Exam 2 Solutions

Revisiting Exam 1

1. Prove by induction that every finite non-empty subset of the real numbers contains a *least* element, where an element $x \in S$ is defined as the least element if $\forall z \in S - \{x\}.\, x < z$. (Note: you should not just assume all finite sets are well ordered for this question.)

(This is nearly identify to Problem 5 on PS5, and Problem 8 on Exam 1. Since a significant number of people still had problems with it on this exam, you shouldn't be surprised to seee a similar question again on the final!)

Our induction predicate is:

$$P(n) ::= \text{all sets of real numbers of size } n \text{ have a } \textit{least} \text{ element.}$$

We what to show this holds for all non-empty sets, $n \in \mathbb{N}^+$.

Base case: $n = 1$. Consider a set of real numbers with size one: $\{x\}$. It has only a single element, x, so that element will be the minimum.

Inductive step: $\forall m \in \mathbb{N}^+.\, P(m) \implies P(m+1)$. Assume all sets of real numbers of size m have a least element. Every set of size $m+1$, S', is the result of adding a new element, q, to a set of size m: $S' = S \cup \{q\}$, $q \notin S$. By the inductive hypothesis, we know S (size m) has a least element, w. There are two possibilities for the new element: (1) $q < m$. Then, minimum$(S') = q$. or (2) $q > m$. Then, minimum$(S') = m$. In both cases, the minimum of S' is well defined.

By induction, we have proven $P(n)$ holds for all $n \in \mathbb{N}^+$.

State Machines and Invariants

2. For each of the statements below, indicate if the implication stated is valid (must always be true) or invalid (may be false). Provide a short justification supporting your answer.

a. If $M = (S, G : S \times S, q_0 \in S)$ is a state machine where G is a total surjective function, all states in S are reachable.

Invalid. Here is a counter-example:

$$\begin{aligned} S &= \{A, B\} \\ G &= \{(A \to A, B \to B\} \\ q_0 &= A \end{aligned}$$

G is a total, surjective function, but the only reachable state is A.

b. If $P(q)$ is a *preserved invariant* for machine $M = (S, G, q_0)$, and r is a reachable state in M, $P(r)$ is true.

Invalid. The definition of a *preserved invariant* means that $P(q) \implies P(r)$ if $(q \to r) \in G$. If $\overline{P(q_0)}$, then $P(q)$ is not true for all reachable states.

c. If $M = (S, G, q_0)$ is a state machine and R is the set of reachable states in M,

$$S = R \implies (G \cup \{(q_0 \to q_0)\}) \text{ is an injective function.}$$

Invalid. The implication claims that if all states are reachable, then $G \cup \{(q_0 \to q_0)$ is an injective (≤ 1 in) function (≤ 1 out). So, to show the implication is false, all we need is a counterexample to at least one of the properties. Here's one:

$$\begin{aligned} S =& \{A, B\} \\ G =& \{A \to B\} \\ q_0 =& A \end{aligned}$$

But, $G \cup \{q_0 \to q_0\}$ is not a function, since it has two outputs for A, so we have a counter-example.

Stable Matching

The state machine model of the Gale-Shapley algorithm from Class 15 and Problem Set 7 is shown (without modification) below.

$$\begin{aligned}
S = \{&(\textit{pairings}, \textit{proposals}) \mid \\
&\quad \textit{pairings} = \{(a,b) \mid a \in A, b \in B, \text{no duplicate occurences of } a \text{ or } b \text{ in } \textit{pairings}\}, \\
&\quad \textit{proposals} = (r_1, r_2, \cdots, r_n), 0 \leq r_i \leq n\} \\
G = \{&(\textit{pairings}, \textit{proposals} = (r_1, r_2, \cdots, r_n)) \to (\textit{pairings}', \textit{proposals}') \mid \\
&\quad i \in \{1, \cdots, n\}, r_i < n \\
&\quad \forall b \in B. (a_i, b) \notin \textit{pairings} \\
&\quad b_x ::= \text{the } r_i\text{-ranked choice for } a_i \\
&\quad \textit{proposals}' = (r_1, \cdots, r_i + 1, \cdots, r_n) \\
&\quad \textit{pairings}' = \begin{cases} \textit{pairings} \cup \{(a_i, b_x)\} & \forall a_z \in A. (a_z, b_x) \notin \textit{pairings} \text{ (Case 1)} \\ \textit{pairings} \cup \{(a_i, b_x)\} - \{(a_z, b_x)\} & \exists a_z \in A. (a_z, b_x) \in \textit{pairings} \wedge a_z \prec_{b_x} a_i \text{ (Case 2)} \\ \textit{pairings} & \text{otherwise (Case 3)} \end{cases} \\
&\quad \} \\
q_0 = (&\textit{pairings} = \{\}, \textit{proposals} = (0, 0, \ldots, 0))
\end{aligned}$$

