Section A

1. Tait’s method for proving strong normalisation uses a notion called reducibility. Consider the simply-typed \( \lambda \)-calculus with constant base type \( B \).

   (a) Give the definition of reducible and neutral terms for this calculus.  
   \[8 \text{ marks}\]

   (b) State the lemma, used in the proof of strong normalisation, relating the properties of reducibility, neutrality and strong normalisability. The lemma is proven by induction on the structure of the types: prove the case for the base type \( B \).  
   \[17 \text{ marks}\]

2. (a) State the rule of subsumption.  
   \[3 \text{ marks}\]

   (b) Give the typing rules for the simply-typed lambda calculus extended with records and subtyping.  
   \[7 \text{ marks}\]

   (c) Give an example of a contravariant type construct in the above language and devise a term which would violate the subject reduction property if the type construct were covariant.  
   \[15 \text{ marks}\]

3. (a) Give the typing rules for System F.  
   \[7 \text{ marks}\]
(b) Give encodings in System F of sum types $T_1 + T_2$ and the term constructs

\begin{align*}
\text{in}_L & : \forall T_1.\forall T_2. T_1 \rightarrow (T_1 + T_2) \\
\text{in}_R & : \forall T_1.\forall T_2. T_2 \rightarrow (T_1 + T_2) \\
\text{case} & : \forall T.\forall T_1.\forall T_2. (T_1 + T_2) \rightarrow (T_1 \rightarrow T) \rightarrow (T_2 \rightarrow T) \rightarrow T
\end{align*}

[15 marks]

(c) Consider the following introduction and elimination rules for recursive types:

\begin{align*}
\Gamma \vdash \text{fold Rec}(X)S : \{ X \mapsto \text{Rec}(X)S \} S \rightarrow \text{Rec}(X)S \\
\Gamma \vdash \text{unfold Rec}(X)S : \text{Rec}(X)S \rightarrow \{ X \mapsto \text{Rec}(X)S \} S
\end{align*}

Are such recursive types encodable in System F? Give reasons for your answer. [3 marks]

Section B

4. (a) Augment the labelled transition system used to formalise CCS semantics to allow for a special action $\kappa$ over which all concurrent agents must synchronise. Thus, for example, $(a.\kappa.0 | \kappa.P | \kappa.Q)$ would only be able to start by performing action $a$, followed by action $\kappa$, after which it behaves like $(0 | P | Q)$.

(b) Prove that the law $P = P | 0$ is no longer valid in the augmented system.

(c) State which agent now acts as the right (and left) one of composition $(|)$.

(d) Prove that $(P | Q | R) \setminus \{\kappa\}$ is strongly bisimilar to $S$ as defined below:

\begin{align*}
P & \overset{\text{def}}{=} b.\kappa.\kappa.Q \\
Q & \overset{\text{def}}{=} a.\kappa.\kappa.R \\
R & \overset{\text{def}}{=} \kappa.c.\kappa.P \\
S & \overset{\text{def}}{=} a.b.c.S + b.a.c.S
\end{align*}

[25 marks]
5. (a) A memory cell can store a single piece of information. It has an output port through which it sends the current value stored in the cell, and an input port from which it may receive a value which it stores (overwriting the old value it was storing). Specify such a cell using the Value Passing Calculus.

(b) Given \(n\) memory cells \(A_1\) to \(A_n\) all storing distinct information, and another \(n\) cells for auxiliary information \((B_1\) to \(B_n\)) the following algorithm selects the minimum value stored in the cells and swaps it with the value in \(A_1\):

i. \(n\) concurrent processes set the auxiliary cells to value 0.

ii. If \(A_i\) has the minimum value stored in the array, \(n^2\) concurrent processes \((P_{1,1}\) to \(P_{n,n}\)) leave value 0 in position \(B_i\) and 1 in all the others: \(P_{i,j}\) reads the value of \(A_i\) and \(A_j\) and if \(A_j\) is the smaller, it writes 1 in \(B_j\).

iii. \(n\) concurrent processes \(P_1\) to \(P_n\) check the auxiliary array. Process \(P_i\) reads \(B_i\) and if it finds 0, it swaps \(A_1\) and \(A_i\).

Specify this algorithm using the Value Passing Calculus.

(c) A simpler algorithm reads the data array sequentially ‘remembering’ the smallest value and its position. After reading the whole array it swaps the value stored in the first position with the minimum value stored in the array.

Specify this algorithm using the Value Passing Calculus.

[25 marks]

6. Recall what we mean by composition of agents \(P \triangleright Q\): we rename certain outputs of \(P\) and certain inputs of \(Q\) which are then hidden away:

\[ P \triangleright Q = (P[out_1, \ldots, out_n/\,m_1, \ldots, m_n] \mid Q[in_1, \ldots, in_n/\,m_1, \ldots, m_n]) \setminus \{m_1, \ldots, m_n\} \]

for certain \(out_1\) to \(out_n\) and \(in_1\) to \(in_n\) (which are specified).

(a) Consider the following definitions:

\[
\text{Buff} \overset{\text{def}}{=} in(x) \cdot \overline{\text{out}}(x) \cdot \text{Buff} \\
\text{P} \overset{\text{def}}{=} in(x) \cdot \overline{\text{out}}(x) \cdot \overline{\text{out}}(x) \cdot P_2
\]

Define a process \(Q\) such that the \(P \triangleright Q\) (with appropriately chosen actions) is weakly bisimilar to \(\text{Buff}\). Prove this result.
(b) Define an agent $S$ which acts like $Buff$ but which may lose one out of every two consecutive messages.

Modify $P$ so that its outputs are tagged with a 1 and 2. Call this new agent $R$.

Carefully define a new agent $T$ such that $(R \circ S) \circ T$ is weakly bisimilar to $Buff$. You do not need to prove this bisimilarity.

Explain how your solution works and why the tags are necessary.

[25 marks]