

Specification in Z and Finite State Automata

1 Specification in Z

In the *GP_Practice* assignment question a number of the definitions and schemas were provided. These are reproduced here for readability.

The Given System

Types

The given types introduced were

[Person]

[Note]

[Order]

There were two types introduced by definition:

$Time \hat{=} \mathbb{R}$

$Source ::= Visit \mid Pathology \mid Radiology \mid Specialist$

Lastly, there was a data type introduced as a schema:

$Record$
$time : Time$
$source : Source$
$notes : \mathbb{P} Note$
$orders : \mathbb{P} Order$
$(source \neq Visit) \Rightarrow orders = \emptyset$
$\#notes > 0$

Functions on Patient Histories

$sequential : seq Record \rightarrow \mathbb{B}$
$\forall rs : seq Record \bullet sequential(rs) \equiv$ $\forall i, j : \mathbb{N} \bullet (0 < i < j \leq \#rs) \Rightarrow time(rs(i)) < time(rs(j))$

and

$lastTime : seq Record \rightarrow Time$
$\forall rs : seq Record \bullet \#rs > 0 \wedge sequential(rs) \Rightarrow lastTime(rs) = time(last rs)$

The State Schema

$\begin{array}{l} \textit{GP_Practice} \\ \textit{allFiles} : \textit{Person} \rightarrow \textit{seq Record} \\ \textit{now} : \textit{Time} \\ \textit{arrivalTimes} : \textit{Person} \rightarrow \textit{Time} \end{array}$
$\begin{array}{l} \langle \rangle \notin \textit{ran allFiles} \\ \forall p : \textit{Person}, rs : \textit{seq Record} \bullet (p \mapsto rs) \in \textit{allFiles} \Rightarrow \textit{sequential}(rs) \\ \forall p : \textit{Person}, rs : \textit{seq Record} \bullet (p \mapsto rs) \in \textit{allFiles} \Rightarrow \textit{now} > \textit{lastTime}(rs) \\ \#\textit{ran arrivalTimes} = \#\textit{arrivalTimes} \\ \forall t : \textit{Time} \bullet t \in \textit{ran arrivalTimes} \Rightarrow t \leq \textit{now} \end{array}$

The Questions and the Answers

Question 1.1

Write a Z schema, called *GetPatients*, that specifies a query that gets the set of people who are (or have been) patients in this practice.

Answer 1.1

$\begin{array}{l} \textit{GetPatients} \\ \exists \textit{GP_Practice} \\ \textit{allPatients!} : \mathbb{P} \textit{Person} \\ \textit{allPatients!} = \textit{dom allFiles} \end{array}$

There is no error case for this query.

Question 1.2

Specify, using Z, the event of a person arriving at the doctor's surgery and joining the queue. Say what the error case for this schema is.

Answer 1.2

$\begin{array}{l} \textit{PatientArrives}_o \\ \Delta \textit{GP_Practice} \\ p? : \textit{Person} \end{array}$
$\begin{array}{l} p? \notin \textit{dom arrivalTimes} \\ \textit{arrivalTimes}' = \textit{arrivalTimes} \cup \{p? \mapsto \textit{now}\} \\ \textit{now}' = \textit{now} \\ \textit{allFiles}' = \textit{allFiles} \end{array}$

The error case for this operation is when the person does not have an arrival time.

Question 1.3

Explain, in English, the invariants of the system state schema.

Answer 1.3

There are five predicates in the state schema.

The first predicate ensures that no patient has an empty record.

The second says that the history of every patient is ordered by time of entry.

The third predicate holds when all record times are in the past.

The fourth says that the times associated with patients in *arrivalTimes* are all distinct.

The last says that no patients have arrival times that are in the future.

Question 1.4

Why is the predicate $\# \text{dom } arrivalTimes = \# arrivalTimes$ a system invariant? If it is, why is it not in the state schema predicate part?

Answer 1.4

It is a theorem that if f is an arbitrary function then $\# \text{dom } f = \# f$. Therefore, the suggested predicate would qualify as a system invariant. It is not included in the schema because it can be derived.

Question 1.5

Specify, in Z , a query, called *CurrentPatient* that gives the attending patient who joined the queue first. Handle the situation where there is no such patient as an error case.

