CSC236, Fall 2012
Assignment 3
sample solution

1. Let $L = \{x \in \{0,1\}^* \mid \text{fourth-last symbol in } x \text{ is } 0\}$. Prove that any DFSA that accepts L has at least 16 states. **Hint**: Consider the sixteen binary strings of length four, and what happens if two of them drive a DFSA to the same state.

   Proof (by contradiction): Assume there is a DFSA M, with start state s and fewer than 16 states, that accepts L.

   Then each of the sixteen binary strings of length four drives M to its respective state, and by the pigeonhole principle, there will be at least one pair of strings that drive M to the same state, let's call it state q. Denote this pair of strings by their bits as $b_0b_1b_2b_3$ and $b'_0b'_1b'_2b'_3$, and note that (being different strings) there is some $0 \leq i \leq 3$ with $b_i \neq b'_i$. Without loss of generality, assume that $b_i = 0$ and $b'_i = 1$.

   Now consider the new pair of strings produced by appending a suffix x consisting of i 0s to each string. By construction $b_0b_1b_2b_3x$ has a zero in the fourth-last place, and should be accepted by M, whereas $b'_0b'_1b'_2b'_3x$ has a 1 in the fourth-last place and should be rejected. However, by our assumption, both strings drive M to state q, that is $\delta^*(s, b'_0b'_1b'_2b'_3) = \delta^*(s, b_0b_1b_2b_3) = q$, so

   $$\delta^*(s, b_0b_1b_2b_3x) = \delta^*(q, x) = \delta^*(s, b'_0b'_1b'_2b'_3x) = q'$$

   However, q' must be either an accepting or non-accepting state, so either $b_0b_1b_2b_3x$ or $b'_0b'_1b'_2b'_3x$ have driven M to the wrong state. Contradiction

   Since I assumed there was a DFSA M that accepted L with fewer than 16 states, and then derived a contradiction, the assumption is false, and any DFSA that accepts L must have at least 16 states.

2. Prove that the following terminates, given the precondition $x \in \mathbb{N}$:

```
y = x * x
while not y == 0 :
  x = x - 1
  y = y - 2 * x - 1
```

   Hint: Trace through the code for a few small values of x, then derive (and prove) a loop invariant that helps prove termination.

Solution: Experimenting with $x \in \{0,1,2,3\}$ I see that at the end of each loop iteration $y = x^2$, and that x is steadily decreasing. I need to exhibit a decreasing sequence in $\mathbb{N}$, and $\langle x_i \rangle$ seems appropriate, provided I can show that $x_i \geq 0$. This suggests the following loop invariant:

$$y_i = x_i^2 \text{ and } x_i \in \mathbb{N}$$

Of course, the claim only makes sense if the loop iterates i times, so I have:

$P(i)$: If the precondition is satisfied, and there are at least i iterations of the loop, then $y_i = x_i^2$ and $x_i \in \mathbb{N}$.

Claim: $\forall i \in \mathbb{N}, P(i)$. I prove this by mathematical induction on i

Base case, $i = 0$: If there have been at least 0 iterations of the loop, then by the precondition $x_0 \in \mathbb{N}$, and $y_0 = x_0^2$ by the assignment statement.

Induction step: Assume $i \in \mathbb{N}$, that $P(i)$, and there is an $(i+1)$th iteration of the loop.

Then, by the IH, $y_i = x_i^2$, and (since the loop condition was satisfied in order to begin iteration $i+1$) $x_i^2 \neq 0$, so $x_i \neq 0$.

By the IH, $x_i \in \mathbb{N}$, so (since it's not 0) $x_i \geq 1$.

By the first assignment statement within the loop, $x_{i+1} = x_i - 1 \geq 1 - 1$, so x_{i+1} is an integer no smaller than 0, in other words a natural number.

By the second assignment statement, since $x_{i+1} = x_i - 1$, and by the IH $y_i = x_i^2$:

$$y_{i+1} \quad = y_i - 2x_{i+1} - 1 = x_i^2 - 2(x_i - 1) - 1 = x_{i+1}^2$$

Thus $P(i+1)$ holds.

Since for a generic $i \in \mathbb{N}$ $P(i)$ implies $P(i+1)$, then $\forall i \in \mathbb{N}, P(i) \Rightarrow P(i+1)$.

I conclude $\forall i \in \mathbb{N}, P(i)$, by mathematical induction.