3. For each of the following predicates, answer whether or not it is a *preserved invariant* for $M = (S, G, q_0)$ as defined above, and provide a brief justification supporting your answer. We use a, a_1, a_2 to denote elements of A, and b, b_1, b_2 to denote elements of B.

a. $P(q = (\textit{pairings}, \textit{proposals})) := \textit{pairings}$ contains a pair including a.

Not Preserved. The Case 2 update for $\textit{pairings}' = \textit{pairings} \cup \{(a_i, b_x)\} - \{(a_z, b_x)\}$ removes a pair (a_z, b_x) from *pairings*, but no pair including a_z is added.

b. $P(q = (\textit{pairings}, \textit{proposals})) := \textit{pairings}$ contains a pair including b.

Preserved. If an element of b is in a pair, it never loses that match without switching to a more preferred match. For the three update cases, case 1 adds a new pair without removing any and case 3 does not change the pairings. In case 2, $\textit{pairings}' = \textit{pairings} \cup \{(a_i, b_x)\} - \{(a_z, b_x)\}$ removes (a_z, b_x) but adds (a_i, b_x), so b_x is still in a pair and the invariant is preserved.

c. $P(q = (\textit{pairings}, \textit{proposals} = (r_1, r_2, \cdots, r_n))) := (r_1 + r_2 + \cdots + r_n) > 7$.

Preserved. The vaues in *proposals* are updated by $\textit{proposals}' = (r_1, \cdots, r_i + 1, \cdots, r_n)$. So, every value except one is the same as in the previous state, and exactly one value is increased by 1. Hence, if the sum $(r_1 + r_2 + \cdots + r_n) > 7$ in one state, it is also > 7 in the next state, and the invariant is preserved.

d. $P(q = (\textit{pairings}, \textit{proposals} = (r_1, r_2, \cdots, r_n))) := |\textit{pairings}| \leq (r_1 + r_2 + \cdots + r_n)$.

Preserved. The size of *pairings* increases by at most one each step; the sum of the values in *proposals* always increases by one. So, the $\leq$ property is preserved.

Recursive Data Types

Consider the recursive data type, *cist*, defined by:

– **Base: null** is a *cist*.
– **Constructor:** for all *cist* object, p, expand(p) is a *cist*.

The *size* of a *cist*, p, is the number of times it takes to expand starting from **null** to produce that *cist*. So, for example, size(expand(expand(expand(**null**)))) = 3.

4. Provide a precise and complete definition of *size* for *cist* all objects.

$$\text{size}(p) = \begin{cases} 0 & p = \textbf{null} \\ 1 + \text{size}(q) & p = \text{expand}(q) \end{cases}$$

Define the merge of two cists as:

$$\text{merge}(p_1, p_2) = \begin{cases} p_2 & p_1 = \textbf{null} \\ \text{expand}(\text{merge}(q, p_2)) & p_1 = \text{expand(q)} \end{cases}$$

5. Prove that for any two cists, p_1 and p_2,

$$\text{size}(\text{merge}(p_1, p_2)) = \text{size}(p_1) + \text{size}(p_2).$$

We prove by structural induction on the first input, p_1, following the definition of merge.

Base case: $p_1 = \textbf{null}$. By the definition, $\text{merge}(p_1 = \textbf{null}, p_2) = p_2$. Hence, $\text{size}(\text{merge}(p_1, p_2)) = \text{size}(p_2) = \text{size}(p_1) + \text{size}(p_2)$ since $\text{size}(p_1 = \textbf{null}) = 0$.

