Exam 3

CS 2800 Section 1, Spring 2012

Name: ______________________

Student Id (last 4 digits): ______________________

- You must take the exam in the section you are registered for.
- Write down the answers in the space provided.
- Write clearly. If we can't read what you write, we can't give you credit for it.
- You may use anything we covered in class or in the class notes. Everything else needs to be defined. If you have any questions, ask!

Question	Points	out of
1		500
2		700
Total		1200

Good luck!

Question 1. (500 = 100 + 200 + 200 points)

Consider the following incomplete list of rewrite rules, listed in the order they were added to the *Logical World*. Unfortunately, the author of the rules wrote illegibly, so that rule (iii) could not be deciphered.

(i) `(h a b) = (h b a)`

(ii) `(h x (f y z)) = (h (f x y) z)`

(iii) ???

(a) Consider the expression

$$e = \texttt{(h r (f (h s (f t u)) v))}\,.$$

Ignoring rule (iii), which is the first rewrite rule that applies to e? Also, show the result of that application.

(b) Show that rules (i) and (ii) alone may lead to an infinite rewrite loop by specifying a suitable expression e and show how, using only rules (i) and (ii), rewriting e some number of steps results in e again. In every rewrite step, indicate the rule that you applied.

Hint: Remember that *permutative rules* (rules that do nothing but permute their variable arguments) alone do not cause an infinite loop, because they are treated in a special way.

(c) Define a rewrite rule (iii) such that the complete set of rules **cannot** lead to an infinite rewrite loop. In particular, your choice for (iii) must prevent the scenario you exhibited in (b). Explain why it does.

Question 2. (700 = 400 + 300 points)

Consider the following functions, whose purpose it is to double the natural numbers in a list, while simply reproducing other elements:

```
(defunc double (l)
  :input-contract (listp l)
  :output-contract (listp (double l))
  (cond ((endp l)         ())
        ((natp (first l)) (cons (* (first l) 2) (double (rest l))))
        (t                (cons    (first l)    (double (rest l))))))


(defunc double-t (l acc)
  :input-contract (and (listp l) (listp acc))
  :output-contract (listp (double-t l acc))
  (cond ((endp l)         acc)
        ((natp (first l)) (double-t (rest l) (cons (* (first l) 2) acc)))
        (t                (double-t (rest l) (cons    (first l)    acc)))))


(defunc double* (l)
  :input-contract (listp l)
  :output-contract (listp (double* l))
  (rev (double-t l ())))
```

For example, we have:

```
(check= (double '(1 () abc (2 3) 10)) '(2 NIL ABC (2 3) 20))
```

(a) Here is the key lemma relating `double-t` with `double`.

```
(listp l) /\ (listp acc) => (double-t l acc) = (app (rev (double l)) acc)
```

Using the induction scheme `double-t` gives rise to, prove **the first recursive case** of this lemma, *i.e.*, the case

```
(listp l) /\ (listp acc) /\ (not (endp l)) /\ (natp (first l)) /\ ... .
```

You do not need to prove any lemmas or theorems from the lectures/homework/notes that you may be using, but clearly identify them.

Space for your proof:

(b) Assuming the lemma in (a) is a theorem, prove:

```
(listp l) => (double* l) = (double l)
```

Use pure equational reasoning. You do not need to prove any lemmas or theorems from the lectures/homework/notes that you may be using, but clearly identify them.