Mini-Quiz 6

1. Write your name:

2. (Rosen, Sec. 2.6, Ex. 29) Let

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Find the following:

(a) $A \vee B$

(b) $A \wedge B$

(c) $A \odot B$

Tutorial 6 Problems

Problem 1 Is a Harvard degree really worth more than an MIT degree?! Let us say that a person with a Harvard degree starts with $40,000 and gets a $20,000 raise every year after graduation, whereas a person with an MIT degree starts with $30,000, but gets a 20% raise every year. Assume inflation is a fixed 8% every year. That is, $1.08 a year from now is worth $1.00 today. (You'll need a calculator to get final answers; if one is not available, it's ok to express the answer as a closed form numerical expression.)

(a) How much will a person with a Harvard degree be making in the n^{th} year?

(b) How much will a person with an MIT degree be making in the n^{th} year?

(c) How much is a Harvard degree worth today if the holder will work for n years following graduation? *Hint*: In Lecture 8 a closed formula for $\sum_{i=0}^{n} ir^i$ was derived.

(d) How much is an MIT degree worth in this case?

(e) If you plan to retire after twenty years, which degree would be worth more?

Problem 2 There are "sum" and "difference" operators that play the same role in discrete math as integral and derivative operators place in calculus. In particular, the difference operator Δ is defined by

$$\Delta f(x) = f(x+1) - f(x)$$

and the summation operator

$$\sum_a^b g(x)\delta x = \sum_{x=a}^{b-1} g(x)$$

This sum operator is a bit annoying, as many sloppy mathematicians will use the left hand side to denote a sum up to b, rather than $b-1$. But let's work with it for today.

(a) Suppose that $g(x) = \Delta f(x)$. Prove that

$$\sum_a^b g(x)\delta x = f(b) - f(a)$$

(b) Suppose that $h(n) = \sum_{k=0}^{n} f(k)$. What is $\Delta h(n)$?

(c) Prove that

$$\Delta(u(x)v(x)) = u(x)\Delta v(x) + v(x+1)\Delta u(x)$$

(d) The *falling power* function is defined by the equation

$$x^{\underline{m}} = x(x-1)\cdots(x-m+2)(x-m+1).$$

In other words, just like x^m, it is a product of m terms, but they fall away from x. Prove

$$\Delta x^{\underline{m}} = m x^{\underline{m-1}}$$

(e) Find the value of

$$\sum_{x=a}^{b} x^{\underline{m}}$$

Problem 3 Consider the Strong Caching system given in lecture as a labeled state machine:

Caching system (strong)

The system is composed of a main memory store that is accessed by two 'clients'. Each client accesses the memory through a cache. When the memory is updated, the caches are cleaned out.

State machine:

- Q: For every $x \in addresses$:
  - $memory(x)$, an element of $data$
  - $cache_1(x)$, an element of $data \cup \{null\}$
  - $cache_2(x)$, an element of $data \cup \{null\}$

  And a number of doubled structures (one for each client/cache):
  - req_1, an element of $\{\text{“}read\text{”}, \text{“}write\text{”}\} \cup \{null\}$
  - $reqaddr_1$, an element of $addresses \cup \{null\}$ (used for both reads and writes)
  - $reqval_1$, an element of $data \cup \{null\}$ (used for writes only)
  - ret_1, an element of $\{\text{“}OK\text{”}, null\} \cup data$
  - req_2, an element of $\{\text{“}read\text{”}, \text{“}write\text{”}\} \cup \{null\}$
  - $reqaddr_2$, an element of $addresses \cup \{null\}$ (used for both reads and writes)
  - $reqval_2$, an element of $data \cup \{null\}$ (used for writes only)
  - ret_2, an element of $\{\text{“}OK\text{”}, null\} \cup data$
- Q_0: *memory* arbitrary, caches all *null*, all other components *null*.
- L: All are for $i \in \{1, 2\}$ (each client). Note that, for each label, the italicized text is just a description of the intended purpose.

  Input labels:

  req-read$_i$ (x), $x \in addresses$
  an input from client i requesting to read the value in address x from $cache_i$.

  req-write$_i$ (x, v), $x \in addresses$, $v \in data$
  a request by client i to write value v in location x in memory (not cache).

  Internal labels:

  comp-read$_i$ (x), $x \in addresses$
  actually performed the read that was requested and stored the result.

  comp-write$_i$ (x, v), $x \in addresses$, $v \in data$
  actually performed the write that was requested.

  copy$_i$ (x), $x \in addresses$
  copied the value at x from memory into $cache_i$.

  drop$_i$ (x), $x \in addresses$
  dropped the value at x from $cache_i$.

Output labels:

ret-read$_i$ (v), $v \in data$
signal result of a read request and computation for the value at x.
ret-write$_i$ $(\text{“}OK\text{”})$
signal completion of a write request and computation.

