CMPT 379 Fall 2012 - Midterm

Anoop Sarkar – Simon Fraser University

(1) **Lexical Analysis and Regular Expressions**

a. (5pts) The following token definitions are provided to you, but they are not in any particular order. The tokens are defined using regular expressions with the usual syntax, `[]` denotes a character class and `?` is an operator for its argument occurring zero or one time.

`[a-zA-Z\_0-9][a-zA-Z0-9]*`	TOKEN-A
`[0-9]+`	TOKEN-B
`:?[a-zA-Z]+`	TOKEN-C

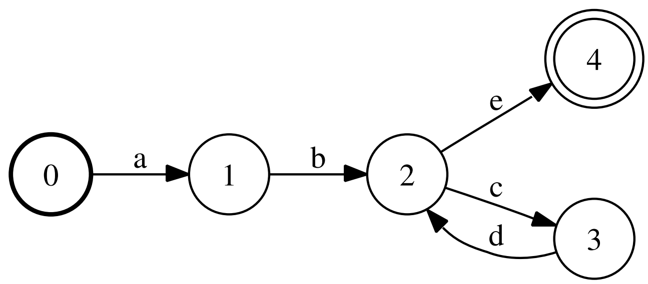
Assuming the usual greedy longest match strategy for lexical analysis, for the input string `a:b0_0` the desired output tokens and lexeme values are TOKEN-C(`a`), TOKEN-C(`:b`), TOKEN-A(`0`), TOKEN-A(`_0`). Provide the correct ordering of the tokens in order to produce this output.

Answer: TOKEN-C > TOKEN-A > TOKEN-B

b. (3pts) For the input string `a0:a` is it possible to get the output sequence TOKEN-C(`a`), TOKEN-B(`0`), TOKEN-C(`:a`) under any ordering of the tokens? Briefly explain your yes/no answer.

Answer: No. TOKEN-A will always be able to match `a0` as the longest match regardless of the ordering of the token definitions.

c. (2pts) Consider the following DFA D:



Provide a regular expression for the regular language generated by this DFA.

Answer: `ab(cd)*e`

(2) **Context-free Grammars and Deterministic Parsing**

Consider the following CFG G:

$$S' \rightarrow S$$
$$S \rightarrow aSa \mid bSb \mid \epsilon$$

a. (2pts) Is G an ambiguous CFG? If your answer is yes, then provide an input string for which G has two leftmost derivations. If your answer is no, then briefly explain why for *any input string* there will always be an unique leftmost derivation for that string.

Answer: G is not ambiguous. For input string ϵ there is exactly one leftmost derivation: $S' \Rightarrow S \Rightarrow \epsilon$. Any other string in $L(G)$ is a string x^n where x^n is a palindrome and $x \in \{a, b\}$ and n is even. Therefore, there must be a unique $n+1$ step leftmost derivation $S' \Rightarrow^* x^n$ to derive a palindrome x^n. This is because at each step there is exactly one S to be expanded and the choice depends on whether x is a or b. The derivations always follow the pattern: $S' \Rightarrow xSx \Rightarrow xxSxx \Rightarrow \ldots \Rightarrow xx\ldots xx$ where the $(n+1)^{\text{th}}$ step uses the $S \rightarrow \epsilon$ rule.

b. (2pts) Is G an LL(1) grammar? Explain your answer using FIRST and FOLLOW sets appropriately.

> *Answer:* No, it is not an LL(1) grammar because the intersection of FIRST(aSa) and FOLLOW(S) = $\{a, b, \$\}$ is non-empty and so we cannot choose deterministically whether to choose $S \to aSa$ or $S \to \epsilon$.

c. (2pts) Is G an LL(2) grammar? Explain your answer using FIRST_2 sets, which contain the symbol pairs that can be observed when expanding a non-terminal, and the FOLLOW_2 sets which are the symbol pairs that can follow a non-terminal. Assume that there are *two* end of input symbols: $$.

> *Answer:* No, it is not an LL(2) grammar because the intersection of $\text{FIRST}_2(aSa) = \{aa, ab\}$ and $\text{FOLLOW}_2(S) = \{aa, ab, ba, bb, \$\$\}$ is non-empty and so we cannot choose deterministically whether to choose $S \to aSa$ or $S \to \epsilon$.

d. (4pts) Is the CFG G an LR(1) grammar? Provide exactly two item sets starting with $S' \to \cdot S, \$$ using the closure condition for LR(1) plus successor to justify the answer. Do not provide the parsing table.

> *Answer:*
>
> $$
> \begin{array}{lll}
> 0: & & \\
> S' & \to & \cdot S, \$ \\
> S & \to & \cdot aSa, \$ \\
> S & \to & \cdot bSb, \$ \\
> S & \to & \epsilon\cdot, \$ \\
> & & \\
> 1: & & \\
> S & \to & a \cdot Sa, \$ \\
> S & \to & \cdot aSa, a \\
> S & \to & \cdot bSb, a \\
> S & \to & \epsilon\cdot, a
> \end{array}
> $$
>
> Item set 1 is the successor for item set 0. Item set 0 has no conflicts but item set 1 has a shift-reduce conflict, shift on a or reduce $S \to \epsilon$ on look ahead a.