Answer 1.5

$$\frac{\begin{array}{l} \text{CurrentPatient}_o \\ \exists GP_Practice \\ \text{currentPatient!} : Person \end{array}}{\begin{array}{l} \#arrivalTimes > 0 \\ \text{currentPatient!} \in \text{dom } arrivalTimes \\ \forall p : People \bullet p \neq \text{currentPatient!} \Rightarrow \\ \quad \quad \quad arrivalTimes(p) > arrivalTimes(\text{currentPatient!}) \end{array}}$$

The error case corresponds to the negation of the single precondition:

$NoCurrentPatient$ $\exists GP_Practice$ $message! : PossibleMessages$
$\#arrivalTimes = 0$ $message! = NoPatient$

Question 1.6

Specify the operation of a radiology report being added to the patient's file.

Answer 1.6

$AddRadiologyReport_o$ $\Delta GP_Practice$ $p? : Person$ $ns? : \mathbb{P} Note$ $newRec : Record$
$p? \in \text{dom } allFiles$ $ns? \neq \emptyset$ $time(newRec) = now$ $source(newRec) = Radiology$ $notes(newRec) = ns?$ $orders(newRec) = \emptyset$ $front(allFiles' p?) = allFiles(p?)$ $last(allFiles' p?) = newRec$ $\forall p : Person \bullet p \neq p? \Rightarrow allFiles'(p) = allFiles(p)$ $now' = now$ $arrivalTimes' = arrivalTimes$

Question 1.7

Specify a query $LatestPathology_o$ which asks for the most recent pathology report for a patient. Say what the error case(s) are.

Answer 1.7

$LatestPathology_o$ $\exists GP_Practice$ $p? : Person$ $report! : \mathbb{P} Note$
$p? \in \text{dom } allFiles$ $\exists i : \mathbb{N}_1 \bullet source(allFiles p? i) = Pathology$ $\exists i : \mathbb{N}_1 \bullet (notes(allFiles p? i) = report!) \wedge (source(allFiles p? i) = Pathology) \wedge$ $\neg \exists j : \mathbb{N}_1 \bullet (j > i) \wedge (source(allFiles p? j) = Pathology)$

or alternatively,

$LatestPathology_o$ $\exists GP_Practice$ $p? : Person$ $report! : \mathbb{P} Note$
$p? \in \text{dom } allFiles$ $(historyFilter (\lambda r \bullet source(r) = Pathology) (allFiles p?)) \neq \emptyset$ $pathRpt! = last(historyFilter (\lambda r \bullet source(r) = Pathology) (allFiles p?))$

The two error cases for this operation are:

- i) The person is not a patient,
- ii) The person is a patient but has no pathology reports on file.

Question 1.8

Specify a query, *Waiters*, which gives as a sequence the people who are in the waiting room.

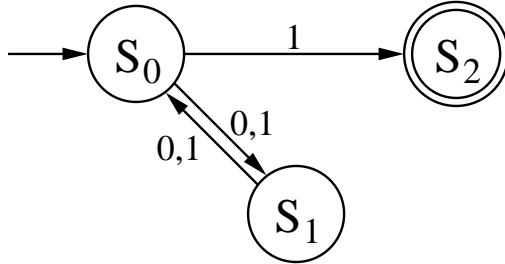
Answer 1.8

$Waiters$ $\exists GP_Practice$ $waiters! : \text{seq } Person$
$\text{ran } waiters! = \text{dom } arrivalTimes$ $\#waiters! = \#arrivalTimes$ $\forall p_1, p_2 : Person, t_1, t_2 : Time \bullet$ $((p_1 \mapsto t_1) \in arrivalTimes) \wedge ((p_2 \mapsto t_2) \in arrivalTimes) \wedge (t_1 < t_2) \Rightarrow$ $\exists i, j : \mathbb{N}_1 \bullet (i < j) \wedge (waiters!(i) = p_1) \wedge (waiters!(j) = p_2)$

2 Finite State Automata and Regular Expressions

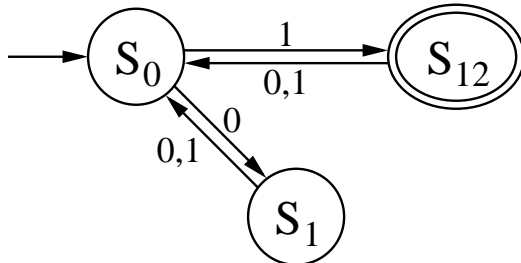
2.1 Construction of FSAs [2.5 marks]

- (a) [1 mark] Construct a non-deterministic finite automaton (NFA) which accepts strings of binary digits (0 or 1), of odd length, where the last digit is 1. (You should be able to find one with only three states).



