

School of Computer Science and Software Engineering

CSE3322 Programming Languages and Implementation

Assignment 2

Due 5pm Thursday 5th September

The purpose of this assignment is to teach more advanced ML programming and an appreciation of functional programming.

Boolean expressions have form

$$y \text{ and } ((\text{not } x) \text{ or } (\text{not } (y \text{ and } \text{false})))$$

Note that x and y are variables: In general variables can be arbitrary strings of characters.

Given a *valuation*, i.e. a function from variable names to Boolean values, we can evaluate a Boolean expression. For example, if we evaluate the above expression with the valuation

```
fn s => if s="x" then false else true
```

we obtain *true*.

A Boolean expression for which there is a valuation which makes it true is said to be *satisfiable*.

We may also simplify Boolean expressions by simplifying “or,” “and” and “not” nodes if they have a constant argument and simplifying nested negations. For example the above expression simplifies to y .

You are to write an abstract data type in Standard ML for representing and manipulating Boolean expressions. You should call the file containing the abstract type `bool.ml`. It should contain the following:

- An `abstype` definition for `BoolExp` a data type representing Boolean expressions. This should have different data constructors for the different kinds of Boolean expressions: *true*, *false*, *not*, *and*, *or* and variables. [1 mark]
- Functions for constructing a Boolean expression:

```
const: bool -> BoolExp
var: string -> BoolExp
lnot: BoolExp -> BoolExp
land: BoolExp * BoolExp -> BoolExp
lor: BoolExp * BoolExp -> BoolExp
```

The function `bool` takes a Boolean constant *true* or *false* and returns the corresponding Boolean expression, `var` takes the name of a variable and returns a Boolean expression variable with that name, while `lnot`, `land` and `lor` combine Boolean expressions with the operators *not*, *and* and *or* respectively. For instance,

```
val b = land( (var "y"), lor( (var "x"), lnot( land((var "y"), (const false)))));
```

binds `b` to the Boolean expression corresponding to the example above. [1 mark]

- A function `BoolExpToString: BoolExp -> string` for returning a string describing a Boolean expression. For instance `BoolExpToString b` should return the string

```
"(y) and ((not(x)) or (not((y) and (false))))"
```

[1 mark]

- A function `evalBoolExp`: `(string -> bool) -> BoolExp -> bool` for evaluating a Boolean expression with a valuation. For instance,

```
evalBoolExp (fn s => if s="x" then false else true) b;
```

should return `true`. [2 marks]

- A function `simplify`: `BoolExp -> BoolExp` for simplifying Boolean expressions. It should perform the following simplifications where B is an arbitrary Boolean expression until no further simplification can be applied.

<code>not(true)</code>	simplifies to	<code>false</code>
<code>not(false)</code>	simplifies to	<code>true</code>
<code>not(not(B))</code>	simplifies to	<code>B</code>
<code>true and true</code>	simplifies to	<code>true</code>
<code>true and B</code>	simplifies to	<code>B</code>
<code>B and true</code>	simplifies to	<code>B</code>
<code>false and B</code>	simplifies to	<code>false</code>
<code>B and false</code>	simplifies to	<code>false</code>
<code>false or false</code>	simplifies to	<code>false</code>
<code>false or B</code>	simplifies to	<code>B</code>
<code>B or false</code>	simplifies to	<code>B</code>
<code>true or B</code>	simplifies to	<code>true</code>
<code>B or true</code>	simplifies to	<code>true</code>

For example,

```
BoolExpToString (simplify b)
```

should return `"y"`. [2 marks]

- A function `vars`: `BoolExp -> string list` for returning a list of the variables occurring in a Boolean expression. Each variable should occur exactly once in the list. For instance, `(vars b)` should return `["x", "y"]` or `["y", "x"]`. [Hint: You may want to use a slightly modified `merge` function from the lecture]. [1 mark]
- A function `satisfiable`: `BoolExp -> bool` which determines if a Boolean expression is satisfiable. For instance, `(satisfiable b)` should return `true`. [Hint: You may want to use `vars` and `eval`]. [2 marks]

Note that you will probably want to use local functions in the above definitions.

Submission Instructions

The above exercises contribute 10% to your total CSE3322 mark.

Submission will be electronic. You should use `/cs/cc/bin/submit` to submit your assignment. To do so login with `ssh` to `ra-clay.cc.monash.edu.au` or `sng.cc.monash.edu.au` and then type `/cs/cc/bin/submit`. `Submit` will ask you for: the course code which is `cse3322`, the assessment code which is `ass2`, your student I.D., and the name of the file to be submitted which should be `bool.ml`. If for some reason you cannot use `submit` then send your answer as an email attachment to `Kim.Marriott@infotech.monash.edu.au`

The assignment is due **5pm Thursday 5th September**.

Your programs will be marked on correctness, style, efficiency and clarity.

Assignments handed in after the due date will attract a late penalty of 5% per day unless special consideration applies or there has been prior agreement in writing from the lecturer. No submission will be accepted after 12 noon Friday 13th of September.

Example

Here is an example SML session using the abstract data type:

```
- use "bool.ml";
[opening bool.ml]
type BoolExp
val BoolExpToString = fn : BoolExp -> string
val const = fn : bool -> BoolExp
val var = fn : string -> BoolExp
val lnot = fn : BoolExp -> BoolExp
val land = fn : BoolExp * BoolExp -> BoolExp
val lor = fn : BoolExp * BoolExp -> BoolExp
val evalBoolExp = fn : (string -> bool) -> BoolExp -> bool
val simplify = fn : BoolExp -> BoolExp
val vars = fn : BoolExp -> string list
val satisfiable = fn : BoolExp -> bool
val it = () : unit

- val b = land( (var "y"), lor( lnot(var "x"), lnot( land((var "y"),
=
      (const false))))));
val b = - : BoolExp

- BoolExpToString b;
val it = "(y) and ((not(x)) or (not((y) and (false))))" : string

- evalBoolExp (fn s => if s="x" then false else true) b;
val it = true : bool
- evalBoolExp (fn s => true) b;
val it = true : bool
- evalBoolExp (fn s => true) b;
val it = true : bool
- evalBoolExp (fn s => false) b;
val it = false : bool

- BoolExpToString (simplify b);
val it = "y" : string

- vars b;
val it = ["x","y"] : string list

- satisfiable b;
val it = true : bool
- satisfiable (land(b, (const false)));
val it = false : bool
```