CS164: Written Assignment 2
(On Grammars, Parsing, and NFAs)

Assigned: Thursday, Sep 23, 2004
Due: Thursday, Sep 30, 2004, at the beginning of class.

Grading and Submission

Your answers must be brief and easy to understand. Your grade (credit / no credit) will depend partly on how easy it is for us to understand and verify your answer. Submit your written assignments either in the classroom (before the lecture) or in 283 Soda. *No late homeworks are accepted.* **Please indicate your login name and Section number.**

1 How to Develop a Predictive $LL(1)$ Parser

Background: In PA3, you will develop an automatic generator of predicitive LL(1) parsers. This problem will help you understand how to do it.

Question: Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow RT \\
R &\rightarrow sURb \\
R &\rightarrow \epsilon \\
U &\rightarrow uU \\
U &\rightarrow \epsilon \\
V &\rightarrow vV \\
V &\rightarrow \epsilon \\
T &\rightarrow VtT \mid \epsilon
\end{aligned}
$$

1. Give a leftmost derivation of $ssuurrtvt$. For each step, indicate which production you applied.
2. Compute the FIRST and the FOLLOW sets for this grammar.
3. Construct an $LL(1)$ predictive parse table for the grammar.
4. Show the moves made by this parser on the input $ssuurrtvt$ (i.e., show a table for the stack, input, and production used at each step, as shown on slide 35 from Sep 21).

2 How to Create an LL(1) Grammar?

Background: In the lecture, we covered several transformations that may help you turn a grammar into an $LL(1)$ grammar (namely, elimination of left recursion, left factoring, and "de-ambiguation"). Unfortunately, performing these transformations does not guarantee to produce an $LL(1)$ grammar.

Question: (a) Give an example of a grammar that is unambiguous, left-factored, and not left recursive that is also not $LL(1)$. Justify your answer.

(b) Is this grammar $LL(k)$ for some k? If yes, what is k?

(c) Can you suggest how to rewrite the grammar to make it $LL(1)$. (Note: no such transformation may be possible for your grammar.)

3 How Expressive are Regular Expressions and Grammars?

Background: Context-free grammars are more powerful than regular expressions because they can specify languages that regular languages can't (e.g., the language of arithmetic expressions with balanced parentheses). However, typical programming languages contain syntactic fragments that can be described with a regular expression, which is often simpler than with a grammar. For example, variable declarations may have the form

```
(int | float) id (, id)* ;
```

So, sometimes it is convenient to write these fragments as regular expressions, and then rewrite them into a context-free grammar requiored by your parser generator. This questions asks how to do this.

Question: Give a context-free grammar that describes the same set of strigns as the following regular expression.

$$a((b|a.c^*)x)^*|x^*a$$

4 Translating Regular Expressions into NFAs

Consider slide 66 from the lecture "Building a Scanner" from Sep 2. The slide shows how to construct an NFA for the regular expression operator '|'. The ϵ-transitions shown in this construction may appear superfluous, in that it looks like it may be possible to build the automaton for $A|B$ by simply merging the start states of A and B, making the merged state the start state of $A|B$; similarly for the accepting states of A and B. (When two states are "merged," their transitions, both incoming and outgoing, are merged, too.) This question asks you to explain why this method for creating the NFA for $A|B$ is incorrect.

Question: Give regular expressions A and B for which the above method for constructing the NFA for $A|B$ produces an NFA that accepts something else than $L(A|B)$.