Assignment 6

CAS CS 320: Principles of Programming Languages

Due: **Friday March 22, 2024 by 11:59PM**

Submission Instructions

- You will submit a `.pdf` of your solutions on Gradescope. Your solutions must be legible. If you do not have neat handwriting, please type your solutions.
- **Put a box around the final answer in your solution.** Or otherwise make the final answer in your solution abundantly clear.
- Choose the correct pages corresponding to each problem in Gradescope. Note that Gradescope registers your submission as soon as you submit it, so you don't need to rush to choose corresponding pages. **For multipart questions, please make sure each part is accounted for.**

We will dock points if any of these instructions are not followed.

1 A Simple Ambiguous Grammar

$$
\begin{array}{lll}
\langle start \rangle & ::= & \texttt{A}\ \langle start \rangle\ \texttt{B} \\
& | & \langle a \rangle \\
\\
\langle a \rangle & ::= & \texttt{A}\ \langle a \rangle \\
& | & \texttt{A}\ \langle b \rangle \\
\\
\langle b \rangle & ::= & \texttt{B}\ \texttt{B}\ \langle b \rangle \\
& | & \texttt{B}\ \texttt{B}
\end{array}
$$

A. Find the *shortest* sentence which has multiple derivations. You do not need to write down the derivations.

B. Write down a right linear regular grammar for the set of sentences of the above grammar which have *at least 2* `A` *symbols and at least 2* `B` *symbols.*

C. Write down a regular expression for the set of sentences of the above grammar.

Solution.

A. AAABBBB

B.

$$
\begin{array}{lll}
\langle a \rangle & ::= & \texttt{A}\langle a \rangle \\
& | & \texttt{AA}\langle b \rangle \\
\\
\langle b \rangle & ::= & \texttt{B}\langle b \rangle \\
& | & \texttt{BB}
\end{array}
$$

C. a^*bb^*

2 A More Interesting Ambiguous Grammar

Consider the following EBNF grammar. Note that there are two uses of parentheses, one which is part of the language and one which is part of EBNF syntax. Also note the use of the visible space symbol '␣' which indicates that a space *must* appear in the sentence (whereas we typically use whitespace for convenience to visually separate terminal symbols).

⟨*prgm*⟩ ::= { `LET` ⟨*idnt*⟩ `=` ⟨*expr*⟩ }

⟨*expr*⟩ ::= `FUN` ⟨*idnt*⟩ `=>` ⟨*expr*⟩
| `CASE` ⟨*expr*⟩ `OF` { `/` ⟨*pat*⟩ `->` ⟨*expr*⟩ }
| ⟨*term*⟩ { (`+` | `-`) ⟨*term*⟩}

⟨*term*⟩ ::= (⟨*idnt*⟩ | ⟨*num*⟩ | `(` ⟨*expr*⟩ `)`) [␣ ⟨*term*⟩]

⟨*pat*⟩ ::= ⟨*idnt*⟩ | ⟨*num*⟩ | `*`

⟨*num*⟩ ::= (`0` | `1` | `2` | ... | `9`) { `0` | `1` | `2` | ... | `9` }

⟨*idnt*⟩ ::= (`a` | `b` | `c` | ... | `z`) { `a` | `b` | `c` | ... | `z` }

A. **TRUE** or **FALSE**, the following program has a derivation in the above grammar:

```
LET fib = FUN x =>
  CASE x OF
  / 0 -> 0
  / 1 -> 1
  / n -> fib␣(n - 1) + fib␣(n - 2)
```

Solution. **TRUE**

B. Give a derivation for the following program.

```
LET z = FUN x =>
  CASE x OF
  / 0 -> 1
  / * -> 0
```

Notes.

- The newlines are for visual convenience, your derivation should end in the single line

  ```
  LET z = FUN x => CASE x OF / 0 -> 1 / * -> 0
  ```

- The sentential form may become somewhat wide. You may shorten terminal and nonterminal symbols if it is sufficiently clear.
- You may replace nonterminal symbols in parallel, as in the previous assignment.
- Do not include EBNF syntax in the derivation, you should immediately replace the EBNF syntax with a corresponding sentential form, e.g.,

  ```
  <num>  ==> 00120

  <expr> ==>
  CASE <expr> OF / <pat> -> <expr> / <pat> -> <expr>
  ```

Solution.

```
        <prgm>
      => LET <idnt> = <expr>
      => LET Z = FUN x => CASE <term> OF /<pat> -> <expr
   > /<pat> - <expr>
      => LET Z = FUN x => CASE x OF /0 -> 1 /* -> 0
```

C. Rewrite the production rule for `<term>` using standard BNF syntax. You may not introduce new nonterminal symbols.

Solution.

$$\begin{array}{lcl} \langle \textit{term} \rangle & ::= & \langle \textit{idnt} \rangle \\ & \mid & \langle \textit{num} \rangle \\ & \mid & (\langle \textit{expr} \rangle) \\ & \mid & \langle \textit{term} \rangle \text{␣} \langle \textit{term} \rangle \end{array}$$

D. Demonstrate that this grammar is ambiguous by finding a sentence which has multiple derivations. You do not need to give the derivations.

Solution. `a ␣ (b) ␣ c`

E. Change the above grammar by introducing a single terminal symbol so that it is no longer ambiguous. You don't need to reproduce the entire grammar, just give the production rule where you put the terminal symbol.

Solution.

$$\begin{array}{lcl} \langle \textit{term} \rangle & ::= & \langle \textit{idnt} \rangle \\ & \mid & \langle \textit{num} \rangle \\ & \mid & (\langle \textit{expr} \rangle) \\ & \mid & \langle \textit{idnt} \rangle \text{␣} \langle \textit{term} \rangle \\ & \mid & \langle \textit{num} \rangle \text{␣} \langle \textit{term} \rangle \\ & \mid & (\langle \textit{expr} \rangle) \text{␣} \langle \textit{term} \rangle \end{array}$$

3 Regular Grammars and Expressions

A. Write down a regular expression which recognizes all binary strings without a contiguous subsequence of 1s of length 3, e.g., ϵ and `001` and `00110001001000001` but not `111` or `0011000101111001`.

Solution.

$$(0|10|110)^*(\epsilon|1|11)$$

B. Write down a right linear regular grammar which recognizes all binary strings with an odd number of zeros and an odd number of ones, e.g., `01` and `001011` but not `001100010` or `0011` (*Note.* It would be a fair amount more difficult to give a regular expression, even though the two representations are equivalent in expressivity).

Solution.

```
<S> := 0<A> | 1<B>
<A> := 0<S> | 1<C> | 1
<B> := 1<S> | 0<D> | 0
<C> := 1<A> | 0<E> | 0
<D> := 0<B> | 1<F> | 1
<E> := 1<C> | 0<S>
<F> := 0<D> | 1<S>
```

C. Write down a regular expression which recognizes all strings of characters without whitespace or parentheses which represent (possibly empty) `int list`'s of `0`'s in OCaml, e.g., `0::0::[]` and `0::[0;0;0]` and `[0;]` (note the optional trailing semicolon) and `[]`.

Solution.

```
(0(::0)*(::(\[0(;0)*(\;)?\])?)?(\;)?|(\[0(;0)*(\;)
?\]))
```