MAU11602 Assignment 9, Due Wednesday 10 April 2024
Solutions

1. Find all the monoid homomorphisms from $(N, +)$, i.e. the monoid whose set is natural numbers and whose operation is addition, to itself.
Hint: Describe them in terms of what value the function takes at 1. Solution: Suppose h is a homomorphism. Then $h(m) = m \cdot h(1)$. This is easily proved by induction. $h(0) = 0$ and if $h(m) = m \cdot h(1)$ then

$$h(m+1) = h(m) + h(1) = m \cdot h(1) + h(1) = (m+1) \cdot h(1).$$

So all homomorphisms must be of the form $h(m) = m \cdot a$ for some $a \in N$. Conversely, if h is of this form then it is a semigroup homomorphism because

$$h(j+k) = (j+k) \cdot a = j \cdot a + k \cdot a = h(j) + h(k).$$

It's a monoid homomorphism because, in addition,

$$h(0) = 0 \cdot a = 0.$$

2. Consider the language of even integers. As with the integers we'll normalise things so that each even integer has a unique representation by getting rid of leading zeroes, double minus signs, etc. This is a regular language so it can be described in the following ways:

   (a) a regular expression

   (b) a strongly deterministic finite state automaton

   (c) a regular grammar

   Give an example of each.
   Hint: This is the kind of problem where a bit of extra time thinking about different ways to solve the problem can end up saving you some time.
   Solution: You're presumably familiar with the fact that the even integers are precisely those whose last digit is even. So a string of characters is an even integer if it is an integer and the last character is a `0`, `2`, `4`, `6`, or `8`. We'll use this for each of the required constructions.

   (a) To come up with a regular expression we mimic the procedure used for the integers in the notes and in lecture. `0|1|2|3|4|5|6|7|8|9` matches single digits and `(0|1|2|3|4|5|6|7|8|9)*` represents strings of digits. `1|2|3|4|5|6|7|8|9` matches non-zero digits and `0|2|4|6|8` matches even digits. A positive even integer starts with a non-zero digit and ends with an even digit and can have arbitrary digits in between. This suggests the regular expression

   ```
   (1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|6|7|8|9)*)(0|2|4|6|8)
   ```

   but this would omit the single digit numbers 2, 4, 6 and 8, so we add those in as another possibility:

`(2|4|6|8)|((1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|6|7|8|9)*)(0|2|4|6|8))`.

A non-zero even integer is this, with an optional minus sign, so

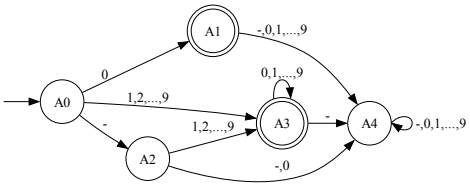
`(|-`
`)((2|4|6|8)|((1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|6|7|8|9)*)(0|2|4|6|8)))`.

An even integer is this or 0, so

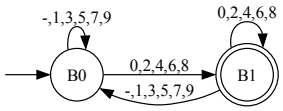
`0|((|-`
`)((2|4|6|8)|((1|2|3|4|5|6|7|8|9)((0|1|2|3|4|5|6|7|8|9)*)(0|2|4|6|8))))`.

This answer is by no means unique. There are other ways to do this.

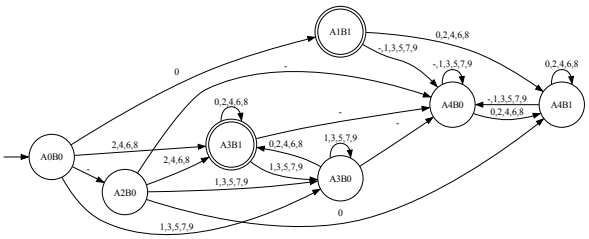
(b) The language we're looking for is the intersection of the language of integers, for which I've already given you a strongly deterministic finite state automaton, and the language of strings whose last character is a `0`, `2`, `4`, `6`, or `8`, for which we can easily construct a strongly deterministic finite state automaton. Once we have those we can construct a strongly deterministic finite state automaton for the intersection by the product construction. Now that we have a plan it's fairly easy to carry it through. The finite state automaton from the integers was



A simple strongly deterministic finite state automaton which recognises strings ending with `0`, `2`, `4`, `6`, or `8` is



A strongly deterministic finite state automaton for the intersection is



You could put in the states A0B1, A1B0 and A2B1 if you wanted to, but they're not reachable so there's no point. The finite state automaton above is not quite the simplest possible. If you constructed one via the Myhill-Nerode Theorem, for example, you'd get one where the states A4B0 and A4B1 are combined, but you'd have to do significantly more work to get it.

(c) It's straightforward to get a grammar from a finite state automaton. There's a non-terminal for each state and an alternate for each allowed transition. The initial state corresponds to the start symbol and the accepting states are the ones where the empty list is an alternate. Applying this construction to the finite state automaton above gives the grammar

```
a0b0 : "-" a2b0 | "0" a1b1 | "1" a3b0 | "2" a3b1
     | "3" a3b0 | "4" a3b1 | "5" a3b0 | "6" a3b1
     | "7" a3b0 | "8" a3b1 | "9" a3b0
a1b1 : | "-" a4b0 | "0" a4b1 | "1" a4b0 | "2" a4b1
     | "3" a4b0 | "4" a4b1 | "5" a4b0 | "6" a4b1
     | "7" a4b0 | "8" a4b1 | "9" a4b0
a2b0 : "-" a4b0 | "0" a4b1 | "1" a3b0 | "2" a3b1
     | "3" a3b0 | "4" a3b1 | "5" a3b0 | "6" a3b1
     | "7" a3b0 | "8" a3b1 | "9" a3b0
a3b0 : "-" a4b0 | "0" a3b1 | "1" a3b0 | "2" a3b1
     | "3" a3b0 | "4" a3b1 | "5" a3b0 | "6" a3b1
     | "7" a3b0 | "8" a3b1 | "9" a3b0
a3b1 : | "-" a4b0 | "0" a3b1 | "1" a3b0 | "2" a3b1
     | "3" a3b0 | "4" a3b1 | "5" a3b0 | "6" a3b1
     | "7" a3b0 | "8" a3b1 | "9" a3b0
a4b0 : "-" a4b0 | "0" a4b1 | "1" a4b0 | "2" a4b1
     | "3" a4b0 | "4" a4b1 | "5" a4b0 | "6" a4b1
     | "7" a4b0 | "8" a4b1 | "9" a4b0
a4b1 : "-" a4b0 | "0" a4b1 | "1" a4b0 | "2" a4b1
     | "3" a4b0 | "4" a4b1 | "5" a4b0 | "6" a4b1
     | "7" a4b0 | "8" a4b1 | "9" a4b0
```