- (b) [1.5 marks] Use the algorithm described in lectures to obtain a deterministic finite automaton (DFA) which accepts the same language. Your answer should include an explanation of how you obtained each transition you show in your DFA.

Note: if you follow the algorithm given in lectures fully, you will get rather a lot of states. Many of these states will not be reachable from the DFA's start state. You are *not* required to include these non-reachable states (or, of course, the transitions leading from them). It is recommended, therefore, that you construct the DFA beginning at its start state, and from there find all reachable states (and the transitions leading from them), and include only those.



The states labelled S_0, S_1, S_{12} in the diagram are the states which would be labelled $\{S_0\}, \{S_1\}, \{S_1, S_2\}$ according to the description given in lectures.

Start in S_0 . With input 1, can go to S_1 or S_2 in the NFA, so go to $\{S_1, S_2\}$, ie, S_{12} in the DFA. With input 0, can go only to S_1 in the NFA, so go to $\{S_1\}$, ie, S_1 , in the DFA.

From S_{12} (ie, $\{S_1, S_2\}$) in the DFA, could be in either S_1 or S_2 in the NFA:

- input 1, from S_1 , goes to S_0 ; from S_2 , can't go anywhere; thus goes to S_0 in the DFA
- input 0, same as for input 1

From S_1 in the DFA, ie, can only be in S_1 in the NFA: input 0 or 1, goes to S_0 (only), thus goes to S_0 in the DFA.

No other states of the DFA can be reached from any of these three; that is, the diagram shown contains all the transitions from all the states shown, so these states are all we need.

- (c) (Only if unable to do (b)) [0.5 marks] : Construct (by any method) a DFA which accepts the language of (a).

2.2 Regular Expressions and NFAs [2 marks]

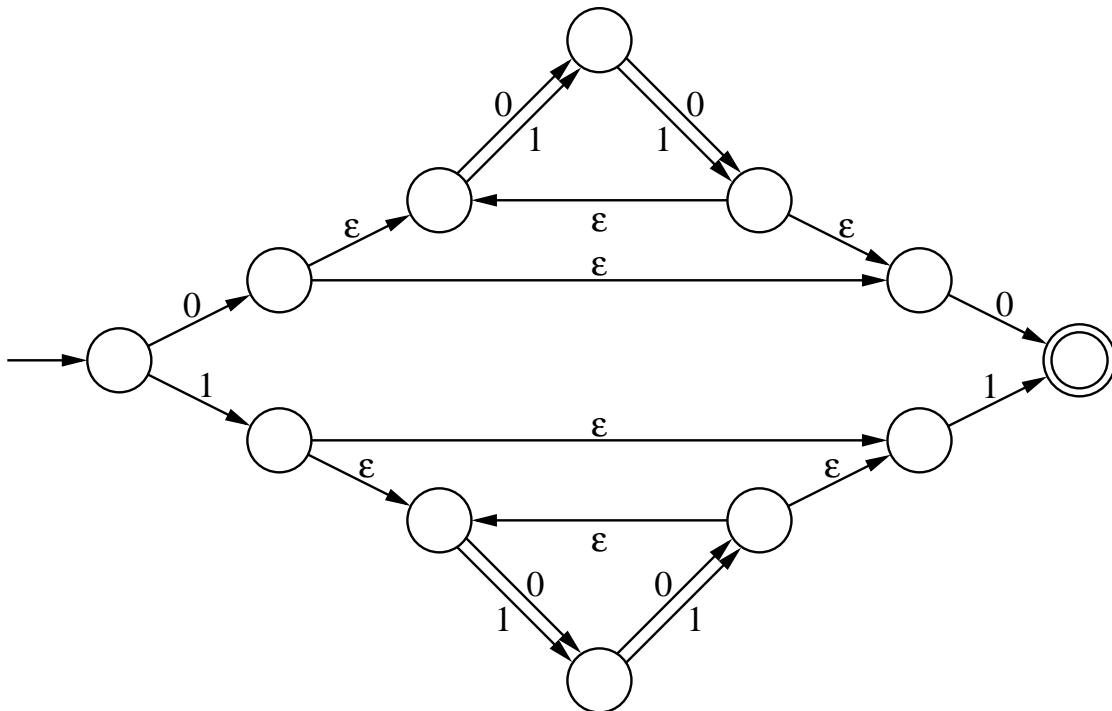
Consider the language L consisting of strings of binary digits (0 or 1), which are of even length and in which the first and last digits are the same.

