CS_M36

FUNCTIONAL PROGRAMMING AND INTERACTIVE THEOREM PROVING (PART 2)

(*Attempt* **2** *questions out of 3*)

Question 1.

(a) Define the notions of *normal form*, *weak normalisation* and *strong normalisation* in the context of reduction systems.

[6 marks]

(b) Are there reduction systems which are strongly normalising but not weakly normalising? If yes, give an example of such a reduction system, and explain why it has this property. If no, briefly explain, why no such system exists. Are there reduction systems which are weakly normalising but not strongly normalising? If yes, give an example of such a reduction system, and explain why it has this property. If no, briefly explain, why no such system exists.

[4 marks]

(c) If a reduction system $(A, \longrightarrow)$ is confluent, it is easier to determine whether two elements $a, b \in A$ are equivalent with respect to $\longrightarrow$, i.e. whether $a \longleftrightarrow^* b$ holds. Why is this the case? Give an example of a reduction system which is not confluent but strongly normalising, and explain why it has this property.

[4 marks]

(d) Define a λ-term which has no normal form. Explain, why this is the case. The fact that the untyped λ-calculus is not normalising causes problems when comparing two terms. What is the exact problem? Why would this be a problem, when using it in combination with an interactive theorem prover based on dependent type theory?

[6 marks]

(e) There is a variant of the λ-calculus, which is strongly normalising. How is it called and what changes to the untyped λ-calculus are applied there?

[5 marks]

Question 2.

(a) In type theory, one uses dependent types in order to express properties of a system expressible by a formula. Why can the same not be done in Haskell, which has only non-dependent types?

[2 marks]

(b) Indicate how to express the formulae $A \wedge B$ and $\exists x : C.D(x)$ as types in dependent type theory, provided this has been already done for the formulae A, B, C and $D(x)$.

[4 marks]

(c) All proofs in Agda are constructive. Therefore not all formulae provable in ordinary classical logic are provable in Agda. Give an example of a formula which is provable in classical logic but not in Agda.

[2 marks]

(d) When using Agda in order to guarantee the correctness of the theorems proved, it does not suffice to type check the program – one needs as well to carry out a termination check. Why is this necessary? Illustrate your statement by an example, which passes the type checker, but is a wrong proof and therefore does not pass the termination checker.

[5 marks]

(e) There are two ways of expressing the formula $A \wedge B$ in Agda. Determine both of them depending on parameters A and B.

[4 marks]

(f) For both of your two versions of expressing the formula $A \wedge B$, give an Agda proof of the formula $(A \wedge B) \rightarrow (B \wedge A)$.

[8 marks]

Question 3.

(a) Introduce in Agda the set `N` of natural numbers, the set `Bool′` of Booleans, the set `True′` expressing the always true formula, and the set `False′` expressing the always false formula.

[8 marks]

(b) Introduce in Agda the function

```
atom (b :: Bool′)
     :: Set
     = {! !}
```

which converts a Boolean value *b* into a formula, which is provable, in case *b* is true, and unprovable, if *b* is false.

[2 marks]

(c) Introduce in Agda the Boolean valued functions

```
Eq_bool (n,m :: N)
        :: Bool′
        = {! !}

Less_bool (n,m :: N)
        :: Bool′
        = {! !}
```

Here `Eq_bool n m` should express that `n` and `m` are equal, and `Less_bool n m`, that `n` is less than `m`.

[4 marks]

(d) Convert the Boolean valued functions from the previous question into formulas, i.e. define

```
(==)  (n,m:: N)
      :: Set
      = {! !}

(<)  (n,m:: N)
     :: Set
     = {! !}
```

where `n == m` expresses that `n` and `m` are equal, and `n < m` expresses that `n` is less than `m`.

[4 marks]

(e) State in Agda the lemma expressing that `(<)` commutes with `(==)` on the left side, i.e. that `n == m` and `m < k` implies `n < k`.

[2 marks]

(f) Prove in Agda the lemma stated in part (e).

[5 marks]