COMP 4161 S2/11
Advanced Topics in Software Verification

Assignment 3

This assignment starts on Thursday, 22.09.2011 and is due on Thursday, 06.10.2011, 23:59h. Submit using `give` on a CSE machine:

```
give cs4161 a3 file1 file2 ...
```

Your submission must include an Isabelle theory (.thy) file, and may also include a text (.txt) or PDF (.pdf) file.

1 Balanced Parentheses (30 marks)

Define and prove the following in Isabelle.

(a) Use an inductive set to define the set of strings over the alphabet of parentheses {`L, R`} that is described by the following grammar where a string is a list of letters and ϵ is the empty string:

$$S := \epsilon \mid \texttt{L}\ S\ \texttt{R} \mid S\ S$$

(b) Prove that the following two strings are balanced according to the grammar: `LRLLRR` and `LLRLLRRR`.

(c) Define a function *is_bal* that counts opening and closing parentheses from left to right and returns true if they are balanced and false if they are not. For example `LRLLRR` is balanced, `LLR` is not.

Evaluate 4 small test cases to gain confidence in the definition.

(d) Show $xs \in S \Longrightarrow is\_bal\ xs\ 0$ (You will need to prove intermediate lemmas.)

2 Optimising Arithmetic Expressions (25 marks)

Consider a small language of arithmetic expressions consisting of natural number constants such as `4`, variables such as `x`, and addition `a + b`.

(a) Write a function `aval a s` that computes the value of expression a in state s. Evaluate the expression `x + 5` in a state where `x` is 3.

(b) Write a function `asimp a t` that performs constant folding on the expression `a`. The function should evaluate all constant arithmetic sub-expressions to their value, e.g. `x + (3 + 5)` should become `x + 8`. The function should also optimise addition with `0`, e.g. `0 + y` should become `y`. Furthermore, assuming the second parameter `t` of the function `asimp` contains a mapping for some variables with known value, the function should also replace these variables with their value, e.g. `asimp (0 + (x + (3 + 2))) t` should become 6 if `t x = Some 1`.

(c) The function `asimp` is correct if for any given state `s` the simplified expression evaluates to the same value as the original expression – provided the parameter `t` agrees with the state `s` for the variables it returns `Some`.

Formulate this correctness statement in Isabelle and prove it.

3 File System Paths (45 marks)

In this question we are going to formalise parts of Isabelle's program code, in particular a simplified implementation of its file system paths.

Paths in most file systems are composed of three kinds of elements: the root of the file system `/`, a parent directory element `..`, and an elementary path segment that is a string. A path is then a list of such elements.

(a) Define a function `str` that converts abstract paths into strings. Separate path elements by `"/"` using the function `sep` from the lecture. For converting path elements to strings, use the function `str_of` defined in the template theory. Recall that strings in Isabelle are just lists of characters.

(b) Define a function `chdir s p` that takes a single path element `s` and a path `p` and that returns the path you would get from executing a change directory command with `s` in directory `p`. To make the definition of this function more convenient, you may want to store path segments from right to left, such that for example, `chdir Parent [Seg a, Seg b, Root] = [Seg b, Root]`, corresponding to `chdir '..' '/b/a' = '/b'`

In some situations, the directory change may not make any sense, e.g. changing to the parent of the root directory. In this case, the result should be recorded, but not executed, i.e. in the example the result should be the path `/..`

(c) Define a function `norm p` that normalises paths such that they can be divided in three parts `xs @ ys @ zs` where `xs` is either empty or the root, `ys` consists only of parent elements and `zs` contains only segment elements. Remember that your representation may be right-to-left, in which case you would exchange `xs` and `zs`. You can (but don't have to) use the function `chdir` in this definition.

(d) Formalise the normal-form predicate from above and prove that the result of `norm` always satisfies it. You will need to prove additional lemmas about `chdir` for this.

(e) (Challenge question, 7 marks) Prove that `norm` does not change a path that is already in normal form.