- (a) [1 mark] Give a regular expression which describes this language L

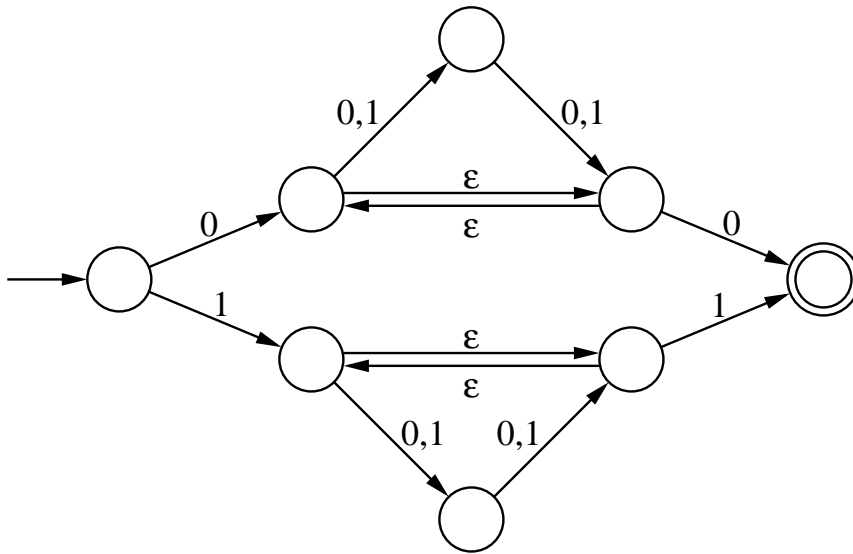
Answer: $0((0|1)(0|1))^*0 | 1((0|1)(0|1))^*1$

- (b) [1 mark] Derive from this regular expression an NFA which accepts L (when following the method in lectures, you may omit unnecessary ϵ -transitions)

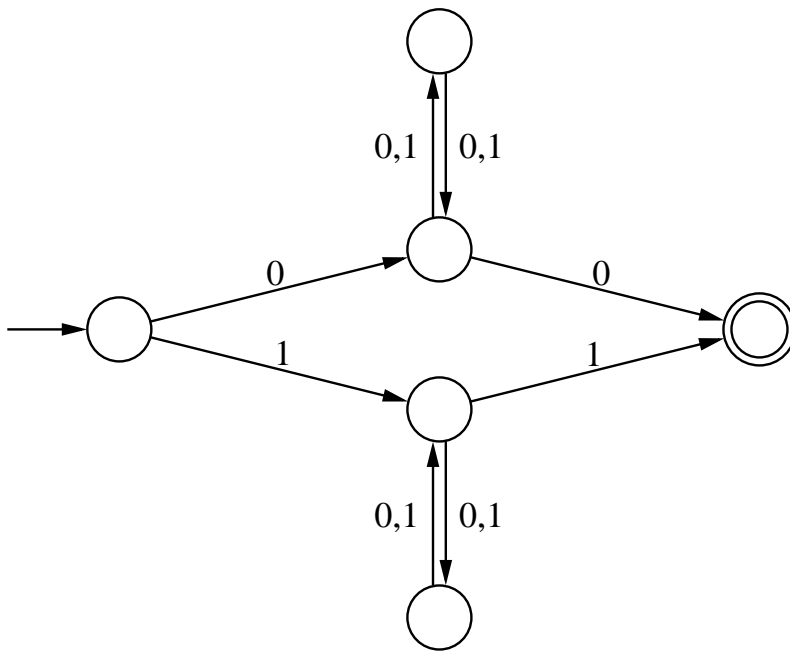
The first solution shown is close to the result you get using the method in lectures (although omitting some ϵ -transitions).



The second solution shown has fewer states; notice that although you can go back and forth along a pair of ϵ -transitions between the same states, the set of strings accepted is the same. Notice our notation: we use a single arrow labelled “0,1” instead of two arrows labelled “0” and “1” between the same states.

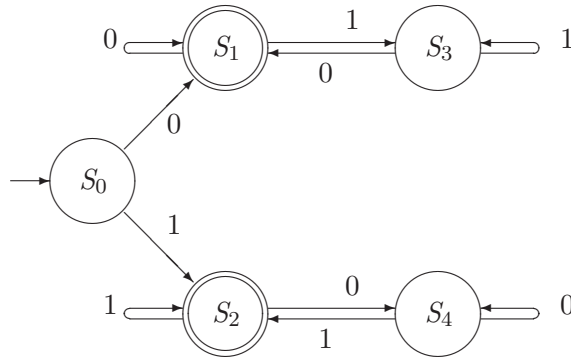


In the second solution, two states joined by an ϵ -transition in each direction are necessarily equivalent, so we can combine them into one, thus giving the third solution.



