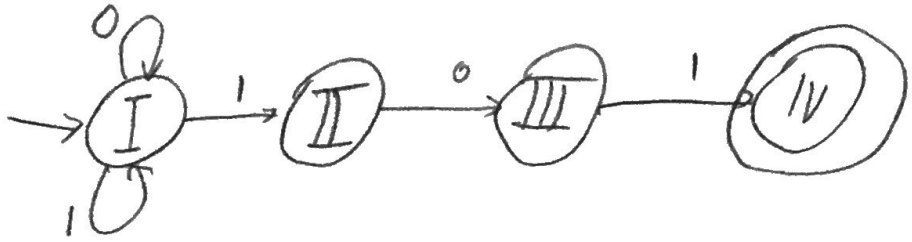
July 10, 2020

# CS360 Quiz 1

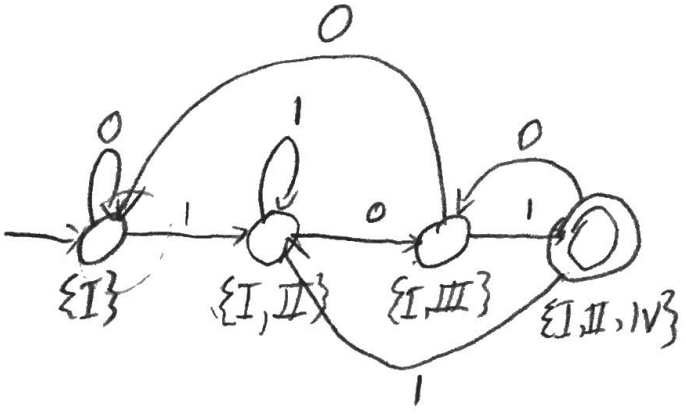
Group members: Shivanshi Nagar, Stephen Hansen, Dennis George

Start Time 12:50 pm
End Time 2:20 pm

1. i:




ii: DFA 的 Transition Table




$\{I\} \xrightarrow{0} \{I\}$

$\xrightarrow{1} \{I, II\}$

$\{I, II\} \xrightarrow{0} \{I, III\}$

$\xrightarrow{1} \{I, II\}$

$\{I, III\} \xrightarrow{0} \{I\}$

$\xrightarrow{1} \{I, II, IV\}$

$\{I, II, IV\} \xrightarrow{0} \{I, III\}$

$\xrightarrow{1} \{I, II\}$

101: $\{I\} \to \{I, II\} \to \{I, III\} \to \{I, II, IV\}$

10101: $\{I\} \to \{I, II\} \to \{I, III\} \to \{I, II, IV\} \to \{I, III\}$

$\downarrow$

$\{I, II, IV\}$

10101: $\{I\} \to \{I, II\} \to \{I, III\} \to \{I, II, IV\}$

$\to \{I, II\} \to \{I, III\} \to \{I, II, IV\}$

2.

**(i)**

**Targets of computation**: Checks if x is a member of the list L

**Data structures used**: List

**Language mechanisms**:

- Selection: A choice is being made based on the (cond) statement. The execution of the program is controlled by (cond)
- Recursion: The (else) statement in the (cond) expression has a recursive call to the (mem) function.
- Exception handling and speculation: One of the risks of recursion is running into a stack overflow. This function has a careful base case with ((null? L) #f), so that the question doesn't infinitely recurse.
- Procedural Abstraction: The programmer now has the ability to use this function whenever they need to perform the action of determining if a value "x" is a member of a list "L". Filter takes in a lambda p which is applied to every element in L to determine which elements to keep and which to remove.

**(ii)**

**Targets of computation**: Insert x into L, if x is not already in L

**Data structures used**: List

**Language mechanisms**:

- Selection: A choice is being made in the (if) statement. The program will have different execution based on the return of (mem x L)
- Procedural Abstraction: The programmer now has the ability to use this function whenever they need to perform the action of adding an element to a list without duplicating.

**(iii)**

**Targets of computation**: Given two sorted lists, return a merged sorted list

**Data structures used**: List

**Language mechanisms**:

- Selection: A choice is being made in the (cond) statement. The program will have different paths of execution.
- Procedural Abstraction: The programmer now has the ability to use this function whenever they need to perform the action merging two sorted lists together.
- Recursion: The (else) statement in the (cond) expression has a recursive call to the (mer) function.

- Exception handling and speculation: Again, to avoid a stack overflow this function has a careful base case with both ((null? fst) snd) and ((null? snd) fst), so that the question doesn't infinitely recurse.