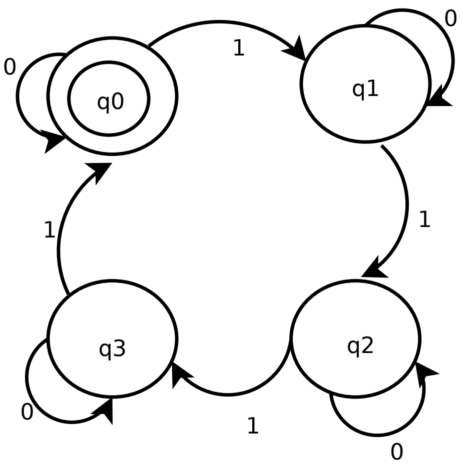
Claim: $\langle x_i \rangle$, the sequence of values of x after the loop iterates i times, is a decreasing sequence in $\mathbb{N}$.

Proof: The loop invariant establishes that $\langle x_i \rangle$ is a sequence in $\mathbb{N}$, it remains to prove that it is decreasing. If there is an $(i+1)$th iteration of the loop, then $x_{i+1} = x_i - 1$, so $x_i > x_{i+1}$. Thus $\langle x_i \rangle$ is a decreasing sequence in $\mathbb{N}$.

By the Well-Ordering Principle, any decreasing sequence in $\mathbb{N}$ is finite, so there are finitely many elements in $\langle x_i \rangle$, hence finite iterations of the loop. In other words, the loop terminates.

3. Design a DFSA that accepts the language of binary strings over $\{0,1\}$ that have a multiple of 4 1s. Devise, and prove a state invariant, and explain how it shows that your DFSA accepts this language.

Solution: I'll need at least four states, one for the remainder of the number of 1s after division by four. My accepting state will be where my machine is driven by strings where the number of ones has remainder 0 after division by four. I'll label each state by the appropriate remainder, and have state 0 be the start state (since ε has 0 ones, a multiple of 4).



I need to prove the following state invariant to convince you that this machine accepts the language L. My predicate is $P(s)$:

$$P(s) : \delta^*(q_0, s) = \begin{cases} q_0 & \text{if the number of 1s in } s \text{ has remainder 0 when divided by 4} \\ q_1 & \text{if the number of 1s in } s \text{ has remainder 1 when divided by 4} \\ q_2 & \text{if the number of 1s in } s \text{ has remainder 2 when divided by 4} \\ q_3 & \text{if the number of 1s in } s \text{ has remainder 3 when divided by 4} \end{cases}$$

I prove that $\forall s \in \{0,1\}^*, P(s)$.

Proof (structural induction on s): s is either ε or $s = ya$, where $y \in \{0,1\}^*$ and $a \in \{0,1\}$. I consider each case.

Case $s = \varepsilon$: In this case $\delta^*(q_0, s) = q_0$ by definition of extended transition function δ^*, and $P(\varepsilon)$ holds. Note that the claims about $|\varepsilon|$ having remainders 1–3 are vacuously true.

Case $s = ya$ where $y \in \{0,1\}^*$ and $a \in \{0,1\}$: I assume $P(y)$, and then use this to prove that $P(s)$ follows. There are two subcases, according to whether $a = 0$ or $a = 1$.

Case $a = 0$: In this case the number of 1s in y is unchanged by appending 0, so

$$\begin{aligned}
\delta^*(q_0, s) &= \delta^*(q_0, y0) = \delta(\delta^*(q_0, y), 0) \\
\text{by } P(y) &= \begin{cases} \delta(q_0, 0) & \text{if the number of 1s in } y \text{ has remainder 0 when divided by 4} \\ \delta(q_1, 0) & \text{if the number of 1s in } y \text{ has remainder 1 when divided by 4} \\ \delta(q_2, 0) & \text{if the number of 1s in } y \text{ has remainder 2 when divided by 4} \\ \delta(q_3, 0) & \text{if the number of 1s in } y \text{ has remainder 3 when divided by 4} \end{cases} \\
&= \begin{cases} q_0 & \text{if the number of 1s in } s = y0 \text{ has remainder 0 when divided by 4} \\ q_1 & \text{if the number of 1s in } s = y0 \text{ has remainder 1 when divided by 4} \\ q_2 & \text{if the number of 1s in } s = y0 \text{ has remainder 2 when divided by 4} \\ q_3 & \text{if the number of 1s in } s = y0 \text{ has remainder 3 when divided by 4} \end{cases}
\end{aligned}$$

So, $P(s)$ follows in this case.

