

Assignment 2

Specification in Z and Finite State Automata Due: 11am on Monday 26th September 2011

The submission of your assignment must be done via the assignment boxes in the student foyer. Failure to submit by the due date will result in late penalties of ten percent per weekday. Other arrangements that are required because of truly exceptional circumstances need to be negotiated before your deadline.

Your submission must be well-presented on clean A4 paper with a fully completed standard cover page, including your *tutor's name* and your *tutorial group*. Failure to follow this instruction will result in a penalty. The COMP2600 Assignments page has a link to an appropriate cover.

1 System Specification in Z

The system we take is a GP's surgery where patients queue for a consultation. When the doctor sees someone a record of the visit is constructed and added to their file; that record would include a copy of any prescription the patient leaves with.

1.1 Types

The doctor deals with a set of *patients* but since this set grows over time, we need an a given type which includes all potential patients. *Person* fits the bill:

[Person]

A given type that encompasses items of information in a patient's medical history is:

[Note]

Note covers all the sorts of information that needs to be recorded. It includes measurements of quantities such as blood pressure, symptoms observed, pathology results, information from radiologists or other medical specialists. A *note* can also be a diagnosis or the record of advice given to the patient.

As a result of judgements, conveyed to the patient and added to the history as notes, the doctor may write out prescriptions, referrals or requests for XRays etc. These documents form the given type

[Order]

Every record in a patient's file indicates when it was added to the history. There will also be a global variable (state variable), *time*, of type *Time* which gives the current time, *in*

minutes since the establishment of the practice.

$$Time \cong \mathbb{R}$$

Each visit by a patient to the doctor's rooms results in that patient's medical history being augmented by an object of type *Record* (given below) which mentions the doctor's observations, conclusions, requests and referrals. The same type is used for reports that come to the GP from elsewhere – blood tests, X-Rays, operation reports etc.

The discrete data type *Source* indicates where any record comes from.

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

<i>Record</i>
<i>time</i> : <i>Time</i>
<i>source</i> : <i>Source</i>
<i>notes</i> : $\mathbb{P} Note$
<i>orders</i> : $\mathbb{P} Order$
<hr style="width: 100%;"/>
$(source \neq Visit) \Rightarrow orders = \emptyset$
$\#notes > 0$

When one defines a data-type in *Z* using a schema (as we did above) one can use the variables in the declaration part as selectors to access the components of any object of that type. In the case at hand, if *rec* is an object of type *Record* then the values of the components of *rec* are given by the expressions *time(rec)*, *source(rec)* etc.

1.2 Functions on Histories

A requirement on the system is that the records in each patient's file should be ordered by time. This inspires the following predicate on histories. (Reminder: By the definition of sequences in *Z*, the term *rs(n)* is the *n*'th element in *rs*.)

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

The following function gives the time of the last entry in a patient history:

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

The functions for dissecting sequences in *Z* are called *head*, *tail*, *front* and *last*. You use them to split a sequence after the first element or before the last item. (They have the same semantics as the Haskell functions that break up *lists*.)

On the subject of useful operations on sequences, you can test for membership using the infix operator *in*, concatenate two sequences with the $\hat{\wedge}$ operator, as illustrated in the definition of function *filterHistory* (below), and reverse a sequence with *rev*.

Note that the following axiom *specifies* the function while not being a recipe for efficient computation. Compare it to a normal recursive function definition of *filter*.

$$\begin{array}{l}
 \hline
 \text{filterHistory} : \text{seq Record} \rightarrow (\text{Record} \rightarrow \mathbb{B}) \rightarrow \text{seq Record} \\
 \hline
 \forall p : (\text{Record} \rightarrow \mathbb{B}) \bullet \text{filterHistory } \langle \rangle p = \langle \rangle \\
 \forall p : (\text{Record} \rightarrow \mathbb{B}), rs : \text{seq Record}, \bullet \\
 \quad \text{filterHistory } \langle rs \rangle p = \text{if } p(rs) \text{ then } \langle rs \rangle \text{ else } \langle \rangle \\
 \forall p : (\text{Record} \rightarrow \mathbb{B}), rs_1, rs_2 : \text{seq Record} \bullet \\
 \quad \text{filterHistory } (rs_1 \hat{\ } rs_2) p = (\text{filterHistory } rs_1 p) \hat{\ } (\text{filterHistory } rs_2 p) \\
 \hline
 \end{array}$$

1.3 State Specification Schema

What follows is a schema that describes the state of the doctor's practice.

