COL226: Programming Languages

Fri 19 Mar 2021 **Minor** 90 minutes Max marks 80

Instructions:
1. Download the paper.
2. Write your name and entry number in the designated space on top and *do not forget to sign the honour statement below.*
3. Answer the question(s) in the appropriate space provided starting from this page.
4. Scan the paper with your completed answer.
5. Upload it on Gradescope 2002-COL226 page within the given time. *Make sure the first page with your name, entry no and signature is also the first page of your uploaded file*
6. Late submissions (within 5 minutes of submission deadline) on the portal will attract a penalty of 20% of the total marks.
7. Email submissions after the closing of the portal will not be evaluated (You get a 0).
8. Uploads without the first page details (including signature) may be awarded 0 marks.

I abide by the Honour code that I have signed on my admission to IIT Delhi. I have neither given any help to anybody nor received any help from anybody or any site on the internet in solving the question(s) in this paper.

Signature: **Date:**

1. **[20 marks.]** Many systems recognize commands, filenames and folder names by the their unique prefix. For instance, given the 3 commands `chmod`, `chgrp` and `chown`, among many of their unique prefixes the shortest unique prefixes are respectively `chm`, `chg` and `cho`. A user can type the unique prefix of the command as input and press the enter key (represented by the termination character \$), and the system will automatically complete it for him/her.

   (a) Draw a DFA which recognizes all unique prefixes as inputs that are at least as long as the shortest unique prefix of each of the above commands. Write the regular expression for the same and show only the associated lexing rule that one must write in the lex specification. You may consider token types CHGRP, CHMOD, CHOWN for specifying the rules.

   (b) Suppose the set of commands also includes two more commands `cmp` and `cmpdir` (with corresponding token types CMP and CMPDIR respectively), state how you will include such commands also in your DFA where one command is a prefix of another. There is no need to specify lexing rules in this case.

2. **[20 marks.]** Prove that the grammar $G = \langle \{S\}, \{a, b\}, \{S \to \varepsilon \mid aSbS\}, S\rangle$ is unambiguous.

   Solution.

   Proof by induction.

   We prove it by induction on the lengths of sentences $x \in \mathcal{L}(G)$ generated by the grammar.

   Basis. For $|x| = 0$ $x = \varepsilon$ is the only sentence and the only way to produce it is by applying the production $S \to \varepsilon$ (any application of the other production would yield a sentence of length at least 2 since it would contain at least one a and one b).

   Induction hypothesis. For every $y \in \mathcal{L}(G)$ with $|y| < k$ for some $k > 0$, there is a unique (derivation) parse tree.

   Induction step. Let $|x| = k$ and $x \in \mathcal{L}(G)$. Then x has to be generated by at least one application of $S \to aSbS$ and hence $x = aybz$ for some $y, z \in \mathcal{L}(G)$ with $|y| < k$ and $|z| < k$. By the induction hypothesis each of y and z has a unique derivation (parse) tree. and hence the derivation tree obtained for the derivation $S \Rightarrow aSbS \Rightarrow^* aybz$ is unique.

   Alternative proof.

   Assume to the contrary that G is ambiguous. Then there is at least one sentence $\mathcal{L}(G)$ which has at least two distinct derivation trees. Let $x \in \mathcal{L}(G)$ be the *shortest* sentence with at least two distinct parse trees. Then $x \neq \varepsilon$ since there is only one way of generating ε from S viz. $S \to \varepsilon$ (any application of the other production would yield a sentence of length at least 2 since it would contain at least one a and one b). Therefore $|x| > 0$ and $S \Rightarrow^* x$ implies $S \Rightarrow aSbS \Rightarrow^* x$. Then clearly $x = aybz$ for some $y, z \in \mathcal{L}(G)$. However since y and z are shorter sentences than x they must have unique parse-trees and since $S \Rightarrow aSbS$ is the only production that can be applied initially in order to generate the sentence $aybz$ and y and z have unique parse trees, there is only one parse tree that generates x, contradicting the assumption that x has at least two distinct parse trees.

