CFG:
   S -> aSa | B
   B -> bC | C
   C -> epsilon

2) Show that the following CFG is ambiguous:
   S -> S + S | 1

3) Convert the following into a non-ambiguous CFG:
   S -> S + S | 1

4) Convert the following CFG into one that can be used with recursive descent parsing:
   S -> S + S | M

5) Convert the following CFG into one that can be used with recursive descent parsing:
   S -> S + S | 1 | 2 | 3

6) Write a CFG for the following:

$a^x b^x$

7) Write a CFG for the following:

$a^x b^y$, where $x = 2y$

8) Write a CFG for the following:

$a^x b^y$, where $x > y$

9) Write a CFG for the following:

$a^x b^y c^z$, where $z = x + y$

10) Write a grammar for all palindromes consisting of a's and b's

11) Write a CFG equivalent to (wp)+g*

12) Write a CFG for the following:

$a^x b^y c^z$, where $y = x + z$

13) Define a lexer for the following token definitions

```
type token =
| Tok_Int of int
| Tok_Add
| Tok_EOF
```

Useful functions:

```
Str.string_match (Str.regexp "some string") string index -> boolean
Str.matched_string input -> string
```

```
let rec lexer (input: string) : token list =
```

14) Define a parser for the above tokens and following CFG and expr type

S -> M + S | M
M -> n
Where n is any integer

```
Type expr =
| Int of int
| Plus of expr * expr
```

Useful functions:

```
match_token (toks : token list) (tok : token) : token list
lookahead (toks : token list) : token
```

```
let rec parser (toks : token list) : expr =
```

```
and parse_S (toks : token list) : (token list * expr) =
```

```
and parse_M (toks : token list) : (token list * expr) =
```