0.1 A refinement of algebraic specifications into state-based implementations

In a large number of situations regarding communicating systems the tracking and the history of input messages plays an important role. The behaviour of a component of such a system will depend in a great number of cases on the timing of the messages.
The tracking in general can either be done by using a large number of states, each one consisting in a certain history of messages, or using a buffer for storing the messages. The large number of states would consist in the fact that one would need to form the Cartesian product of the states combined with the received input. Then each state would be combined with a certain history and one could transition between those states. The second possibility would be to initialise a buffer. The buffer would store the important information and could do certain comparisons between values and implications. In the following a buffer will be used.
Consider an SPF which is time sensitive and produces some input messages as output only if the input is repeated in a certain matter defined by the following equations. Let $x \in M^{\underline{\omega}}, a, b \in M, a \neq b$.

1. f(<>) = <>,
2. f (<a>) = <✓>,
3. f(✓& x) = ✓& f(x),
4. f(a & a & x) = ✓& a & f(x),
5. f(a & b & x) = ✓& f(b & x),
6. f(a & ✓& a & x) = ✓& ✓& a & f(x),
7. f(a & ✓& b & x) = ✓& ✓& f(b & x),
8. f(a & ✓& ✓& x) = ✓& ✓& ✓& f(x).

Note that the function does not change the length of the stream. This means that the input and the output stream have the same number of elements. This function can be seen as a login function using a password. If the correct password will not be typed in a certain time the login will be aborted and one will need to start from the beginning because the previous input is lost.
The empty stream will be mapped by this function onto the empty stream, a stream consisting only of one element will be mapped onto a checkmark. Checkmarks will always be mapped onto checkmarks again.
Looking a little closer at the function f one can see that if the element a is repeated two times one after another, the second a will not be transformed into a checkmark but the a will be given back as an output. If another letter follows the a, starting from there another computation will be started of the function f. If a checkmark is followed, then the second element after the first a will be evaluated before starting a new computation (see rules 6-8). In the other case if the next a is more than two elements away, the first one will not be considered anymore.
This means that the special buffer property can be seen during the computation of the function f only if the same letters will be repeated directly after one another or between

Password

<u>*Password:*</u> $S = \{(Init, x), (F1, x), (F2, x)\}, M = \{-, x\}, I = (Init, -)$

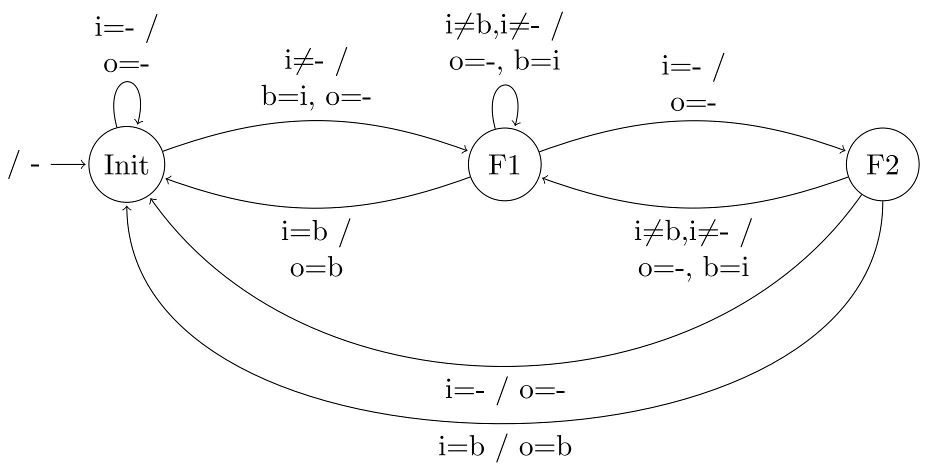

Figure 1: Password Automaton

the two a maximum of one checkmark will be sent. If a letter is followed by another letter then the first will be "lost" and transformed into a checkmark.
A state-based implementation shall be given for this SPF. The automaton will contain explicit (instead of implicit) states, variables and a buffer. Afterwards it can be shown that the implementation of the specification of the SPF will be a refinement. This can be shown by proving the properties 1 to 8 in the formal definition of the SPF.
Moreover an extension of the framework can be done by general lemmas given by the requirements of the case study regarding the automaton in figure 1. As this automaton has for each combination of state and input symbol one transition it is known that the automaton is deterministic and total. Therefore, it visualises exactly one stream processing function. [**?**]
Regarding the automaton one can see that there was next to input and output elements also a buffer b considered to check the equality of the letters in the next two time units. Out of simplicity reasons for the transition function, the states are to be considered in combination with the one element that is in the buffer in that moment for the implementation of the automaton. Therefore, the states will take the form $(state, element)$ and the automaton therefore has the following elements. Note that $x \in M$:

- the set of states $S = \{(Init, x), (F1, x), (F2, x)\}$,
- the set of input and output symbols $M = \{-, x\}$,
- the initial state $I = (Init, -)$ and

S_{old}	M	$Guard$	S_{new}	Out
$(Init, x)$	$-$	$-$	$(Init, x)$	$-$
$(Init, x)$	i	$i \neq -$	$(F1, i)$	$-$
$(F1, j)$	i	$i \neq -, i \neq j$	$(F1, i)$	$-$
$(F1, x)$	$-$	$-$	$(F2, x)$	$-$
$(F1, i)$	i	$-$	$(Init, i)$	i
$(F2, x)$	$-$	$-$	$(Init, x)$	$-$
$(F2, j)$	i	$i \neq -, i \neq j$	$(F1, i)$	$-$
$(F2, i)$	i	$-$	$(Init, i)$	i

Table 1: Table for the algebraic specification

- the transition function consisting in the table 1

As the SPF has not yet been defined formally, here is a formal definition:

$$\begin{gathered} f : M^\omega \to M^\omega \\ f(i) = - : h((Init, -), i) \text{ with} \\ h : (\{Init, F1, F2\} \times M) \to M^\omega \to M^\omega \end{gathered}$$

with

$$\begin{aligned} h(S, -) &= - \\ h((Init, x), - : rest) &= - : h((Init, x), rest) \\ h((Init, x), i : rest) &= - : h((F1, i), rest) \\ h((F1, j), i : rest) &= - : h((F1, i), rest) \\ h((F1, x), - : rest) &= - : h((F2, x), rest) \\ h((F1, i), i : rest) &= i : h((Init, i), rest) \\ h((F2, x), - : rest) &= - : h((Init, x), rest) \\ h((F2, j), i : rest) &= - : h((F1, i), rest) \\ h((F2, i), i : rest) &= i : h((Init, i), rest) \end{aligned}$$

In this chapter an automaton has been constructed using predefined equations and the corresponding SPF has been visualised. The interesting part about this automaton was that a buffer had to be considered to compare input messages with each other. It has been shown what differences may appear between the initial automaton and the implemented one, as the possibility of having a buffer storing messages is not always given.