Name ______________________ Student No. ______________________

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.

Total Marks: 100

[30] 1. Recall Kruskal's Algorithm for computing a minimum cost spanning tree of a graph G (given below as **C**-like pseudo-code):

Sort edges by costs: $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$
$T \longleftarrow \emptyset$
for $(i = 1;\ i \leq m;\ i = i + 1;)$
  if $(T \cup \{e_i\}$ is acyclic)
    $T \longleftarrow T \cup \{e_i\}$
  end if
end for

Show that $\{G = (V, T)$ is connected$\}$ and $\{(V, T)$ is connected$\}$ are a precondition and postcondition, respectively, of Kruskal's Algorithm. To this end:

(a) Give a suitable definition of "T is promising".

Sol: "T is promising" if $T \cup \{e_{i+1}, \ldots, e_m\}$ is connected, where $\{e_{i+1}, \ldots, e_m\}$ are edges that have not been considered yet (so the algorithm went through the loop i times).

(b) Prove by induction that "T is promising" is a loop invariant.

Sol: Basis Case: At step $i = 0$, T is the empty set, and $\{e_{i+1}, \ldots, e_m\}$ is the entire set of edges, so since G is connected, so $T \cup \{e_1, \ldots, e_m\}$ is connected, so T is promising.

Induction Step: Suppose that T remains promising after stage i, and consider stage $i + 1$. Consider two cases: e_{i+1} accepted and rejected. If e_{i+1} is accepted, then T becomes $T \cup \{e_{i+1}\}$, and $(T \cup \{e_{i+1}\}) \cup \{e_{i+2}, \ldots, e_m\} = T \cup \{e_{i+1}, \ldots, e_m\}$, so $T \cup \{e_{i+1}\}$ clearly remains promising. If e_{i+1} is rejected, then $T \cup \{e_{i+2}, \ldots, e_m\}$ is promising because T already contains a path from one end-point of e_{i+1} to the other.

(c) Show that the postcondition follows from the precondition and the fact that "T is promising" is a loop invariant.

Sol: At the end, since there are no more edges to consider, but T is promising, it follows that T itself is connected (since G was connected).

[30] 2. Consider the following program for division (given as **C** code):

```
int q=0;
int r=x;
while (y <= r) {
   r=r-y;
   q=q+1;
}
```

The above program, on input $x \geq 0, y > 0$, computes $q = \text{div}(x, y)$ and $r = \text{rem}(x, y)$.

(a) Give suitable pre and postcondition for this program (i.e., a pre and postcondition from which correctness of the program follows).

Sol: Precondition $\{x \geq 0 \wedge y > 0\}$ and postcondition $\{x = (q*y) + r \wedge r \geq 0 \wedge r < y\}$.

(b) Give a suitable (see part (c) below) loop invariant and show that it holds by induction.

Sol: A good loop invariant would be $x = (q * y) + r \wedge r \geq 0$. To see that this is indeed a loop invariant we argue by induction on the number of iterations of the while loop. Basis Case: $q = 0, r = x$, so $x = (q * y) + r$ and since $x \geq 0$ and $r = x$, $r \geq 0$. Induction Step: suppose $x = (q * y) + r \wedge r \geq 0$ and we go once more through the loop. Then $y \leq r$, so the assignment `r=r-y` results in $r \geq 0$. Since $(q * y) + r = ((q + 1) * y) + (r - y)$ for $y \leq r$, we have that the new q, r, still satisfy $x = (q * y) + r$.

(c) Show that your postcondition follows from the precondition and the loop invariant.

Sol: Once we have the loop invariant $x = (q * y) + r \wedge r \geq 0$ it is very easy to show that the postcondition holds if the precondition does, since the loop ends when it is no longer true that $y \leq r$, i.e., when it is true that $r < y$.

[20] 3. Prove the following two assertions:

(a) $\exists x(\alpha \wedge \beta) \models (\exists x\alpha \wedge \exists x\beta)$

Sol: Suppose that $\mathcal{M}$ is a structure, and σ is an object assignment, and that $\mathcal{M} \models \exists x(\alpha \wedge \beta)[\sigma]$. By the Basic Semantic Definitions (BSDs), $\mathcal{M} \models (\alpha \wedge \beta)[\sigma(m/x)]$ for some m in the domain of discourse of $\mathcal{M}$. Again, by BSDs, $\mathcal{M} \models \alpha[\sigma(m/x)]$ and $\mathcal{M} \models \beta[\sigma(m/x)]$. Thus, $\mathcal{M} \models \exists x\alpha[\sigma]$ and $\mathcal{M} \models \exists x\beta[\sigma]$, so $\mathcal{M} \models (\exists x\alpha \wedge \exists x\beta)[\sigma]$. This shows that the RHS is a logical consequence of the LHS, as required.

(b) $(\exists x\alpha \wedge \exists x\beta) \not\models \exists x(\alpha \wedge \beta)$

Sol: We exhibit a structure $\mathcal{M}$ and particular α, β such that $\mathcal{M} \models (\exists x\alpha \wedge \exists x\beta)$, but $\mathcal{M} \not\models \exists x(\alpha \wedge \beta)$. We ignore $[\sigma]$ as the LHS and RHS formulas will be a sentences. Let $\mathcal{M}$ be the standard model $\underline{\mathbb{N}}$, and let α, β be $x = x + x, x = x * x \wedge x \neq 0$, respectively. Note that $\underline{\mathbb{N}} \models (\exists x(x = x + x) \wedge \exists x(x = x * x \wedge x \neq 0))$ (just let the first x be 0 and the second x be 1). But $\underline{\mathbb{N}} \not\models \exists x(x = x + x \wedge (x = x * x \wedge x \neq 0))$, as there is no natural number $n \neq 0$ for which both $n = n + n$ and $n = n * n$ hold simultaneously. Note that this is just one example of a structure; there are others that work just as well.

[20] 4. Given any first order language $\mathcal{L}$, show that the set of $\mathcal{L}$-formulas can be generated with a grammar of type 2.

You can assume that $\mathcal{L} = [f_1, \ldots, f_n; P_1, \ldots, P_m]$ where the f_i's are function symbols of k_i-arity, and the P_j's are predicate symbols of l_j-arity. Now give a grammar G (of type 2) such that $\alpha \in L(G)$ iff α is an $\mathcal{L}$-formula.

Explain in particular how to resolve the variable problem: a first order language has infinitely many variables at its disposal, but a grammar has only a finite set of terminals.

Sol: The following rules would do the job (T^i means T repeated i many times). Since a grammar can only have a finite set of terminals, we generate variables as follows: $T \longrightarrow xI, I \longrightarrow 1B, B \longrightarrow \varepsilon|1B|0B$ thus all variables are of the form x followed by a binary index. Terms are now built like this: $T \longrightarrow f_1T^{k_1}|f_2T^{k_2}|\cdots|f_nT^{k_n}$. Atomic formulas are built like this: $A \longrightarrow P_1T^{l_1}|P_2T^{l_2}|\cdots|P_mT^{l_m}$. Finally, general formulas are built like this $F \longrightarrow A|\neg F|(F \wedge F)|(F \vee F)|\exists xIF|\forall xIF$.