3.

**(i)**

**Targets of computation**: primesto yields a list of prime numbers from 2 to n. intlist generates a list of numbers with range [m, n]. sieve takes a given list and starting at the first element, adds the first element to the result, removes all elements in the list divisible by that first element, and then runs sieve again on the rest of the list. If the first element of the list is 2, then sieve effectively works as a prime sieve. Not-divisible-by uses a lambda to create a filter for checking if a number m is divisible by n. This function returns a lambda, it does not actually perform the computation itself. Filter operates like it does in problem 2, taking a lambda p and and a list L, removing all elements from L where the evaluation of p on L returns false (#f).

**Data structures used**: List

**Language mechanisms**:

- Recursion: filter, sieve, intlist, are using recursion
- Selection: filter has a condition selection
- Procedural abstraction: rather than defining primes as one method, each component of calculating the prime number list is defined as a helper function to abstract out functionality. The function not-divisible-by uses procedural abstraction to define a lambda which is then applied to all elements in L using filter.
- Exception handling and speculation: To avoid a stack overflow the (sieve) and (filter) function has a careful base case with both ((null? L) L) so that the question doesn't infinitely recurse.

**(ii)**

**Targets of computation**: primes gives a list (implementing a stream) of primes starting from 2. intsfrom generates a delayed list of items starting at m - it does so by delaying the cons action per each element. Sieve takes a list and filters out the integers that are not divisible by the first element, i.e. 2 in this case, again producing a delayed list. The list that sieve takes is also delayed and only forces parts of the list when it needs to use them. Filter uses a condition to filter out based on a function again also taking a delayed list and producing a delayed list, only forcing elements of the list to evaluate when necessary. The function being used is not-divisible-by, which is not defined here again so we assume that the lambda used by not-divisible-by is the same as previously defined in 3i (why? Because each element fed as input to not-divisible-by is forced first. The function has no need to force the input again).

**Data structures used**: List (effectively modelling a Stream by delaying the cons expression and using force to evaluate each number in the sieve when necessary.)

**Language mechanisms**:

- Recursion: filter, sieve, intsfrom are using recursion
- Selection: filter has a condition selection, same goes for sieve.
- Procedural abstraction: rather than defining primes as one method, each component of calculating the prime number list is defined as a helper function to abstract out functionality. The function not-divisible-by uses procedural abstraction to define a lambda which is then applied to all elements in L using a filter.
- Exception handling and speculation: Again, to avoid a stack overflow the (filter) function has a careful base case with both ((null? L) L) so that the question doesn't infinitely recurse.

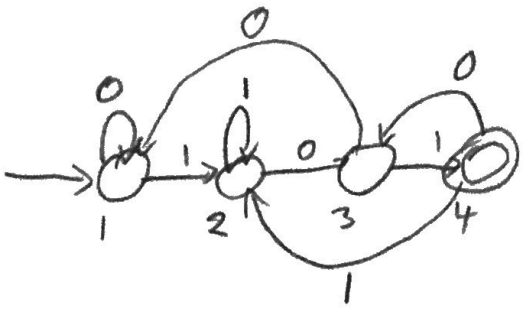
**Similarities**

- Both (i) and (ii) use recursion, selection, procedural abstraction and exception handling and speculation.
- Both operate the same way in terms of the end goal of getting a list of prime numbers from integer 2. The model and abstraction over procedures is generally the same for both; they both use the approach of filtering based on a prime sieve to find a list of prime numbers.

**Differences**

- (i) uses a finite list to list all the primes from 2 to integer n. (ii) uses Stream, an infinite list to give an infinite list of prime numbers from 2 to infinity.
- (ii) uses delay and force as it uses Stream, while (i) does not as it generates a finite list of prime numbers within a finite range.
- (ii) only generates each number in the prime list when necessary due to the use of force and delay. Since the action of cons is delayed per each call to sieve, getting the first item of the prime list requires forcing the cons action. This results in the next item (and next call to cons) being delayed, so, each element is only generated when necessary, by forcing it. Hence why (ii) can have an infinite number of primes; we force each prime number as needed. In (i) since force and delay are not used, all primes from 2 to n are generated at runtime.

4.




Proof A minimal states — choose strings for every pair of states such that one leads to accept, the other rejects.

1, 2: 01  2, 3: 1  3, 4: ε

1, 3: 1  2, 4: ε

1, 4: ε

Each pair of states contains a string sequence such that one accepts and the other rejects. This DFA therefore is in minimal form, hence 4 is the minimal number of states.