Case $a = 1$: In this case the number of 1s in y is increased by 1 upon appending 1, so

$$\begin{aligned}
\delta^*(q_0, s) &= \delta^*(q_0, y1) = \delta(\delta^*(q_0, y), 1) \\
\text{by } P(y) &= \begin{cases} \delta(q_0, 1) & \text{if the number of 1s in } y \text{ has remainder 0 when divided by 4} \\ \delta(q_1, 1) & \text{if the number of 1s in } y \text{ has remainder 1 when divided by 4} \\ \delta(q_2, 1) & \text{if the number of 1s in } y \text{ has remainder 2 when divided by 4} \\ \delta(q_3, 1) & \text{if the number of 1s in } y \text{ has remainder 3 when divided by 4} \end{cases} \\
&= \begin{cases} q_1 & \text{if the number of 1s in } s = y1 \text{ has remainder 1 when divided by 4} \\ q_2 & \text{if the number of 1s in } s = y1 \text{ has remainder 2 when divided by 4} \\ q_3 & \text{if the number of 1s in } s = y1 \text{ has remainder 3 when divided by 4} \\ q_0 & \text{if the number of 1s in } s = y1 \text{ has remainder 0 when divided by 4} \end{cases}
\end{aligned}$$

So, $P(s)$ follows in this case (the branches of the invariant are simply permuted).

So, in both possible cases $P(s)$ follows.

I conclude, by structural induction, that $\forall s \in \{0,1\}^*, P(s)$.

Since all four possible states are listed in the invariant, I know that $P(s)$ means that s drives the machine to state q_0 if, and only if, s has a multiple of 4 1s. So this machine accepts the language.

4. Design an iterative binary search algorithm that is correct with respect to the following precondition/postcondition pair:

Precondition: `A` has elements that are comparable with `x`, `|A|=n>0`, and `A` is sorted in non-decreasing order.

Postcondition: `binSearch(x, A)` terminates and returns an index p that satisfies:

$$\begin{aligned} A[0\ldots p] &\leq x < A[p+1\ldots n-1] \\ -1 &\leq p \leq n-1 \end{aligned}$$

Prove that if the precondition is satisfied, then your algorithm terminates and satisfies the postcondition. **Hint:** Use the approach from lecture (no need to provide the pictures) where you develop a loop invariant as you write the code.

Solution: Initially the precondition guarantees a sorted, non-empty array `A`, and I know nothing about how any of the elements compare to `x`. After i loop iterations, I'd like to reduce the scope of this ignorance to the subarray `A[b ... e]`, for indices `b` and `e`, in other words

$$A[0 \ldots b_i - 1] \leq x < A[e_i + 1 \ldots n - 1]$$

I can certainly make this initially true if I initialize with

```
b = 0
e = n-1
```

There's still work to do so long as the subarray `A[b ... e]` isn't empty, so my loop condition is

```
while b <= e
```

Within the loop I should cut the remaining search space in half, so I choose `m` midway through

```
m = (b+e) // 2
```

Then I determine whether the midpoint belongs to the portion to the left of `b`, or the portion to the right of `e`:

```
if A[m] <= x :
    b = m+1
else :
    e = m - 1
```

After the last iteration we'll return `p=b-1`, so our completed code looks like:

```
binSearch(A, x) # A non-empty, sorted non-decreasing, comparable to x
  b = 0
  e = len(A) - 1
  while b <= e : # A[0.. b-1] <= x < A[e+1 .. len(A)-1] AND b <= e+1
    m = (b+e) // 2
    if A[m] <= x :
      b = m + 1
    else :
      e = m - 1
  return b-1
```

The loop invariant should be true even after the last iteration of the loop (which explains `b <= e+1`), which gives the claim:

> $P(i)$: If the precondition is satisfied and there are i iterations of the loop,
> $0 \leq b_i \leq e_i + 1 \leq n$ and $A[0..b_i - 1] \leq x < A[e_i + 1..n - 1]$

Claim: $\forall i \in \mathbb{N}, P(i)$

Proof (mathematical induction on i):

Base case: When $i = 0$, I examine the claim before the loop iterates (i.e., after its 0th iterations). The initial assignment statements set $b_0 = 0 \leq e_i + 1 = n$, since A is a non-empty array and $n \geq 1$, so this part of $P(0)$ holds. Also $A[0..b_0 - 1]$, and $A[e_0 + 1..n - 1]$ are both empty subarrays, so the $P(0)$ holds vacuously for them.

Induction step: Assume $i \in \mathbb{N}$, $P(i)$, and that there is an $(i+1)$th iteration of the loop.

