Mid-term Exam
COSE312 Compilers, Fall 2015

Instructor: Hakjoo Oh

Problem 1 Consider the following subset of regular expressions:

$$R \to \emptyset \mid \epsilon \mid a \in \Sigma \mid R_1^+ \mid R_1?$$

The semantics of the language is defined by a function L as follows:

$$\begin{array}{rcl} L(\emptyset) & = & \emptyset \\ L(\epsilon) & = & \{\epsilon\} \\ L(a) & = & \{a\} \\ L(R_1^+) & = & (L(R_1))^+ = \bigcup_{i\geq 1}(L(R_1))^i \\ L(R_1?) & = & L(R_1) \cup \{\epsilon\} \end{array}$$

Define Thomson's construction (i.e., "compilation" from regular expressions to NFAs) for the above language.

Problem 2 Consider the expression grammar:

$$\begin{array}{lcl} E & \to & T\ E' \\ E' & \to & +\ T\ E' \mid \epsilon \\ T & \to & F\ T' \\ T' & \to & *\ F\ T' \mid \epsilon \\ F & \to & (E) \mid \mathbf{id} \end{array}$$

The $FIRST$ and $FOLLOW$ sets are given:

- $FIRST(F) = FIRST(T) = FIRST(E) = \{(, \mathbf{id}\}$.
- $FIRST(E') = \{+, \epsilon\}$.
- $FIRST(T') = \{*, \epsilon\}$.
- $FOLLOW(E) = FOLLOW(E') = \{), \$\}$.
- $FOLLOW(T) = FOLLOW(T') = \{+,), \$\}$.
- $FOLLOW(F) = \{+, *,), \$\}$.

The predictive parsing table is constructed as follows:

	id	$+$	$*$	(	)	\$
E	$E \to T\ E'$			$E \to T\ E'$		
E'		(1)			(2)	$E' \to \epsilon$
T	$T \to F\ T'$			$T \to F\ T'$		
T'		(3)	(4)		(5)	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Find the production rules for (1)–(5).

Problem 3 Consider the expression grammar:

$$\begin{array}{llcl} (1) & E & \to & E + T \\ (2) & E & \to & T \\ (3) & T & \to & T * F \\ (4) & T & \to & F \\ (5) & F & \to & (E) \\ (6) & F & \to & \mathbf{id} \end{array}$$

and its bottom-up parsing table:

	id	$+$	$*$	(	)	\$	E	T	F
0	$s5$			$s4$			$g1$	$g2$	$g3$
1		$s6$				acc			
2		$r2$	$s7$		$r2$	$r2$			
3		$r4$	$r4$		$r4$	$r4$			
4	$s5$			$s4$			$g8$	$g2$	$g3$
5		$r6$	$r6$		$r6$	$r6$			
6	$s5$			$s4$				$g9$	$g3$
7	$s5$			$s4$					$g10$
8		$s6$			$s11$				
9		$r1$	$s7$		$r1$	$r1$			
10		$r3$	$r3$		$r3$	$r3$			
11		$r5$	$r5$		$r5$	$r5$			

Complete the following parsing sequence for string $\mathbf{id} * \mathbf{id}$:

Stack	Symbols	Input	Action
0		$\mathbf{id} * \mathbf{id}\$$	shift to 5
0 5	**id**	$*\mathbf{id}\$$	reduce by 6 ($F \to \mathbf{id}$)

Problem 4 Consider the following high-level imperative language C:

$$
\begin{array}{rcl}
C & \to & \texttt{skip} \\
 & | & x = E \\
 & | & \texttt{while}\ B\ C \\
 & | & C_1; C_2 \\
E & \to & n \mid x \mid E_1 + E_2 \\
B & \to & \texttt{true} \mid \texttt{false} \mid E_1 < E_2
\end{array}
$$

and a low-level language T:

$$
\begin{array}{rcl}
T & \to & \mathit{LabeledInstruction}^* \\
\mathit{LabeledInstruction} & \to & \mathit{Label} \times \mathit{Instruction} \\
\mathit{Instruction} & \to & \texttt{skip} \\
 & | & x = y\ \mathit{bop}\ z \\
 & | & x = y \\
 & | & x = n \\
 & | & \texttt{goto}\ L \\
 & | & \texttt{if}\ x\ \texttt{goto}\ L \\
\mathit{bop} & \to & \texttt{+} \mid \texttt{<}
\end{array}
$$

The semantics of T should be clear from what we discussed in class.

