Entry: 2012(\$10253 Gp: 3

CSL302

Indian Institute of Technology Delhi Department of Computer Science and Engineering Programming Languages

Minor I

February 7, 2014

Open notes. Write your name, entry number and group at the top of each sheet in the blanks provided. Answer all questions in the space provided, in blue or black ink (no pencils, no red pens). Budget your time according to

16:00-17:00

Q0. (1x4 marks) Lexical and syntactic analysis. Hopefully, you have been programming in OCaml. Write an example each of simple but incorrect OCaml expressions that will produce

I a lexical error detected by the scanner: dete toye var = Van of itn ; hat leader

2. a syntax error, detected by the parser:

8. a type error, detected by the type checker:

Antaofran Vass of antij

a run-time error, detected during program execution:

- Q1. (8 marks) Compilers and Interpreters. Byte code is the code that is interpreted on a virtual machine, whereas native code is machine code for a physical machine, and source code is program code written in a high-level programming language. Remember a program in source code cannot run, unless it is translated into byte code to be run on a virtual machine that is running on a physical machine, or else translated into native code for the physical machine. Suppose I have a Maces machine and a Wintel machine, as well as the following software objects:
 - 1. The source code, written in language j, of a compiler jc; jc takes programs in language j, and produces native code for a Macos machine;
 - 2. A compiler jbc already compiled into byte code language B; jbc takes programs in language j and produces code in byte code B;
 - 3. A byte-code B-interpreter BI wantel (this interpreter is in native code), which runs on a Wintelmachine, which takes code in B and interprets it.

Using the notation X(Y) to mean running the software entity X on input Y, and in particular BI Wintel (X) to denote running the byte-code interpreter BI Wintel to interpret (execute) the B program called X, indicate the steps by which, following a small variant on boot-strapping, I can produce a native code compiler for language j programs, which runs on Macos-machines (and produces native Macos-machine code). Name the output objects (files/programs/...) of each step appropriately.

Step 1.

Step 2.

Q2. (8 marks) Type checking Consider the simple OCaml program filter, and determine its type to solving the various type constraint equations. I have already provided you with the typing constraints for various sub-expressions. You have to formulate the equations between the various type expressions, and solve them.

let rec filter p 1 = match 1 with spit c (if [] -> [] | (x::zs) -> if (p x) then x :: (filter p xs) else filter p xs

(P)

 $\begin{array}{l} 1:\beta \\ rhs:\gamma \\ \\ lhs\; []:\delta\; \mathrm{list} \\ \\ rhs\; []:\epsilon\; \mathrm{list} \end{array}$

p: a

xs:θ list

x : 0

(p x) : bool

filter p xs : 6

filter:

p-(suin-ecia) ((p-10) sup 6) 10

Q3. (6 marks) Lexical Analysis. Consider a simplified version of datatype definitions in OCaml. Such a definition consists of the keyword type followed by an identifier starting with a lowercase letter (the datatype being defined), then an "=" sign followed by a series of cases separated by the vertical faar "|". Each case has the following form: an identifier (the constructor name, starting with an Uppercose letter), then the key word of, followed by a type expression. A type expression r is either an identifier or a (cartesian) product of type expressions r₁ * . . . * r_n. Parentheses may be used around type expressions, to indicate associativity. The definition terminates with a double semicolon.

Write down a specification of the tokens that may appear in a type definition in a lex like notation.

country to type exp = label of int | Mult of Mint tint | Where of easiel four

gymbol: (3), ()

keywords: type, of, int

usu's choice : esp, Label, Mutt, Value, Cube

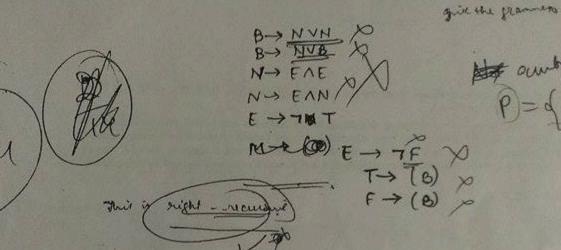




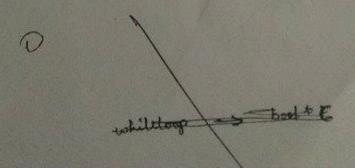
Q4. (14 marks) Context Free Grammars. Consider the following grammar for boolean expressions:

$$B := p \mid T \mid F \mid \neg B \mid B \vee B \mid B \wedge B \mid (B)$$

where p is an element from a set of propositional letters P considered terminals. This grammar is ambiguous. Assume we wish to avoid unnecessary parenthesization so that parenthesization has highest priority, then not binds tighter than A, which in turn binds tighter than V, and both A and V are right associative. Provide an unambiguous grammar that is not left-recursive for this language



- Q5. (6+4 marks) Abstract syntax. Consider a simple imperative language consisting of commands c which can be: (i) Assignment statements of the form x := e, where x is an identifier and e is a numerical expression, (ii) Sequential composition of commands, (iii) A conditional "if-then-else" construct, with a boolean test and two branches, and (iv) An iterative "while loop", with a boolean test and a body which is a command. Provide the following:
 - 1. An abstract syntax for commands, assuming you have abstract syntax for identifiers, expressions and boolean expressions. (Present it as an abstract grammar).



 Encode the above abstract syntax as an OCaml datatype assuming datatypes ident, exp. and bexp.
 type command =



Q6. (5+5 marks) Σ-homomorphisms. Consider the following signature Σ = {0⁽⁰⁾, 1⁽⁰⁾, +⁽³⁾}. Let T_Σ the the corresponding term algebra.

In the following, complete the following definitions of the functions odd (which returns true if the input represents an odd number, and false otherwise) and post (which returns a post-order traversal of the term/tree), such that they are Σ -homomorphisms to the indicated Σ -algebras B and C respectively. Clearly define the meanings of the operations Φ and #, which represent $+_B$ and $+_C$ respectively.

(i) Let
$$\mathcal{B} = (\mathfrak{A}, F, T, \oplus)$$

$$odd(0) = T$$

$$odd(1) = F$$

$$odd(n_1 + n_2) = (\mathfrak{O}(\mathfrak{A}_{12}, \mathfrak{O}(\mathfrak{A})) \oplus (\operatorname{orb}((\mathfrak{n}_1), \operatorname{odd}(\mathfrak{n}_2))$$

where $\Phi(x,y) = Computer add (n) and add(y) are which are both freedom, and the approximation of mother of bootens.

(11) The C = ((0,1,+)*, "0","1",#).$

(ii) Let
$$C = (\{0, 1, +\}^*, {}^{\circ}0^{\circ}, {}^{\circ}1^{\circ}, \#)$$
.

 $post(0) =$
 $post(1) =$
 $post(n_1 + n_2) =$

where $\#(x, y) =$