Consider the grammar G for the language L(G):

$S \Rightarrow A\ B\ C\ \$$
$A \Rightarrow B$
$A \Rightarrow b\ B$
$B \Rightarrow b\ B$
$B \Rightarrow b$
$C \Rightarrow c$

1a. (14 points) Give the CFSM for G. Note that if G is not LR(0), then the CFSM will have what are termed *inadequate* states in *Crafting a Compiler with C*. That is, the CFSM will contain states that are incompatible with constructing an LR(0) parser from the CFSM.

1b. (6 points) Is the grammar LR(0)? Explain in terms of the CFSM constructed in **1a**.

1c. (6 points) Is the grammar LL(1)? Why or why not?

1d. (6 points) What is the shortest member of the language recognized by G?

1e. (6 points) Let *s* be a string of terminals that is a member of the language L(G) , where L(G) is the language recognized by G. Let *S* be the set of all such *s*. Does *S* have a finite or an infinite number of members? Stated less formally, does the grammar recognize a finite, or infinite number of strings of terminals?

1f. (4 points) Consider the two derivations;

1. $S \Rightarrow A\ B\ C\ \$ \Rightarrow b\ B\ B\ C\ \$ \Rightarrow b\ b\ B\ C\ \$ \Rightarrow b\ b\ b\ C\ \$ \Rightarrow b\ b\ b\ c\ \$$
2. $S \Rightarrow A\ B\ C\ \$ \Rightarrow B\ B\ C\ \$ \Rightarrow b\ B\ B\ C\ \$ \Rightarrow b\ b\ B\ C\ \$ \Rightarrow b\ b\ b\ C\ \$ \Rightarrow b\ b\ b\ c\ \$$

What property of the grammar do these derivations illustrate?

Given the loop nest:

```
for (i = 0; i < 1000; i++) {
        for (j = 0; j < 1000; j++) {
                S1    ...    = a[i,j-1];
                S2    a[i,j] = ... ;
        }
}
```

2a. (7 points) What dependence(s) (input, output, anti, true (or flow)) exists between statements **S1** and **S2**? Assume a perfect dependence analysis

2b. (7 points) What is the direction vector(s) for the dependence(s)?

2c. (7 points) What is the distance vector(s) for the dependence?

2d. (8 points) Can the ***i*** loop be parallelized in the code above, assuming no other transformations are performed?

2e. (8 points) Can the ***j*** loop be parallelized, assuming no other transformations are performed?

2f. (5 points) Parallelize as many loops as possible in the code above. Indicate a parallel loop as a *parfor* loop. You can perform transformations other than simply marking loops as *parfor* loops, as needed.

3a. (8 points) The CGG compiler will transform the code:

```
if (x = y) {
        ...
        a = u+v;
} else {
        ...
        a = u+v;
}
```

into

```
a = u+v;
if (x = y) {
        ...
} else {
        ...
}
```

The generated code only uses general purpose registers. On machine X this leads to additional register spills and loads, and on machine Y it does not. What can you infer about the relative number of registers on X and Y? (One sentence is sufficient to answer this question.)

3c. (8 points) Given the code:

ld a r_1
ld b r_2
ld c r_3
$r_3 = r_1 + r_2$
$r_5 = r_2 + r_3$
$r_6 = r_3 + r_1$

What is the minimum number of registers needed so that there is no need for register spills?