Since there is another iteration of the loop, I know that $b_i \leq e_i$, so $m = (b_i + e_i)//2$ has $m \geq 2b_i//2 = b_i$ and $m \leq 2e_i//2 = e_i$. When the `if` statement is evaluated, the program takes one of two paths:

Case 1: If $A[m] \leq x$, the the program executes $b_{i+1} = m + 1$, and $e_{i+1} = e_i$, so

$$\begin{aligned} 0 &\leq b_i \qquad \text{\# by IH} \\ &\leq m \leq m+1 = b_{i+1} \qquad \text{\# by construction of } m \\ &\leq e_i + 1 = e_{i+1} + 1 \qquad \text{\# by construction of } m \\ &\leq n \qquad \text{\# by IH} \end{aligned}$$

So $0 \leq b_{i+1} \leq e_{i+1} + 1 \leq n$. From the IH, $0 \leq b_i$, and since array A is sorted and $A[m = b_{i+1} - 1] \leq x$, we have $A[0..b_{i+1} - 1] \leq x$, and, from the IH and $e_{i+1} = e_i$, we have $x < A[e_{i+1}..n-1]$.

Case 2: If $A[m] > x$ the the program executes $e_{i+1} = m - 1$, and $b_{i+1} = b_i$, so

$$\begin{aligned} 0 &\leq b_i = b_{i+1} \qquad \text{\# by IH} \\ &\leq m = e_{i+1} + 1 \qquad \text{\# by construction} \\ &\leq e_i \leq n \qquad \text{\# by IH} \end{aligned}$$

So $0 \leq b_{i+1} \leq n_{i+1} + 1 \leq n$. From the IH and the fact that $b_{i+1} = b_i$, we have $A[0..b_{i+1}-1] \leq x$. Since A is sorted and $A[m = e_{i+1}+1] > x$ we have $A[e_{i+1}+1..n-1] > x$.

In both cases, $P(i)$ implies $P(i+1)$, so for any $i \in \mathbb{N}$, $P(i) \Rightarrow P(i+1)$.

I conclude $\forall i \in \mathbb{N}, P(i)$, by mathematical induction.

Partial correctness: I assume the precondition is satisfied and that, once executed, the loop eventually terminates.

Since the look terminates, I'll assume that the loop condition is violated after some $k \in \mathbb{N}$ iterations. By the loop invariant $P(k)$, I have

$$0 \leq b_k \leq e_k + 1 \leq n$$

...and since the loop condition fails, $b_k > e_k$. Since b and e are integers (loop indices, and we perform integer arithmetic on them), I have $b_k > e_k$ and $b_k \not> e_k + 1$ implies $b_k = e_k + 1.$, and the code returns $p = b_k - 1 = e_k$. By the loop invariant $P(k)$, I also have:

$$A[0..p = b_k - 1] \leq x < A[p + 1 = e_k + 1..n-1]$$

Also, by the loop invariant $P(k)$, I know $0 \leq b_k \leq e_k + 1 \leq n$, so $p = b_k - 1 \geq 0 - 1 == 1$, and also $p = e_k \leq n - 1$. So, given termination, the postcondition follows from the precondition.

Termination: Consider the sequence $\langle e_i + 1 - b_i \rangle$, where e_i and b_i are the values of e and b after i loop iterations. Since e and b are initialized to $n - 1$ and 0 initially, and have only integer math performed on them, the sequence is clearly an integer. Furthermore, the loop invariant $P(i)$ guarantees that $e_i + 1 \geq b_i$, so the terms of the sequence are non-negative, hence this is a sequence of natural numbers.

Assume that an $(i+1)$th iteration of the loop occurs. There are two cases:

Case 1: If $A[m] \leq x$, then $b_{i+1} = m + 1$, and $e_{i+1} = e_i$,

$$e_{i+1} + 1 - b_{i+1} = e_i - m - 1 < e_i - b_i \qquad \text{\# since } m \geq b_i$$

So, in this case $\langle e_{i+1} + 1 - b_{i+1} \rangle$ is strictly less than $\langle e_i + 1 - b_i \rangle$.

Case 2: If $A[m] > x$, then $b_{i+1} = b_i$ and $e_{i+1} = m - 1$, so

$$e_{i+1} + 1 - b_{i+1} = m - b_i < e_i + 1 - b_i \qquad \# \text{ since} m \leq e_i$$

So, in this case $\langle e_{i+1} + 1 - b_{i+1} \rangle$ is strictly less than $\langle e_i + 1 - b_i \rangle$.

In both cases, the sequence $\langle e_i + 1 - b_i \rangle$ is strictly decreasing. A decreasing sequence in $\mathbb{N}$ is finite, so there are finitely many loop iterations, and the loop must terminate.

Correctness: I have shown that the loop terminates, and that given the precondition, execution, and termination, the post condition follows. So, the algorithm is correct with respect to its precondition and postcondition.