Concrete Semantics with Isabelle/HOL

Exercise Sheet 2

This exercise sheet depends on definitions from the file AExp.thy and on real numbers. They can be imported as follows:

```isar
theory ex02
  imports Complex_Main "~/src/HOL/IMP/AExp"
begin
```

Remark: So that a theory can be imported by another or processed from the command line, it is necessary that it ends with the keyword `end`.

Please remember to do so in your exercise sheets.

**Exercise 2.1  Substitution Lemma**

A syntactic substitution replaces a variable by an expression.

Define a function `subst :: vname ⇒ aexp ⇒ aexp ⇒ aexp` that performs a syntactic substitution, i.e., `subst x a′ a` will be the expression `a` in which every occurrence of variable `x` has been replaced by expression `a′`.

Instead of syntactically replacing a variable `x` by an expression `a′`, we can also change the state `s` by replacing the value of `x` by the value of `a′` under `s`. This is called *semantic substitution*.

The *substitution lemma* states that semantic and syntactic substitution are compatible. Prove the substitution lemma:

```isar
lemma subst lemma: "aval (subst x a′ a) s = aval a (s(x := aval a′ s))"
```

Note: The expression `s(x := v)` updates a function at point `x`. It is defined as

```
f(a := b) = (λx. if x = a then b else f x)
```

Compositionality means that one can replace equal expressions by equal expressions. Use the substitution lemma to prove *compositionality* of arithmetic expressions:

```isar
lemma comp: "aval a1 s = aval a2 s ⇒ aval (subst x a1 a) s = aval (subst x a2 a) s"
```
Exercise 2.2  Arithmetic Expressions with Side-Effects and Exceptions

We want to extend arithmetic expressions with the division operation and the postfix increment operation \( x++ \), as known from Java or C++.

The problem with the division operation is that division by zero is not defined. In this case, the arithmetic expression should evaluate to a special value indicating an exception.

The increment can only be applied to variables. The problem is that it changes the state, and the evaluation of the rest of the term depends on the changed state. We assume left-to-right evaluation order.

Define the datatype of extended arithmetic expressions. Hint: If you do not want to hide the standard constructor names from IMP, add a tick (′) to them, e.g., \( V'x \).

The semantics of extended arithmetic expressions has the type \( \text{aval}' :: \text{aexp}' \Rightarrow \text{state} \Rightarrow (\text{val} \times \text{state}) \text{ option} \), i.e., it takes an expression and a state, and returns a value and a new state, or an error value. Define the function \( \text{aval}' \).

Hint: To make things easier, we recommend an incremental approach to this exercise: First define arithmetic expressions with incrementing, but without division. The function \( \text{aval}' \) for this intermediate language should have type \( \text{aexp}' \Rightarrow \text{state} \Rightarrow \text{val} \times \text{state} \). After completing the entire exercise with this version, modify your definitions to add division and exceptions.

Test your function for some terms. Is the output as expected? Note: \(<> \) is an abbreviation for the state that assigns every variable to zero:

\[
<> \equiv \lambda x. 0
\]

\[
\text{value } \text{"aval}' (\text{Div}' (V' ''x'') (V' ''x'')) <>
\]

\[
\text{value } \text{"aval}' (\text{Div}' (\text{PI}' ''x'') (V' ''x'')) <''x'':=1''> 
\]

\[
\text{value } \text{"aval}' (\text{Plus}' (\text{PI}' ''x'') (V' ''x'')) <>
\]

\[
\text{value } \text{"aval}' (\text{Plus}' (\text{Plus}' (\text{PI}' ''x'') (\text{PI}' ''x'')) (\text{PI}' ''x'')) <>
\]

Is the plus operation still commutative? Prove or disprove.

When we’re faced with a “prove or disprove,” we’re usually better off trying first to disprove with a counterexample, for two reasons: A disproof is potentially easier (we need just one counterexample); and nitpicking arouses our creative juices.

— Graham, Knuth, and Patashnik, Concrete Mathematics (1988)

Show that the valuation of a variable cannot decrease during evaluation of an expression:

\[
\text{lemma } \text{aval}'_{\text{inc}}: \text{"aval}' a s = \text{Some} (v,s') \Rightarrow s \ x \leq s' \ x
\]

Hint: If auto on its own leaves you with an if in the assumptions or with a case, you can modify it like this: \( \text{auto split: split_if_asm option.splits} \).
**Exercise 2.3** Variables of Expression

Define a function that returns the set of variables occurring in an arithmetic expression:

```haskell
fun vars :: "aexp ⇒ vname set"
```

Show that arithmetic expressions do not depend on variables that they do not contain:

```haskell
lemma ndep: "x ∈ vars e ⇒ aval e (s(x := v)) = aval e s"
```

**Homework 2.1** An Iterator (5 points)

Submission until Monday 11.05.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE2]” in the subject of your email. If your name is Firstname Lastname, please call your theory file Lastname_Firstname.thy.

Define a function

```haskell
fun iter :: "'(a ⇒ 'a) ⇒ nat ⇒ 'a ⇒ 'a"
```

such that

\[
iter f n x = f \ldots (f x)\ldots \text{ } n \text{ times}
\]

The next task is to show how to implement addition, multiplication, and exponentiation (power) on in terms of `iter`. In other words, find appropriate terms for `Fa`, `Na`, `Xa`, etc., below and prove the resulting lemmas:

```haskell
lemma "((m::nat) + n = iter Fa Na Xa"

lemma "((m::nat) * n = iter Fb Nb Xb"

lemma "((m::nat) ^ n = iter Fc Nc Xe"
```

Note: Answers à la `iter undefined 0 (m + n)` will not be accepted. Obviously, we are expecting `iter` to do the heavy lifting.