Constructor case: $p_1 = \text{expand}(q)$. By the structural induction hypothesis, we know $\text{size}(\text{merge}(q, p_2)) = \text{size}(q) + \text{size}(p_2)$. We'll use z to represent $\text{size}(\text{merge}(q, p_2)$. To prove the property holds, we show that $\text{size}(\text{merge}(p_1, p_2)) = z + 1 = \text{size}(p_1) + \text{size}(p_2)$. By the definition of size, $\text{size}(p_1 = \text{expand}(q)) = 1 + \text{size}(q)$. So, $\text{size}(\text{merge}(p_1, p_2)) = z + 1$. Also, since $\text{size}(q) + \text{size}(p_2) = z$ and $\text{size}(p_1) = \text{size}(q) + 1$, we know $\text{size}(p_1) + \text{size}(p_2) = z + 1$.

By showing the property holds for the base object, and is preserved by all constructors, we have shown the property holds for all cists by structural induction. Note that we did not make any constraints on p_2 in our proof, so it was only necessary to do the induction on p_1.

6. Complete the definition of the state machine below that models `max_element`, that returns the greatest element of the input list `p`. You may assume `p` is a non-empty list of natural numbers.

```
def max_element(p):
    x = p[0]
    i = 1
    while i < len(p):
        if p[i] > x:
            x = p[i]
        i = i + 1
    return x
```

$$\begin{aligned} S &= \{(x, i) \mid x, i \in \mathbb{N}\} \\ G &= \{(x, i) \to (x', i') \mid \\ &\qquad i < \texttt{len(p)} \wedge \\ &\qquad i' = i + 1 \wedge \\ &\qquad x' = \text{p[i] if p[i]} > x \\ &\qquad x' = x \text{ otherwise} \\ q_0 &= (\text{p[0]}, 1) \end{aligned}$$

7. Prove that the state machine (from problem 6) always terminates.

There are no edges out from any state where $i \geq$ `len(p)` and the value of `len(p)` is finite. Each step increases the value of i by 1 because G includes $i' = i + 1$ in the update state rules. In q_0, $i = 1$. Hence, the value of i in the state must eventually reach `len(p)`, which must be a terminating state.

8. Prove that `max_element`, as modeled by the state machine from Problem 6, always returns the maximum value of that list.

We proved termination in Problem 7. So, it remains to show partial correctness. We prove this using the Invariant Principle.

The preserved invariant is:

$$P(q = (x, i)) ::= x \text{ is the maximum value of p[0 ... (i-1)]}$$

To use the invariant principle, we need to show that (1) the invariant is preserved, (2) it holds in q_0, and (3) in the terminating state, the preserve invariant implies the desired property.

(1) The invariant is preserved: $P(q) \implies P(r)$ if $(q \to r) \in G$. By the invariant hypothesis, we know $P(q = (x, i))$ so x is the maximum value of the first i elements of p. The transition rule is $i' = i + 1$ and $x' =$ p[i] if p[i] $> x$ or $x' = x$ otherwise. So, there are two cases to consider: (case 1) p[i] $> x$ means that the element p[i] is greater than any element in p[0 ... (i-1)]. The next state will be ($x' =$ p[i], $i' = i + 1$) so it satisfies the invariant since x is the maximum value of p[0. . . i]. (case 2) p[i] $\leq x$ means that x is still the maximum value in p[0. . . i]. Hence, the next state ($x' = x, i' = i + 1$) satisfies the invariant. Thus, $P(q)$ is a preserved invariant.

(2) $P(q_0)$. $q_0 = (x =$ p[0]$, i = 1)$. The invariant holds since p[0] is the only element in p[0..(i-1)], so it must be the maximum element.

(3) $P(\text{terminating}) \implies$ max_element returns the maximum value of p. In the terminating state, $i =$ len[p]. By the preserved invariant, $x$is the maximum value of p[0 ... (i - 1)]. So, then $i =$ len[p], x is the maximum value of all elements of p.