Name:

Directory ID:

University ID:

Midterm 1

CMSC 430
Introduction to Compilers
Fall 2018

Instructions

This exam contains 12 pages, including this one. Make sure you have all the pages. Write your name, directory ID, and university ID number on the top of this page, and write your directory ID at the bottom left of *every* page, before starting the exam.

Write your answers on the exam sheets. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.

If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn't need to do this at all, so be careful when making assumptions.

Question	Score	Max
1		30
2		35
3		35
Total		100

Question 1. Short Answer (30 points).

a. (3 points) What do the letters L, R, and K refer to in $LR(K)$ parsing terminology?

Answer:
L – consume input *left* to right
R – produces a *rightmost* derivation
K – lookahead K tokens

b. (1 point each) Circle **true** or **false** for each statement.

(i.) If a given grammar is $LR(1)$ it is also $LR(0)$. **true** **false**

(ii.) If a given grammar is $LR(0)$ it is also $SLR(1)$. **true** **false**

(iii.) If a given grammar is $LR(1)$ it is also $LALR(1)$. **true** **false**

(iv.) It is possible for a grammar to be $LL(1)$ but not $LR(1)$ **true** **false**

(v.) Ocamlyacc is a parser generator for $LR(1)$ grammars **true** **false**

Answer: (i.) false, (ii.) true, (iii.) false, (iv.) false, (v.) false

c. (6 points) Briefly describe what a language virtual machine is and give two reasons why a language developer might implement one.

Answer: A language virtual machine is a software implementation of a low-level, concrete machine interpreter. It is often used as the target for a compiler from a higher-level language. Virtual machines are useful because they abstract low-level details of real hardware, thereby allowing the compiler writer to focus on generating code that is portable across any platform for which the VM runtime has been implemented. It also provides a convenient platform for implementing different front-end languages that will execute, and interoperate, on a common, portable runtime. Additionally, the bytecode semantics may be closer to the higher-level language than a real machine, thereby simplifying the translation that is required by the compiler. For example, the JVM bytecode operates directly on objects and methods, which are typically not abstractions that are present in physical processors.

d. (2 points) Briefly describe one difference (other than syntactic) between polymorphic and non-polymorphic variants in OCaml.

Answer:

- With polymorphic variants, you do not need to specify all possible tags a priori.
- With polymorphic variants, you can reuse the same tag in different type definitions.

e. (1 point) What data structure or internal representation (IR) does a parser typically produce?

Answer: Abstract Syntax Tree

f. (2 points) In project 2, during which phase of the compiler were comments removed?

Answer: Lexer (credit also given for Parser)

g. (4 points) Briefly list two techniques to resolve an ambiguous grammar so that it can be parsed with ocamlyacc.

Answer:

- Operator associativity (e.g., `%left`)
- Rewrite the grammar so it is not ambiguous, e.g., by adding more non-terminals
- Specifying operator precedence via directives.

h. (2 points) In class, we studied the big-step semantics for the IMP language. Arithmetic expressions evaluate to numbers. Boolean expressions evaluate to boolean values (true or false). What kind of values do commands evaluate to in the big step semantics? As a reminder, the grammar for IMP is:

$$
\begin{array}{l}
a ::= n \mid X \mid a + a \mid a - a \mid a \times a \\
b ::= true \mid false \mid a = a \mid a \leq a \mid \neg b \mid b \wedge b \mid b \vee b \\
c ::= \text{skip} \mid X := a \mid c; c \mid \text{if } b \text{ then } c \text{ else } c \mid \text{while } b \text{ do } c
\end{array}
$$

Answer: A new state (sigma).

i. (5 points) Complete the grammar rules (inside the curly braces) in the following `calc.mly` file such that the main function returns the evaluated expression. Then, make the changes necessary to add support for subtraction.

```
                          /* calc.mly */
%token <int> INT
%token EOL PLUS LPAREN RPAREN SUB
%start main
%type <int> main
%%
main:
| expr EOL              {           $1                      }


expr:
| term                  {           $1                      }

| expr PLUS term        {           $1 + $3                 }
| expr SUB term         {           $1 - $3                 }

term:
| INT                   {           $1                      }

| LPAREN expr RPAREN    {           $2                      }
```

Question 2. Parsing (35 points).
a. (15 points) Consider the following grammar and associated parsing table.