2.3 Proofs Involving Automata [3 marks]

Consider this automaton, whose alphabet Σ is $\{0, 1\}$.



Prove formally that a string of the form $0\alpha x$ is accepted by this automaton if and only if $x = 0$. (Hint: there is an intermediate result which you will have to prove by induction either on the length or on the structure of α).

Solution The required intermediate result is that $N^*(S_0, 0\alpha)$ is S_1 or S_3 . This can be proved by induction on the length of α .

Base case, $\alpha = \epsilon$. Then we have $N^*(S_0, 0\alpha) = N(S_0, 0) = S_1$

Step case, let $\alpha = \alpha'y$, so α' is shorter than α , so we can use the induction hypothesis that $N^*(S_0, 0\alpha')$ is S_1 or S_3 .

Then $N^*(S_0, 0\alpha) = N^*(S_0, 0\alpha'y) = N^*(N^*(S_0, 0\alpha'), y)$ by the ‘‘Append’’ theorem, that is, $N(N^*(S_0, 0\alpha'), y)$

Since $N^*(S_0, 0\alpha')$ is S_1 or S_3 , and y is 0 or 1, there are four cases for $N(N^*(S_0, 0\alpha'), y)$: it may be $N(S_1, 0)$, $N(S_1, 1)$, $N(S_3, 0)$ or $N(S_3, 1)$. These are all either S_1 or S_3 . This proves the intermediate result.

Now, if $x = 0$, the argument is similar to the above, $N^*(S_0, 0\alpha x) = N(N^*(S_0, 0\alpha), 0)$ which is either $N(S_1, 0)$ or $N(S_3, 0)$. Both of these are S_1 , which is a final state, so the string $0\alpha x$ is accepted.

Conversely we have to show that if x is not 0, then the string $0\alpha x$ is not accepted. For this case, $N^*(S_0, 0\alpha x) = N(N^*(S_0, 0\alpha), 1)$ which is either $N(S_1, 1)$ or $N(S_3, 1)$. Both of these are S_3 , which is not a final state, so the string $0\alpha x$ is not accepted. QED

Alternative solution method As an alternative approach, if you wanted to treat a string like a Haskell list, and do structural induction on lists, where you would, for a string $\alpha = a\alpha'$, prove some $P(\alpha)$ using inductive hypothesis $P(\alpha')$ you could do it like this:

Your intermediate lemma would have to be that if S is S_1 or S_3 , then $N^*(S, \alpha)$ is S_1 or S_3 . You could prove this by induction on α , proving that it holds for $\alpha = a\alpha'$, given the inductive hypothesis that it holds for α' . From this, then show that $N^*(S_0, 0\alpha x)$ is S_1 for $x = 0$ and is S_3 for $x = 1$.

2.4 Proof that a language is not regular [2.5 marks]

Let L be the set of strings over $\Sigma = \{0, 1, 2\}$ in which (considering each symbol as a separate number) the sum of all the 2s is greater than the sum of all the 1s. (That is, if

the string contains n_0 occurrences of 0, n_1 occurrences of 1, and n_2 occurrences of 2, the condition is that $2n_2 > n_1$).

Show that the language L is not regular.

Solution The general method of proof of such a result is to show that an automaton which accepts L would have to have infinitely many different states. The method is to assume that there is a DFA which accepts L , and find an infinite set of strings $\{\alpha_n \mid n = 0, 1, \dots\}$ such that all the $N^*(S_0, \alpha_n)$ are different (for start state S_0).

Here we can use the strings $\alpha_n = 2^n$. Let Q_n be the state $N(S_0, \alpha_n)$. Then consider which of the strings 1^m can be accepted by the automaton, *starting in state* Q_n ; that is, consider which m give $N^*(Q_n, 1^m) \in F$. (F is the set of final states).

Clearly $N^*(Q_n, 1^{2n}) \in F$ (because $2^n 1^{2n} \in L$), and $N^*(Q_n, 1^m) \notin F$ for $m \neq 2n$ (because then $2^n 1^m \notin L$). Therefore $Q_n \neq Q_k$ for $k \neq n$, so the set $\{Q_n \mid n = 0, 1, \dots\}$ is an infinite set of distinct states. This cannot happen in a Deterministic *Finite* Automaton.

Therefore L is not accepted by any DFA, ie, L is not regular.