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

The global variable *allFiles* is the set of patient histories, the variable *now* is the current time (from which date and time-of-day could be derived), and *arrivalTimes* is a mapping of patients to the respective times they arrived for their appointment. The domain of the partial function *arrivalTimes* is just those people who are waiting for their turn, together with the one being seen. The patients are seen in order of arrival (and then disappear from the domain of *arrivalTimes*).

The initial state of the system is given by the *Initial* schema:

$$\begin{array}{l}
 \hline
 \text{Initial} \\
 \hline
 \text{GP_Practice} \\
 \hline
 \text{allFiles} = \emptyset \\
 \text{time} = 0 \\
 \text{arrivalTimes} = \emptyset \\
 \hline
 \end{array}$$

The Assignment Questions

- Q 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. Use *allPatients!* as the name of the output variable. [1 mark]

- Q 1.2) Specify, using Z , the event of a person arriving at the doctor's surgery and joining the queue. Call the schema for this operation $PatientArrives_o$. Say what the error case for this schema is. [1 mark]
- Q 1.3) Explain, in English, the invariants of the system state schema. [2 marks]
- Q 1.4) Is the predicate $\# \text{ dom } arrivalTimes = \# arrivalTimes$ a system invariant? If it is, why is it not in the state schema predicate part? If it is not, why not? [1 mark]
- Q 1.5) Specify, in Z , a query, called $CurrentPatient$ that gives the attending patient who joined the queue first. The output variable should be $currentPatient!$. Handle the situation where there is no such patient as an error case schema. [2 marks]
- Q 1.6) Specify the operation of a radiology report (that has just arrived at the practice) being added to the patient's file. Call the schema $AddReport_o$ and use $p? : Person$ and $ns? : \mathbb{P} Note$ as the input variables. ($AddReport_o$ is the error-free case.)
Hint: You must remember the fact that the predicate part consists of postconditions that relate the values of the updated global variables to the previous values. [1 mark]
- Q 1.7) Specify a query $LatestPathology_o$ which asks for the most recent pathology report for a patient. Describe the error case(s). [1 mark]
- Q 1.8) Specify a query, $Waiters$, which gives *as a sequence* the people who are in the waiting room (i.e. the patients in the domain of $arrivalTimes$). The output variable should be $waiters! : \text{seq } Person$. [1 mark]

2 Finite State Automata and Regular Expressions

2.1 Construction of FSAs

- (a) 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) 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.

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

2.2 Regular Expressions and NFAs

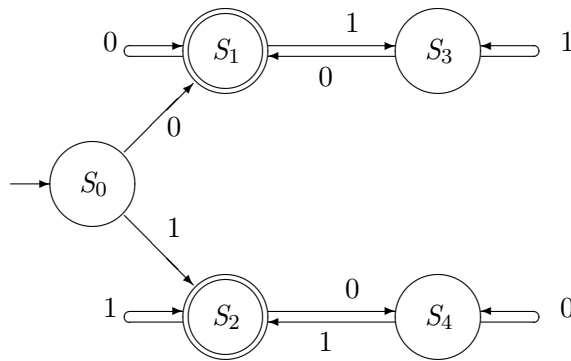
Consider the language L consisting of strings of binary digits (0 or 1), which are of even length and in which the first and last digits are the same.

- Give a regular expression which describes this language L
- Derive from this regular expression an NFA which accepts L (when following the method in lectures, you may omit unnecessary ϵ -transitions)

Note: the precedence of the regular expression operators is $*$, then concatenation, then $|$. Thus $ab^* | c^*d$ means $(a(b^*)) | ((c^*)d)$

2.3 Proofs Involving Automata

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).

2.4 Proof that a language is not regular

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.