MP 2 – Verifying Basic Operations on Boolean Expressions

CS 477 – Spring 2018
Revision 1.0

Assigned February 19, 2020
Due February 26, 2020, 9:00 PM
Extension extend 48 hours (penalty 20% of total points possible)

1 Change Log

1.0 Initial Release.

2 Objectives and Background

The purpose of this MP is to test the student's ability to

- To write recursive programs over datatypes in Isabelle;
- write simple proofs about these recursive programs by induction and equational rewriting.

Another purpose of MPs in general is to provide a framework to study for the exam. Several of the questions on the exam will appear similar to the MP problems.

3 Turn-In Procedure

The pdf for this assignment (`mp1.pdf`) and a skeleton version of the file `mp2.thy` for this assignment should be found in the `assignments/mp2/` subdirectory of your git directory for this course. You should put code answering each of the problems below in the file `mp2.thy`. Your completed `mp2.thy` file should be put in the `assignments/mp2/` subdirectory of your git directory (where it was originally found) and committed as follows:

```
git pull
git add mp1.thy
git commit -m "Turning in mp2"
git push
```

Please read the *Instructions for Submitting Assignments* in

`http://courses.engr.illinois.edu/cs477/mps/index.html`

You may find it helpful to read Chapter 2 of Concrete Symantics, which you can find in you git repository at `assignments/resources/concrete_semantics.pdf`, or alternately as **prog-prove** in the Isabelle Documentation sidebar.

4 Boolean Expressions, Evaluation and Substitution

The following is a slightly revised version of the type of boolean expressions presented in class.

datatype *'a boolexp* =
  TRUE
  | *FALSE*
  | *Atom 'a*
  | *Not 'a boolexp*
  | *And 'a boolexp 'a boolexp*
  | *Or 'a boolexp 'a boolexp*
  | *Implies 'a boolexp 'a boolexp*

And the following is an implementation of the Standard Interpretation function defined on page 4 0f `02_prop_log_model.pdf`.

fun *boolexp-eval*
where
  boolexp-eval v TRUE = *True*
 | *boolexp-eval v FALSE* = *False*
 | *boolexp-eval v* (*Atom x*) = *v x*
 | *boolexp-eval v* (*Not b*) = ($\neg$ (*boolexp-eval v b*))
 | *boolexp-eval v* (*And a b*) =
   ((*boolexp-eval v a*) $\wedge$ (*boolexp-eval v b*))
 | *boolexp-eval v* (*Or a b*) =
   ((*boolexp-eval v a*) $\vee$ (*boolexp-eval v b*))
 | *boolexp-eval v* (*Implies a b*) =
   (($\neg$ (*boolexp-eval v a*))$\vee$ (*boolexp-eval v b*))

5 Problems

Below you are asked to define substitution and the set of variables occurring in a proposition. You are additionally asked to prove a few simple properties about these two functions. To prove a theorem, you should first remove the non-proof **oops**

1. (7 pts) Define *boolexp-subst* that, given a propositional atom (x), and two boolexps, creates a third boolexp that is the result of replacing all occurrences of the propositional atom in the second boolexp with the first boolexp.

   You will likely find it helpful to use something like *if x = y then result1 else result2*

fun *boolexp-subst* :: *'a* $\Rightarrow$ *'a boolexp* $\Rightarrow$ *'a boolexp* $\Rightarrow$ *'a boolexp* **where**
 boolexp-subst x b TRUE = *TRUE*

If you have done your work right, the following should give a result of *And* (*Atom ''b''*) (*Implies* (*Atom ''b''*) *TRUE*)

value *boolexp-subst ''a''* (*Implies* (*Atom ''b''*) *TRUE*) (*And* (*Atom ''b''*) (*Atom ''a''*))

2. (3 pts) Prove that if you evaluate with a valuation v a boolexp that is the result of substituting all occurences of a propositional atom x by the proposition *TRUE*, the result is the same as if you had evaluated the original boolexp using the valuation λy. *if y = x then TRUE else v y*

lemma *subst-to-eval*:
boolexp-eval v (*boolexp-subst x TRUE b*) =
 boolexp-eval (λ *y*. *if y = x then True else v y*) *b*
 oops

3. (6 pts) Define a function *atoms-of-boolexp*::*'a boolexp* ⇒ *'a list* that returns a list containing exactly the propositional atoms that occur in the input boolexp. Duplicates are allowed in the list. There is no required order. The empty list is represented by []. To insert an element *x* at the front of a list *lst* use *x* # *lst*. To append list *l2* onto list *l1* use *l1* @ *l2*.

fun *atoms-of-boolexp* :: *'a boolexp* ⇒ *'a list* **where**
atoms-of-boolexp TRUE = []

If your definition in Problem 3 is correct, the following lemma should complete, producing a theorem

lemma *test-problem3*:
set (*atoms-of-boolexp* (*Or* (*Implies* (*Atom* ''b'') *TRUE*) (*And* (*Atom* ''b'') (*Atom* ''a'')))) =
{''a'', ''b''}
oops

4. (3 pts) Prove that if two valuations are the same on the propositional atoms in a boolexp, then they will give the same result when used to evaluate the boolexp.

lemma *same-on-atoms-same-value*:
(∀ *x*. *x* : *set* (*atoms-of-boolexp b*) ⟶ *v1 x* = *v2 x*) ⟶
(*boolexp-eval v1 b* = *boolexp-eval v2 b*)
oops

5. (3 pts) Prove that substituting for an atom that is not in a boolexp yields the same boolexp.

lemma *subst-nonexistant-atom*:
x ∉ *set*(*atoms-of-boolexp b*) ⟶ (*boolexp-subst x b' b* = *b*)
oops