Semantics of Programming Languages
Exercise Sheet 3

Homework 3.1 Extending arithmetic expressions

Submission until Tuesday, November 7, 10:00am.

We define a new type for arithmetic expressions with two changes from $aexp$:

- variables carry an additional constant factor
- a new constructor for negation

```plaintext
datatype mexp =
  N int | Plus mexp mexp |
  Neg mexp |
  V int vname
```

First, define a function $mval$, analogously to $aval$.

```plaintext
fun mval :: "mexp ⇒ state ⇒ val"
value "mval (V 3 "x") <"x":=3> = 9"
value "mval (Neg (N 3)) <> = -3"
```

We now want to optimize these expressions in multiple different ways.

**Simplification** Adapt the $asimp$ function from the lecture that evaluates constant subexpressions and eliminates all occurrences of $mexp.N 0$ in additions. Prove correctness!

**Accumulating variables** In an expression that contains multiple occurrences of a particular variable, all occurrences can be replaced by a single one. For example, the expression $mexp.Plus (mexp.V 3 "x") (mexp.V 2 "x")$ is equivalent to $mexp.V 5 "x"$. Define a function $optimize$ that performs this optimization for one variable and prove its correctness. Furthermore, prove that $optimize$ only contains one single occurrence of the specified variable.

Hints:
Start with a function that accumulates all constant factors for the variable.

For the last lemma, you need to define an auxiliary function that counts occurrences of variables.

You may need more auxiliary functions.

For your proofs, you may need some additional arithmetic facts, that you can pass to the simplifier as follows: apply (auto simp add: algebra_simps)

fun optimize :: “mexp ⇒ vname ⇒ mexp”

Elimination of negation The Neg constructor is unneeded. Provide a function un_neg that removes negation and prove that it does. Also prove correctness.

hint: You have to define a function no_negs that checks that an expression contains no negation.

fun un_neg :: “mexp ⇒ mexp”