Prove the following lemma:

```haskell
lemma "iter f m (iter f n x) = iter f (m + n) x"
```

Finally, show how to implement `fact` (the factorial function, or “Fakultät”) and prove the corresponding correctness lemma.

Hint: This may require some creativity.
Homework 2.2  Polynomials (10 points)

Submission until Monday 11.05.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE2]” in the subject of your email. If your name is Firstname Lastname, please call your theory file Lastname_Firstname.thy.

This homework is about polynomials on real numbers. A polynomial can be represented in Isabelle as a list of real numbers the coefficients. To use real numbers in Isabelle, you must import the theory Complex_Main instead of Main.
For example the list [12, 18, 42, \pi, 0] is the polynomial 12 + 12 \times x + 42 \times x^2 + \pi \times x^3 + 0 \times x^4.

define synonym poly = “real list”

We will need additional simplifications lemmas.

declare algebra_simps

Step A: Define the sum of two polynomials

fun sump :: “poly ⇒ poly ⇒ poly”

Prove that the length of sum p is the maximum of the length of the arguments:

lemma length_sump: “length (sump p q) = max (length p) (length q)”

Observe that the length does not necessarily coincide with the degree of the polynomials, since there might be trailing zeros.

Step B: Define the evaluation of a polynomial at a given point, using the \( \hat{\text{op}} \) operator. The argument of type nat represents the degree of the mononomial we are considering.

fun evalp :: “real ⇒ poly ⇒ nat ⇒ real”

Show that the valuation of the sum is the sum of the valuation:

lemma evalp_sum_eq_sum_evalp[simp]:
“evalp x (sum p q) n = evalp x p n + evalp x q n”

Step C: Define the evaluation of a polynomial based on the Horner method: If \( p(x) = a_0 + a_1 \times x + \cdots + a_n \times x^n \), then \( p(x) = a_0 + x \times (a_1 + x \times (\cdots + a_n \times x)) \)

fun evalpH :: “real ⇒ poly ⇒ real”

Show that the sum of the evaluation is the evaluation of the sum:

lemma evalpH_sum_eq_sum_evalp[simp]:
“evalpH x (sum p q) = evalpH x p + evalpH x q”

Step D: Prove that the two evaluation methods are equal:

lemma “evalpH x p = evalp x p 0”
Homework 2.3  More Polynomials (5 bonus points)

Submission until Monday 11.05.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE2]” in the subject of your email. If your name is Firstname Lastname, please call your theory file Lastname_Firstname.thy.

Note: A bonus point counts on the side of your personal score, but not for the maximal reachable points by which we will divide your personal score to get your percentage.

Step A: We will now compare the two evaluation schemes, by counting the number of multiplications.

By definition \(x^n = x \times x \times \cdots \times x\). Thus it takes \(n\) operations to calculate the power function:

Define the number of multiplications of \(evalp\):

\[
\text{fun } \alpha :: \text{“poly } \Rightarrow \text{nat } \Rightarrow \text{nat”}
\]

Define the number of multiplications of \(evalpH\):

\[
\text{fun } \beta :: \text{“poly } \Rightarrow \text{nat”}
\]

Compare the two function and prove which one has a lower complexity.

Hint: You might need a more general lemma, since \(\alpha\) has two arguments.

Step B: Define multiplication using \((a + x \times p) \times q = a \times q + (0 + x \times p \times q)\). Reuse the \(sump\) function.

Hint: You can use the \(map\) function.

\[
\text{fun } multp :: \text{“poly } \Rightarrow \text{poly } \Rightarrow \text{poly”}
\]

Prove

\[
\text{lemma } evalpH_multp: \text{“evalpH x (multp p q) = evalpH x p \times evalpH x q”}
\]

Step C: We will now evaluate the complexity of the \(sump\) and the \(multp\) function assuming unit time for arithmetic operations.

For example \(sump \begin{bmatrix} 1, 12 \end{bmatrix} \begin{bmatrix} 42, 42, 42 \end{bmatrix} = \begin{bmatrix} 1 + 42, 12 + 42, 42 \end{bmatrix}\) has a cost of 2, since two additions are needed.

\[
\text{fun } \sigma :: \text{“poly } \Rightarrow \text{poly } \Rightarrow \text{nat”}
\]

Hint: The definition should be very close to the definition of \(sump\) (but involving \(\sigma\)).

The complexity of the \(sump\) is the minimum of the length of the two lists.

\[
\text{lemma } \sigma_{\text{min}} : \text{“} \sigma \ p \ q = \text{min} \ (\text{length} \ p) \ (\text{length} \ q) \text{”}
\]
What is the complexity of the polynomial multiplication (still assuming unit time for arithmetic operations)?

fun \( \gamma \) :: “poly \( \Rightarrow \) poly \( \Rightarrow \) nat”

Hint: The definition should be very close to the definition of \( \text{multp} \) (but involving at least \( \sigma, \gamma \)).

Prove the following bound:

lemma “\( \gamma p q \leq (\text{length } p + \text{length } q) \ast (\text{length } p + \text{length } q) \)”

Using the big-O notation from the scientific literature, this bound would be written \( O((\max (\text{length } p) (\text{length } q)) ^ 2) \).

Finally, to make Isabelle ultra-happy:

end