Name:

Midterm 1

CMSC 430
Introduction to Compilers
Fall 2012

October 10, 2012

Instructions

This exam contains 9 pages, including this one. Make sure you have all the pages. Write your name on the top of this 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		20
2		42
3		16
4		22
Total		100

Question 1. Short Answer (20 points).

a. (7 points) Briefly explain the difference between *big-step* and *small-step* operational semantics. What kinds of programs can we assign meaning to using a small-step semantics that are difficult to assign meaning to using a big-step semantics? (You might find it useful to refer to IMP in your discussion.)

b. (7 points) Explain briefly how LALR(1) parsing differs from LR(1) parsing, and give one advantage and one disadvantage of LALR(1) as compared to LR(1).

c. (6 points) Consider the data type for lambda expressions from Project 1:

```
type expr =
  | Var of string
  | Lam of string * expr
  | App of expr * expr
```

Write a function `free_vars : expr -> string list` that returns the list of free variables in a lambda expression.

Question 2. Parsing (42 points).

a. (10 points) Consider the following grammar and the corresponding action and goto tables.

	S'	$\rightarrow$	S
0.	S	$\rightarrow$	ABb
1.	A	$\rightarrow$	ε
2.	A	$\rightarrow$	aA
3.	B	$\rightarrow$	b
4.	B	$\rightarrow$	cB

	action				*goto*		
state	a	b	c	\$	A	B	S
0	$s1$	$r1$	$r1$		3		9
1	$s1$	$r1$	$r1$		2		
2			$r2$				
3		$s7$	$s8$			4	
4		$s5$					
5				$r0$			
6		$r4$					
7		$r3$					
8		$s7$	$s8$			6	
9				*accept*			

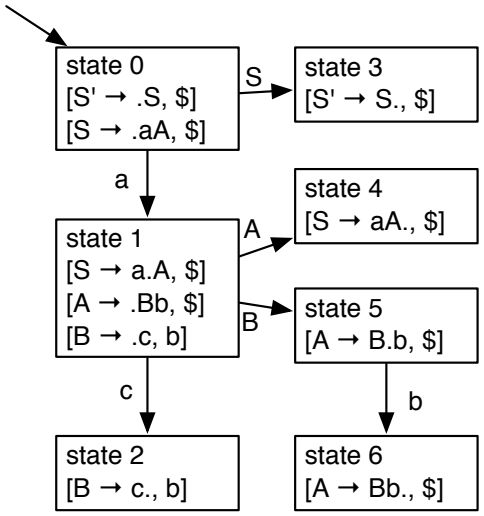
Fill in the following table with the steps taken by the parser on the given input string. Note that we have explicitly written in the end-of-file character, \$, at the end of the input, which we omitted on Homework 1. (You may not need to use all the lines below.)

Stack	Input	Action
0	*accbb*\$	

b. (14 points) Draw the LR(1) parsing DFA for the following grammar.

$$\begin{array}{lcl} S & \rightarrow & aSb \\ S & \rightarrow & c \end{array}$$

c. (12 points) Write down the grammar and the action and goto tables corresponding to the following LR(1) parsing DFA.



d. (6 points) Write down one production such that, if it were added to the grammar to part c just above, it would introduce a shift/reduce conflict. Justify your answer by describing which state would change and how it would exhibit the conflict.

Question 3. Operational semantics (16 points). Consider extending the call-by-value lambda calculus with exceptions. To keep things simple, we will have one exceptional value, *error*, that will propagate through evaluation until caught by a *try e with e* handler (or until it reaches the top level of the expression).

$$\begin{array}{lcl} e & ::= & v \mid x \mid e\ e \mid \mathit{error} \mid \mathit{try}\ e\ \mathit{with}\ e \\ v & ::= & n \mid \lambda x.e \end{array}$$

$$\textsc{Beta}\ \frac{}{(\lambda x.e_1)\ v_2 \to e1[x \mapsto v_2]} \qquad \textsc{Left}\ \frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \textsc{Right}\ \frac{e_2 \to e_2'}{v\ e_2 \to v\ e_2'} \qquad \textsc{Err-left}\ \frac{e_1 \to \mathit{error}}{e_1\ e_2 \to \mathit{error}} \qquad \textsc{Err-right}\ \frac{e_2 \to \mathit{error}}{v\ e_2 \to \mathit{error}}$$

$$\textsc{Try-v}\ \frac{}{\mathit{try}\ v\ \mathit{with}\ e \to v} \qquad \textsc{Try-err}\ \frac{}{\mathit{try}\ \mathit{error}\ \mathit{with}\ e \to e} \qquad \textsc{Try-ctxt}\ \frac{e_1 \to e_1'}{\mathit{try}\ e_1\ \mathit{with}\ e_2 \to \mathit{try}\ e_1'\ \mathit{with}\ e_2}$$

a. (10 points) What is the normal form of *try* $(\lambda x.\mathit{try}\ x\ 3\ 4\ \mathit{with}\ 5)\ (\lambda y.\mathit{error})$ *with* 6 under these semantics? Show each step of evaluation to justify your answer. (You need not show the derivations underlying each step, but do show the final $e \to e'$ steps.)

b. (6 points) Suppose we consider *error* to be a value, i.e., we modify the production for v to be $v ::= n \mid \lambda x.e \mid \mathit{error}$. Give a program that could behave differently, and unexpectedly (compared to typical implementations of exceptions), with this change, and explain your answer.

Question 4. Type checking (22 points). Now consider type checking the language from Question 3. Here is the language again, with type annotations this time:

$$\begin{array}{rcl} e & ::= & v \mid x \mid e\ e \mid \mathit{error} \mid \mathit{try}\ e\ \mathit{with}\ e \\ v & ::= & n \mid \lambda x : t.e \\ t & ::= & \mathit{int} \mid t \to t \\ A & ::= & \cdot \mid x : t, A \end{array}$$

Here are the four standard type rules for simply typed lambda calculus.

$$\textsc{Int}\ \frac{}{A \vdash n : \mathit{int}} \qquad \textsc{Var}\ \frac{x \in \mathit{dom}(A)}{A \vdash x : A(x)} \qquad \textsc{Lam}\ \frac{x : t, A \vdash e : t'}{A \vdash \lambda x : t.e : t \to t'} \qquad \textsc{App}\ \frac{A \vdash e_1 : t \to t' \qquad A \vdash e_2 : t}{A \vdash e_1\ e_2 : t'}$$

a. (10 points) Let $B = x : \mathit{int}, y : \mathit{int} \to \mathit{int}$. What type does $\lambda x : \mathit{int} \to \mathit{int}.\lambda z : \mathit{int}.x\ (y\ z)$ have in environment B? Justify your answer by drawing a derivation showing the typing holds. (To save some writing, you can use i instead of *int*. Also feel free to introduce new abbreviations, e.g., C, D, E, for environments, as needed.)

b. (6 points) Write down the most general possible type rule for *try* e_1 *with* e_2, and explain briefly why your rule is correct.

c. (6 points) Write down the most general possible type rule for *error*, and explain why your rule is correct. (Hint: think about what type `raise exn` has in OCaml, for some exception `exn`.)