0. $S \rightarrow E\$$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow n$
4. $T \rightarrow (E)$

	Action					Goto		
State	\$	)	(	n	+	S	T	E
0			$s5$	$s4$		3	2	1
1	$r0$				$s7$			
2	$r2$	$r2$			$r2$			
3	acc							
4	$r3$	$r3$			$r3$			
5			$s5$	$s4$			2	6
6		$s9$			$s7$			
7			$s5$	$s4$			8	
8	$r1$	$r1$			$r1$			
9	$r4$	$r4$			$r4$			

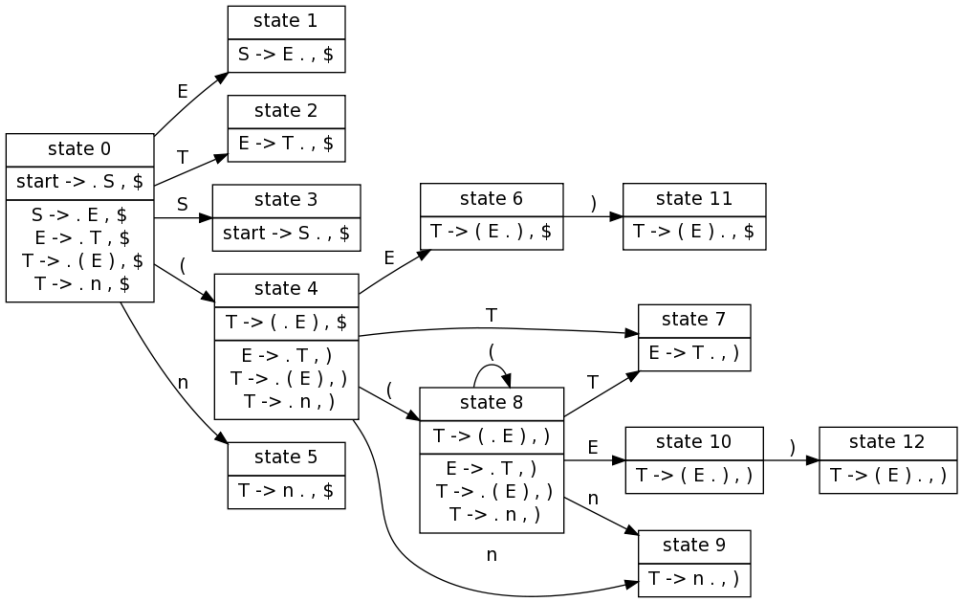
Fill in the following table to show how the string $(7) + (1 + +4)\$$ is parsed. Note that the input is **not** valid with respect to the grammar. Complete the table until you reach a parser error and clearly indicate where the error is detected. You may or may not need to use all the rows. Add extra rows if necessary.

Stack	Input	Action
0	$(7) + (1 + +4)\$$	$s5$
0 (5	$7) + (1 + +4)\$$	$s4$
0 (5 7 4	$) + (1 + +4)\$$	$r3$
0 (5 T 2	$) + (1 + +4)\$$	$r2$
0 (5 E 6	$) + (1 + +4)\$$	$s9$
0 (5 E 6) 9	$+(1 + +4)\$$	$r4$
0 T 2	$+(1 + +4)\$$	$r2$
0 E 1	$+(1 + +4)\$$	$s7$
0 E 1 + 7	$(1 + +4)\$$	$s5$
0 E 1 + 7 (5	$1 + +4)\$$	$s4$
0 E 1 + 7 (5 1 4	$+ + 4)\$$	$r3$
0 E 1 + 7 (5 T 2	$+ + 4)\$$	$r2$
0 E 1 + 7 (5 E 6	$+ + 4)\$$	$s7$
0 E 1 + 7 (5 E 6 + 7	$+4)\$$	**ERROR**
	\$	
	\$	

b. (15 points) Construct the $LR(1)$ DFA for the following grammar:

