CS232F: Programming Languages
II semester 2001-02
Tutorial Sheet on Operational Semantics

1. In the simple functional language with definitions and expressions prove or disprove the following from the operational semantics. Under what conditions would the following equalities hold (if they are not always true)?
   (a) $\textbf{let } d_0; d_1 \textbf{in } e =^e \textbf{let } d_0 \textbf{in } (\textbf{let } d_1 \textbf{in } e)$
   (b) $\textbf{local } d_1 \textbf{in } d_2 = d_1; d_2$
   (c) $\textbf{local } d_1 \textbf{in } (d_2 \parallel d_3) =^d (\textbf{local } d_1 \textbf{in } d_2) \parallel (\textbf{local } d_1 \textbf{in } d_2)$
   (d) $d_1; (d_2 \parallel d_3) =^d (d_1; d_2) \parallel (d_1; d_3)$

2. In the light of the previous question consider the following ML segments and intuitively define conditions under which the two segments are equal and conditions under which the two segments are unequal
   - $\textbf{fun } fx = \textbf{ let } \textbf{ fun } gy = e_0 \textbf{ in } e_1$
   - $\textbf{local } \textbf{ fun } gy = e_0 \textbf{in } \textbf{ fun } fx = e_1$

3. Prove that every program written in the WHILE language has a deterministic behaviour (i.e. every non-final configuration has a unique successor), provided the language of expressions has a deterministic behaviour.

4. Write an iterative program in the WHILE language to compute the $n-th$ fibonacci number $fib(n)$ where $fib(0) = 0$ and $fib(1) = 1$. What is the value of the realtion $\mathcal{F}_i$ for each value of i. Give a closed form expression for it inductively.

5. Assume the language WHILE is extended by the following addition of unnamed functions to the language of expressions.
$$e ::= \ \ldots |\textbf{fun } c \textbf{ result } e \textbf{ nuf}$$
   where c is a command of the language and **result** e denotes the result to be returned.
   (a) How would such functions be used in programming with WHILE?
   (b) How does the transition system for the expression language change because of the inclusion of this new construct?
   (c) The new construct could also contain local declarations of variables. Then how does the semantics of expressions change?
   (d) Rewrite the operational semantics of expressions to include such expressions (with and without local declarations within the command c.

6. **Multiple assignment**. In order to perform exchange operations and more generally, to transform the store more than one location at a time, it is useful to consider the following generalization of the assignment command. $x_1, \ldots, x_n := e_1, \ldots, e_n$
   (a) Define a structural rules for this statement so that when $n = 1$ the command has the same effect as an ordinary assignment command. For $n > 1$, the semantics is such that it is possible to right rotate or left rotate a list of values. That is,
      - $x, y := y, x$ exchanges the values in the locations x and y without necessarily using a temporary variable.
      - $x, y, z := y, z, x$ left rotates the values of variables x, y and z.
      - $x, y, z := z, x, y$ right rotates the values of variables x, y and z.
   (b) What side-conditions would you impose on the rules to ensure there is no confusion in the user's mind?
   (c) What kinds of semantic equivalences can you think of using the multiple assignment command?

(d) Multiple assignment is a useful programming tool while transforming recursion to iteration (see the CS120 notes). However, most imperative programming languages do not support it. How would you translate a multiple assignment command into a sequence of single assignment commands?

(e) Prove the correctness of your translation using your semantic rules. In particular, what side conditions do you need to impose on the syntax and semantics of multiple assignments?

(f) How do your semantic rules stand up to the onslaught of array and pointer component assignments (e.g. $A[i], i := i + 1, A[i] + 1$)?

7. Write CSP programs for the Welfare Crook problem for the following network topologies.

(a) P_0 is the leader and can communicate with both P_1 and P_2, but neither P_1 nor P_2 can communicate with each other.

(b) P_i can send messages only to $P_{i \oplus 1}$ and can receive messages only from $P_{i \ominus 1}$, for $i \in \{0, 1, 2\}$ where $\oplus$ and $\ominus$ denote addition and subtraction modulo 3 respectively.

(c) All the processes can communicate with each other.