COMS22201: Language Engineering

Lab Exercises - Week 15 - Questions

08/02/2016, csxor@bristol.ac.uk

This worksheet uses the language **While** from Nielson's book (of which you used an extension in Coursework 1) in order to provide you with further practice using grammars, proofs, invariants and Haskell data types.

1. Consider the following grammar which defines a bit as being either on or off and a word as being any non-empty sequence of bits:

$$\begin{array}{lcl} \langle \textit{bit} \rangle & ::= & \text{`}\texttt{on}\text{'} \mid \text{`}\texttt{off}\text{'} \\ \langle \textit{word} \rangle & ::= & \langle \textit{bit} \rangle \mid \langle \textit{bit} \rangle \ \langle \textit{word} \rangle \end{array}$$

   (a) Define a Haskell type `Bit` which has two possible values `On` and `Off`. Ensure your type automatically derives membership of the `Eq`, `Ord`, `Enum`, `Show` and `Read` classes.

   (b) Define a recursive Haskell type `Word` which represents a sequence of `Bits`. Instantiate your type into the `Show` class by writing a function `show` that turns the word representing '`on off on off`' into the string "`1010`", for example.

   (c) Assuming words represent unsigned binary numbers with the *least* significant bit on the left, write a Haskell function that computes the integer value represented a given word (which would be 5 for the word illustrated above).

2. Consider the following grammar defining the arithmetic expressions, Boolean expressions and program statements of the language **While** in terms of numerals and variables:

$$
\begin{array}{lcl}
\langle Aexp \rangle & ::= & \langle Num \rangle \mid \langle Var \rangle \\
& \mid & \langle Aexp \rangle \text{ `+' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ `*' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ `-' } \langle Aexp \rangle \\
\langle Bexp \rangle & ::= & \text{`true'} \mid \text{`false'} \\
& \mid & \langle Aexp \rangle \text{ `}=\text{' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ `}\leq\text{' } \langle Aexp \rangle \\
& \mid & \text{`}\neg\text{' } \langle Bexp \rangle \mid \langle Bexp \rangle \text{ `}\wedge\text{' } \langle Bexp \rangle \\
\langle Stm \rangle & ::= & \langle Var \rangle \text{ `:=' } \langle Aexp \rangle \mid \text{`skip'} \mid \langle Stm \rangle \text{ `;' } \langle Stm \rangle \\
& \mid & \text{`if'} \langle Bexp \rangle \text{ `then'} \langle Stm \rangle \text{ `else'} \langle Stm \rangle \\
& \mid & \text{`while'} \langle Bexp \rangle \text{ `do'} \langle Stm \rangle
\end{array}
$$

(a) Define two Haskell types `Num` and `Var` as synonyms for Haskell's native `Integer` and `String` types, respectively.

(b) Define three Haskell types `Aexp`, `Bexp` and `Stm` to represent the syntactic categories of **While**.

Note that you may have to use the qualified syntax `Main.Num` for numerals to avoid a conflict with Haskell's native `Num` class.

(c) Represent the following program as a Haskell term:

```
z:=x; while (2≤y) do (z:=z*x; y:=y-1)
```

3. Use a loop invariant to prove the program above computes the value of `x` raised to the power of (the initial value of) `y` for all initial $\texttt{x}, \texttt{y} > 0$.