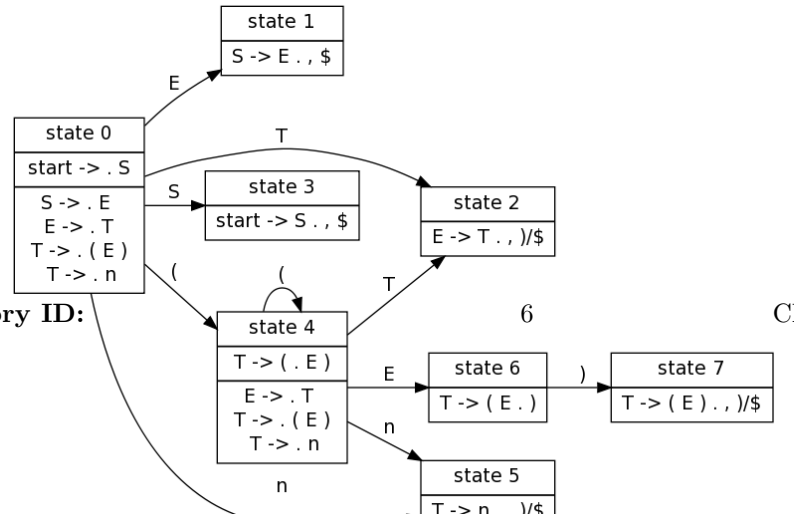
$$
\begin{array}{lrcl}
0. & S & \rightarrow & E\$ \\
1. & E & \rightarrow & T \\
2. & T & \rightarrow & (E) \\
3. & T & \rightarrow & n
\end{array}
$$

Answer:



c. (5 points) Briefly describe the difference between an $LR(1)$ DFA and an $LALR(1)$ DFA. Give an example from your $LR(1)$ DFA above.

Answer: An $LALR(1)$ DFA has fewer states than an $LR(1)$ DFA because the former merges states from the latter where the core items are identical but the lookaheads are different. In the above example, states 5 and 9 can be combined because they contain the same core items `T -> n .` but different lookaheads. The resulting state has the same item twice, with both possible lookaheads ("`)`" and "`$`"). States 11/12, 4/8, 6/10, and 2/7 can also be combined. The resulting DFA looks like this:



Question 3. Operational Semantics (35 points).
a. (10 points) Here are partial big-step operational semantics for arithmetic expressions

$$a ::= n \mid X \mid a + a \mid a - a$$

where $X \in \mathit{Var}$ ranges over variables, and a program state $\sigma : \mathit{Var} \to n$ maps variables to integers n.

$$\textsc{Int}\ \frac{}{\langle n, \sigma\rangle \to n} \qquad \textsc{Var}\ \frac{}{\langle X, \sigma\rangle \to \sigma(X)}$$

$$\textsc{Plus}\ \frac{\langle a_1, \sigma\rangle \to n \qquad \langle a_2, \sigma\rangle \to m \qquad p = n + m}{\langle a_1 + a_2, \sigma\rangle \to p} \qquad \textsc{Minus}\ \frac{\langle a_1, \sigma\rangle \to n \qquad \langle a_2, \sigma\rangle \to m \qquad p = n - m}{\langle a_1 - a_2, \sigma\rangle \to p}$$

Draw a derivation showing that $\langle (X + 5) - (Y - 2), \sigma\rangle \to 5$ if $\sigma = [X \mapsto 6, Y \mapsto 8]$.