- δ:

req-read$_i$ (x)
Can occur anytime
if $req_i = null$ then
  $req_i := \text{“}read\text{”}$
  $reqaddr_i := x$

req-write$_i$ (x, v)
Can occur anytime
if $req_i = null$ then
  $req_i := \text{“}write\text{”}$
  $reqaddr_i := x$
  $reqval_i := v$

ret-read$_i$ (v)
Can occur if $retval_i \in values$
  $req_i := null$
  $reqaddr_i := null$
  $reqval_i := null$
  $retval_i := null$

ret-write$_i$ $(\text{“}OK\text{”})$
Can occur if $retval_i = \text{“}OK\text{”}$
  $req_i := null$
  $reqaddr_i := null$
  $reqval_i := null$
  $retval_i := null$

comp-read$_i$ (x)
Can occur if $req_i = \text{“}read\text{”}$, $reqaddr_i = x$, $cache_i(x) \neq null$, and $retval_i = null$
  $retval_i := cache_i(x)$

comp-write$_i$ (x, v)
Can occur if $req_i = \text{“}write\text{”}$, $reqaddr_i = x$, $reqval_i(x) = v$, and $retval_i = null$
  $memory(x) := v$
  $retval_i := \text{“}OK\text{”}$
  $cache_1(x) := null$
  $cache_2(x) := null$

copy$_i$ (x)
Can occur anytime
  $cache_i(x) := memory(x)$

drop$_i$ (x)
Can occur anytime
  $cache_i(x) := null$

Also consider a simple Centralized Memory system as a labeled state machine:

Centralized memory system

The system is composed of a main memory store that is accessed by two 'clients'. Each client accesses the memory directly.

State machine:

- Q: For every $x \in addresses$:
  - $memory(x)$, an element of $data$

  And a number of doubled structures (one for each client):
  - req_1, an element of $\{\text{“}read\text{”}, \text{“}write\text{”}\} \cup \{null\}$
  - $reqaddr_1$, an element of $addresses \cup \{null\}$ (used for both reads and writes)
  - $reqval_1$, an element of $data \cup \{null\}$ (used for writes only)
  - ret_1, an element of $\{\text{“}OK\text{”}, null\} \cup data$
  - req_2, an element of $\{\text{“}read\text{”}, \text{“}write\text{”}\} \cup \{null\}$
  - $reqaddr_2$, an element of $addresses \cup \{null\}$ (used for both reads and writes)
  - $reqval_2$, an element of $data \cup \{null\}$ (used for writes only)
  - ret_2, an element of $\{\text{“}OK\text{”}, null\} \cup data$
- Q_0: *memory* arbitrary, all other components *null*.
- L: All are for $i \in \{1, 2\}$ (each client). Note that, for each label, the italicized text is just a description of the intended purpose.

  Input labels:

  req-read$_i$ (x), $x \in addresses$
  an input from client i requesting to read the value in address x from memory.

  req-write$_i$ (x, v), $x \in addresses$, $v \in data$
  a request by client i to write value v in location x in memory.

  Internal labels:

  comp-read$_i$ (x), $x \in addresses$
  actually performed the read that was requested and stored the result.

  comp-write$_i$ (x, v), $x \in addresses$, $v \in data$
  actually performed the write that was requested.

  Output labels:

  ret-read$_i$ (v), $v \in data$
  signal result of a read request and computation for the value at x.

  ret-write$_i$(“OK”)
  signal completion of a write request and computation.

- δ:

  req-read$_i$ (x)
  Can occur anytime
  if $req_i = null$ then
  $req_i :=$ "$read$"
  $reqaddr_i := x$

  req-write$_i$ (x, v)
  Can occur anytime
  if $req_i = null$ then
  $req_i :=$ "$write$"
  $reqaddr_i := x$
  $reqval_i := v$

  ret-read$_i$ (v)
  Can occur if $retval_i \in values$
  $req_i :=null$
  $reqaddr_i := null$
  $reqval_i :=null$
  $retval_i := null$

  ret-write$_i$("OK")
  Can occur if $retval_i =$ "OK"
  $req_i :=null$
  $reqaddr_i := null$
  $reqval_i :=null$
  $retval_i := null$

  comp-read$_i$ (x)
  Can occur if $req_i =$ "$read$", $reqaddr_i = x$, and $retval_i = null$
  $retval_i := memory(x)$

  comp-write$_i$ (x, v)
  Can occur if $req_i =$ "$write$", $reqaddr_i = x$, $reqval_i(x) = v$, and $retval_i = null$
  $memory(x) := v$
  $retval_i :=$ "OK"

(a) Try to prove, as carefully as you can, that the strong caching system labeled state machine in fact implements the centralized memory labeled state machine.

This proof can be done using ad hoc methods, starting from any execution of the caching system and constructing the (possibly infinite) execution of the centralized memory machine. The construction can proceed by structural induction on the definition of the execution of the caching system.

Prove by induction that the trace of the n^{th} finite prefix of the execution of the cache system is the same as the trace of the corresponding finite prefix of the centralized memory machine.