3. **[20 marks.]** You have already seen how to generate abstract syntax trees (AST) after scanning and parsing a context-free grammar. It is also sometimes possible to reverse this process – though the reverse process is not guaranteed to yield exactly the original input (syntactically), it will yield a (semantically equivalent) string which has the same AST.

   NO FORMAL PROOFS ARE REQUIRED FOR THIS QUESTION – PROCEED WITH INTUITION AND UNDERSTANDING.

   Consider the following context-free grammar $G = \langle N, T, P, A \rangle$ where

$$\begin{array}{rcl} N &=& \{A, M, X, F\} \\ T &=& \{+, *, **, (,)\} \cup \{\mathtt{i} \mid \mathtt{i} : \mathtt{int}\} \end{array}$$

   with the set P of productions defined by

$$\begin{array}{rcl} A &\to& M \mid A{+}M \\ M &\to& X \mid M{*}X \\ X &\to& F \mid F{**}X \\ F &\to& \mathtt{i} \mid (A) \end{array}$$

   Now consider the SML data-type of abstract syntax trees (ASTs) of the above grammar.

```
dataype ast = NUM of int | PLUS of ast * ast
            | MULT of ast * ast | POW of ast * ast
```

   where `PLUS` is generated by $+$, `MULT` by $*$, `POW` by $**$ and `NUM(i)` by `i`.

   (a) Give an example of two strings $s, s' \in \mathcal{L}(G)$ such that $s \neq s'$ but they both have the same abstract syntax tree.

   (b) Define a SML function `toString : ast -> string` such that for any `t` : `ast`, `toString t = s:string` such that the abstract syntax tree of `s` is `t`.

<u>*Solution.*</u>

   (a) Consider a string s = "`3*(4+5+1)`" which can be generated by the grammar G. The AST `t` for this sentence will be

   `MULT (NUM(3), PLUS(PLUS(NUM(4),NUM(5)),NUM(1))`

   and `toString (t)` will generate the fully parenthesized string s' = "`(3*((4+5)+1))`" which has the same AST but a different derivation (parse) tree because of the extra pairs of parentheses. The outermost pair of parentheses could be discarded if we define `toString (t)` in a little more complicated fashion, but the the two pairs of inner parentheses would still have to be present.

   (b) We define a fully parenthesized form for each abstract syntax tree which guarantees that an equivalent string is obtained which can be generated by the grammar. Parentheses are absolutely essential to ensure that whenever precedences or associativity have to be over-ridden then parentheses may be used for a semantics-preserving translation. Only the outermost pair of parentheses may be safely deleted. But for uniformity we present a fully parenthesized string.

```
fun toString (NUM i) = Int.toString i
  | toString (PLUS (t1, t2)) = "("^(toString t1)^"+"^(toString t2)^")"
  | toString (MULT (t1, t2)) = "("^(toString t1)^"*"^(toString t2)^")"
  | toString (POW (t1, t2)) = "("^(toString t1)^"**"^(toString t2)^")"
```

4. **[20 marks.]**

   (a) Design an implementation in SML, with the help of a stack data structure, for the evaluation as a semantic action for the grammar $G = \langle \{E\}, \{+, *, i\}, \{E \to EE+ | EE* | i\}, E\rangle$ where

   - $+$ and $*$ are postfix binary operators for addition and multiplication on integers respectively,
   - the i token described by the regular expression $0|[1-9][0-9]*$, represents an integer constant.

   Note that the result of expression evaluation must be at the top of the stack. The `STACK` signature is as follows:

```
signature STACK =
sig
    type  'a stack
    exception  EmptyStack
    val isEmpty : 'a stack -> bool
    val push : ('a * 'a stack) ->  'a stack
    val  pop : 'a stack -> 'a stack
    val  top : 'a stack -> 'a
end
```

   (b) Consider the statement: *"Infix operators at the same precedence level can have different associativities"*. Analyse the statement for its correctness with justifications.