CSE 3401 – Functional and Logic Programming, Fall 2008
Department of Computer Science and Engineering
York University

Take-Home Final Examination

7 p.m., Monday, February 23, 2009 – 7 p.m., Tuesday, March 3, 2009
There are 4 pages in this exam.

INSTRUCTIONS:

- This take-home exam contains 3 questions.
- Submit your solutions electronically by 7 p.m., Tuesday, March 3, 2009. No late submissions will be accepted. Submission instructions are provided for each question.
- The exam is open book. You may use course materials and other on-line and in-print sources.
- While the exam period is in effect, you may not communicate about this exam in any form with any human being (except me). No cheating will be tolerated.

Question 1

(20 points) In the class we have defined the union `formula`, used to represent propositional formulas, in the following way:

```
type formula =
    Var of string
  | Not of formula
  | And of formula * formula
  | Or of formula * formula
;;
```

(a) *(5 points)* Write a recursive function `postorder` *op f*, which returns the result of the postorder application of the function *op* to the formula *f* and all its subformulas (i.e. for every subformula f' of f, *op* is first applied to the subformulas of f', and then to f' itself). Your function should have the type `(formula -> formula) -> formula -> formula`.

(b) *(5 points)* Write a recursive function `preorder` *op f*, which returns the result of the preorder application of the function *op* to the formula *f* and all its subformulas (i.e. for every subformula f' of f, *op* is first applied to f', and then to the subformulas of opf'). Your function should have the type `(formula -> formula) -> formula -> formula`.

(c) *(5 points)* Write a function `to_nnf` f, which returns the result of the conversion of the formula f to the negation normal form (NNF). Your function should only use the functions `de_morgan` and `double_neg` we defined in the class, and one of the two functions you defined in parts (a) and (b). You function should have the type `formula -> formula`.

(d) *(5 points)* Write a function `to_cnf` f, which returns the result the conversion of the formula f to the conjunctive normal form (CNF). Your function should use only the function `dist_1` we defined in the class, the function `to_nnf` you defined in part (c) and one of the two functions you defined in parts (a) and (b). You function should have the type `formula -> formula`.

Submission instructions: save the four functions you defined in the file named `question1.ml`. Above each function write a few lines of comments that show your understanding of the workings of the function. Submit the file using the following command: `submit 3401 exam question1.ml`

Question 2

(30 points) In this question we continue to work with propositional formulas, and this time we allow the operations of conjunction and disjunction to have an arbitrary number of operands. The union that represents this type of formulas can be defined in the following way:

```
type formula =
    Var of string
  | Not of formula
  | And of formula list
  | Or of formula list
;;
```

For example, the OCaml expression

```
Not (Or [Var "p"; Var "q"; And [Var "r"; Var "s"; Not (Var "p")]])
```

represents the propositional formula $\neg(p \vee q \vee (r \wedge s \wedge \neg p))$. You can assume that the lists of subformulas of `And` and `Or` are always of length 2 or more.

For this question I ask you to research and understand the functionality of two OCaml list functions `List.map` and `List.fold_left`. Using these two functions, implement the following:

(a) *(10 points)* A recursive function `to_string` f, which returns the string representation of the formula f. The string representation of the variables and the

negations is the same as in the class; for the conjunction and disjunction, take the string representations of the operands, conjoin them using the appropriate operand symbol (`&`,`v`) and enclose the result in brackets. For example, the string representation of the formula above is `"-(p v q v (r & s & -p))"`. Your function should have the type `formula -> string`.

(b) *(10 points)* A recursive function `preorder` *op* f, which returns the result of the preorder application of the function *op* to the formula f and all its subformulas (i.e. for every subformula f' of f, *op* is first applied to f', and then to the subformulas of opf'). Your function should have the type `(formula -> formula) -> formula -> formula`.

(c) *(10 points)* A function `de_morgan` f, which implements the generalized De-Morgan rule:

$$\neg(f_1 \wedge f_2 \wedge \cdots \wedge f_n) \equiv \neg f_1 \vee \neg f_2 \vee \cdots \vee \neg f_n,$$
$$\neg(f_1 \vee f_2 \vee \cdots \vee f_n) \equiv \neg f_1 \wedge \neg f_2 \wedge \cdots \wedge \neg f_n.$$

Your function should behave the same way as the function `de_morgan` we defined in the class, that is, if the rule applies to f itself (no recursion into subformulas), return the modified f, otherwise return f unchanged. Your function should have the type `formula -> formula`.

Important: in this question you are *required* to take advantage of the OCaml library functions `List.map` and `List.fold_left`. Any implementation that duplicates the functionality of these functions will be severely penalized.

Submission instructions: save the three functions you defined in the file named `question2.ml`. Above each function write a few lines of comments that show your understanding of the workings of the function. Submit the file using the following command: `submit 3401 exam question2.ml`

Question 3

(20 points) In this question we will work with binary trees whose nodes store integer values. The OCaml definition of the union used to represent such trees is given below:

```
type tree =
    Leaf of int
  | Node of int * tree * tree
;;
```

(a) *(5 points)* Write a recursive function `sum` t, which returns the sum all the values in the tree t. Your function should have the type `tree -> int`.

(b) *(15 points)* Write a function `sum2 t`, which is a *tail-recursive* version of the function `sum` from part (a).

Submission instructions: save the two functions you defined in the file named `question3.ml`. Above each function write a few lines of comments that show your understanding of the workings of the function. Be particularly verbose in part (b). Submit the file using the following command: `submit 3401 exam question3.ml`

Good luck !