# CS3300 Midterm

Dept of CSE, IIT Madras
Total marks = 40

September 30, 2013

Read the questions carefully. State all your assumption clearly. There are total six questions, each carrying 8 marks each. Maximum marks you can attempt = 48. Maximum marks you can score = 40. For questions that have sub-parts, the division for the sub-parts are mentioned in square brackets.

1. [8] **Lexical Analysis** a new language expects the following lexical tokens:
   - Function names: Letter followed by one or more occurrences of letters / digits, but ending with a letter.
   - Type names: Letter followed by one or more occurrences of letters / digits, but ending with a digit.
   - Variable names: A single Letter.

   Draw a single transition diagram (DFA) for recognizing these three tokens. [2]
   Use the diagram to build the lexical transition table. [2]
   Give a sketch of lexical recognizer that uses this transition table to recognize a series of these tokens delimited by white space [4].

2. [8] **Parsing I** Define the FIRST and FOLLOW relations and define the conditions under which a LL(1) grammar can be declared unambiguous. [2]
   If we have a LL(k) grammar, would the definitions of FIRST and FOLLOW change to use the above unambiguity conditions? If yes, give the new definitions. Else give a reason why not. [3]
   A mother has the option to Kiss or scold her child. For the child's growth to be healthy, it is desirable that at any point of time during the day, the number of times the child got scolded so far that day should be $\leq$ the number of times the child got kissed so far that day. Some example desirable sequences are: *Kiss Scold Kiss Kiss* and *Kiss Kiss Kiss*. Some undesirable sequences are: *Kiss Scold Scold Kiss* and *Scold*. Write an LL(1) grammar to generate all the "desirable" sequence of maternal interactions each day and argue that the grammar is LL(1). [3]

3. [8] **Parsing II** Show that the following grammar is LR(1) [4], but not LALR(1) [4].
   `S` $\rightarrow$ `A a | b A c | B c | b B a`
   `A` $\rightarrow$ `d`
   `B` $\rightarrow$ `d`

4. [8] **Syntax Directed Definition** Give an SDD to translate infix expressions with + and $*$ into equivalent expressions without redundant parenthesis, by taking into consideration precedence and associativity. For example, `((a*(b+c))*(d))` translates into `a*(b+c)*d`.

5. [8] **Syntax directed translation** The following SDT computes the value of a string of 0's and 1's interpreted as a signed binary integer.
   $N \rightarrow$ `Sign` $L$ {$N.val$ = `Sign.`$bit$ `*` $L.val$}
   `Sign` $\rightarrow$ `'+'` | $\epsilon$ {`Sign.bit = 1`}
   `Sign` $\rightarrow$ `'-'` {`Sign.bit = -1`}
   $L \rightarrow L_1$ `0` {$L.val = 2 \times L_1.val$}
   $L \rightarrow L_1$ `1` {$L.val = 2 \times L_1.val$ `+ 1`}
   $L \rightarrow$ `1` {$L.val = 1$}
   Rewrite the grammar to be LL(1) and yet we compute the $N.val$ for the entire input string.

6. [8] **IR Generation** Write a SDT to generate IR in three-address code (similar to the one discussed in the class) for the following grammar. Briefly explain about the attributes you use.
   `P` $\rightarrow$ `S`
   `S` $\rightarrow$ `SwitchStmt S | Assignment; S` | $\epsilon$
   `SwitchStmt` $\rightarrow$ `switch ( Id ) { CaseBlocks }`
   `CaseBlocks` $\rightarrow$ `case Lit : Stmt2; CaseBlocks | default: Stmt2`
   `Stmt2` $\rightarrow$ `Assignment | Break`
   `Assignment` $\rightarrow$ `x = E`
   `E : RelEx | AddEx | Id`
   `RelEx` $\rightarrow$ `E` $<$ `E`
   `AddEx` $\rightarrow$ `E` $+$ `E`
   `Break` $\rightarrow$ `break`