Answer:

$$\dfrac{\dfrac{\dfrac{}{\langle X, \sigma\rangle \to 6} \qquad \dfrac{}{\langle 5, \sigma\rangle \to 5} \qquad 11 = 6 + 5}{\langle X + 5, \sigma\rangle \to 11} \qquad \dfrac{\dfrac{}{\langle Y, \sigma\rangle \to 8} \qquad \dfrac{}{\langle 2, \sigma\rangle \to 2} \qquad 6 = 8 - 2}{\langle Y - 2, \sigma\rangle \to 6} \qquad 5 = 11 - 6}{\langle (X + 5) - (Y - 2), \sigma\rangle \to 5}$$

b. (8 points) Here are partial small-step semantics rules for the same language:

$$\textsc{Var}\ \frac{\sigma(X) = n}{X \to_\sigma n} \qquad \textsc{Right+}\ \frac{a_2 \to_\sigma a_2'}{n + a_2 \to_\sigma n + a_2'} \qquad \textsc{Left+}\ \frac{a_1 \to_\sigma a_1'}{a_1 + a_2 \to_\sigma a_1' + a_2} \qquad \textsc{Plus}\ \frac{p = n + m}{n + m \to_\sigma p}$$

Write the missing rules for subtraction. Your rules should **evaluate the right-hand side before the left-hand side**.

Answer:

$$\textsc{Right-}\ \frac{a_2 \to_\sigma a_2'}{a_1 - a_2 \to_\sigma a_1 - a_2'} \qquad \textsc{Left-}\ \frac{a_1 \to_\sigma a_1'}{a_1 - n \to_\sigma a_1' - n} \qquad \textsc{Minus}\ \frac{p = n - m}{n - m \to_\sigma p}$$

c. (5 points) Show that $(X + 5) - (Y - 2) \to_\sigma^* 5$ by showing each step of the reduction, where $\sigma = [X \mapsto 6, Y \mapsto 8]$. You don't need to show the derivations that lead to the individual steps, just the steps themselves. (The relation $a \to_\sigma^* a'$ just means a gets to a' in zero or more steps of $\to_\sigma$.)

Answer:

$$\begin{array}{lll} (X + 5) - (Y - 2) & \to_\sigma & (X + 5) - (8 - 2) \\ & \to_\sigma & (X + 5) - 6 \\ & \to_\sigma & (6 + 5) - 6 \\ & \to_\sigma & 11 - 6 \\ & \to_\sigma & 5 \end{array}$$

d. (2 points) The following is an OCaml type definition for arithmetic expressions. Construct an OCaml expression that represents the abstract expression $(X + 5) - (Y - 2)$.

```
type aexpr =
  | Int of int
  | Var of string
  | Plus of aexpr * aexpr
  | Minus of aexpr * aexpr
```

Answer:

```
Minus(Plus(Var "X", Int 5), Minus(Var "Y", Int 2))
```

e. (5 points) Write an OCaml function `aeval` that evaluates arithmetic expressions using the above big step semantics and `aexpr` type definition. Also write the type signature for your function. Use association lists to represent σ (i.e., `type sigma = (string*int) list`). You may use standard library functions, if necessary.

Answer:

```
val aeval : aexpr -> (string*int) list -> int

let rec aeval a s =
    match a with
      | Int n -> n
      | Var x -> List.assoc x s
      | Minus(a0,a1) -> (aeval a0 s) - (aeval a1 s)
      | Plus(a0,a1) -> (aeval a0 s) + (aeval a1 s)
```

f. (5 points) Write an OCaml function `aevals` that evaluates arithmetic expressions using the above small step semantics (including your extension) and `aexpr` type definition. Also write the type signature for your function. Use association lists to represent σ (i.e., `type sigma = (string*int) list`). You may use standard library functions, if necessary.

Answer:

```
val aeval : aexpr -> (string*int) list -> aexpr

let rec aeval a s =
    match a with
      | Int x -> Int x
      | Var x -> Int(List.assoc x s)
      | Minus(Int n, Int m) -> Int(n-m)
      | Minus(a0, Int n) -> Minus((aeval a0 s), Int n)
      | Minus(a0, a1) -> Minus(a0, (aeval a1 s))
      | Plus(Int n, Int m) -> Int(n+m)
      | Plus(Int n, a1) -> Plus(Int n, (aeval a1 s))
      | Plus(a0, a1) -> Plus((aeval a0 s), a1)
```

This page is intentionally blank for extra work space. If you want the work on this page to count, clearly label which question you are answering and write "see back page" in the answer space for the question.

This page is intentionally blank for extra work space. If you want the work on this page to count, clearly label which question you are answering and write "see back page" in the answer space for the question.