Define a translator

$$\mathsf{trans} : C \to T$$

that takes a program in C and converts it to a semantically equivalent T program.

$$
\begin{array}{rcl}
\mathsf{trans}_e(n) & = & (t, [t = n]) \\
\mathsf{trans}_e(x) & = & (t, [t = x]) \\
\mathsf{trans}_e(E_1 + E_2) & = & \text{let } (t_1, \mathit{code}_1) = \mathsf{trans}_e(E_1) \\
 & & \text{let } (t_2, \mathit{code}_2) = \mathsf{trans}_e(E_2) \\
 & & \text{in } (t_3, \mathit{code}_1 @ \mathit{code}_2 @ [t_3 = t_1 + t_2]) \\
\\
\mathsf{trans}_b(\texttt{true}) & = & (t, [t = 1]) \\
\mathsf{trans}_b(\texttt{false}) & = & (t, [t = 0]) \\
\mathsf{trans}_b(E_1 < E_2) & = & \text{let } (t_1, \mathit{code}_1) = \mathsf{trans}_e(E_1) \\
 & & \text{let } (t_2, \mathit{code}_2) = \mathsf{trans}_e(E_2) \\
 & & \text{in } (t_3, \mathit{code}_1 @ \mathit{code}_2 @ [t_3 = t_1 < t_2]) \\
\\
\mathsf{trans}(\texttt{skip}) & = & [\texttt{skip}] \\
\mathsf{trans}(x = E) & = & \text{let } (t_1, \mathit{code}_1) = \mathsf{trans}_e(E) \\
 & & \text{in } \mathit{code}_1 @ [x = t_1] \\
\mathsf{trans}(\texttt{while}\ E\ C) & = & \text{let } (t_1, \mathit{code}_1) = \mathsf{trans}_e(E) \\
 & & \text{in } \mathit{code}_b = \mathsf{trans}(C) \\
 & & [(l_e, \texttt{skip})] @ \\
 & & \quad \mathit{code}_1 @ \\
 & & \quad [\texttt{if}\ t_1\ \texttt{goto}\ l_b] @ \\
 & & \quad [\texttt{goto}\ l_x] \\
 & & \qquad [(l_b, \texttt{skip})] @ \\
 & & \qquad \mathit{code}_b @ \\
 & & \qquad [\texttt{goto}\ l_e] \\
 & & \quad [(l_x, \texttt{skip})] \\
\mathsf{trans}(C_1; C_2) & = & \mathsf{trans}(C_1) @ \mathsf{trans}(C_2)
\end{array}
$$

Problem True/false questions:

1. A C compiler can be implemented in C.
2. The type of the function L in Problem 1 is $L \in R \to 2^{\Sigma^*}$.
3. The language of regular expressions (over some alphabet Σ) can be expressed by a regular expression.
4. The language of HTML can be parsed by regular expressions.
5. Context-free grammars are regular expressions with recursion.
6. Regular expression $c^* a(a \mid b \mid c)^*$ describes the strings over alphabet $\{a, b, c\}$ where the first a precedes the first b.
7. There is a language that is context-free but not regular.
8. The ϵ-closure of NFA states I is defined as the smallest set such that
$$I \cup \bigcup_{s \in T} \delta(s, \epsilon) \subseteq T$$
9. $\mathit{fix}(\lambda X.((X - \{1, 2, 3\}) \cup \{1\})) = \{1\}$.
10. Every inductively defined object has an equivalent fixed point definition.
11. The following language is in LR(k) for some k.
$$
\begin{array}{rcl}
S & \to & i\ E\ t\ S\ S' \mid a \\
S' & \to & e\ S \mid \epsilon \\
E & \to & b
\end{array}
$$
12. Every bottom-up parser constructs a parse tree following the rightmost derivation in reverse.
13. In bottom-up parsing, a handle is always found at the leftmost substring of a right sentential form.
14. If a context-free grammar is unambiguous, every right-sentential form of the grammar has exactly one handle.
15. The following language is in LL(1):
$$
\begin{array}{rcl}
E & \to & E + T \\
E & \to & T
\end{array}
$$
16. There is one-to-one relationship between parse trees and derivations.
17. An ambiguous grammar is one that produces more than one rightmost derivation for the same sentence.
18. Consider the expression grammar:
$$E \to E + E \mid E * E \mid (E) \mid \mathbf{id}$$
The SLR parsing for string $\mathbf{id}+\mathbf{id}*\mathbf{id}$ encounters the following shift/reduce conflict:

Stack	Input	Action
$E + E$	$*\mathbf{id}$	shift or reduce

Assuming that $*$ takes precedence over $+$, the correct action here is to take the reduce action.
19. Automatic translations between programming languages are *always* done recursively on the structure of the source language.
20. In static single-assignment form, a variable definition (e.g., $x = 1$) can be executed many times at runtime.