Control Synthesis for a Smart Card Personalisation System using a Symbolic Model Checker

Biniam Gebremichael   Frits Vaandrager

biniam@cs.kun.nl   fvaan@cs.kun.nl

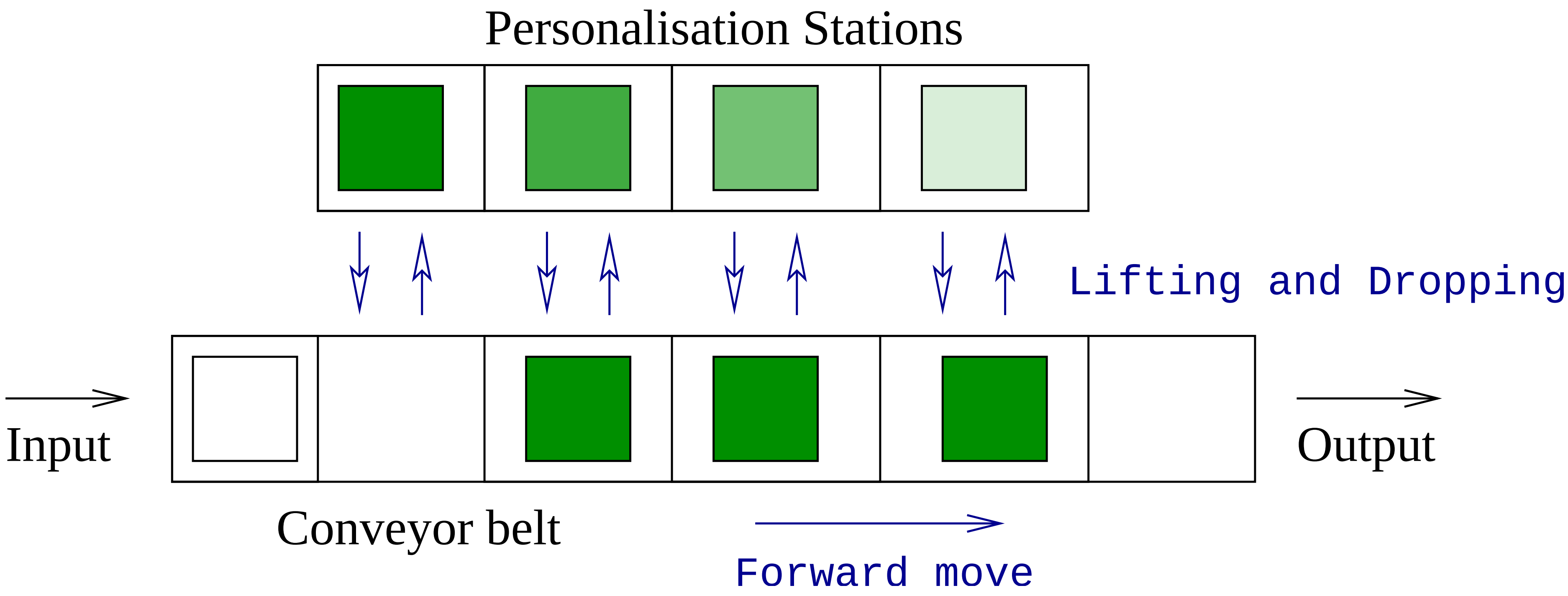
Computing Science Department
University of Nijmegen
The Netherlands

September 6-7 2003, Marseille, France

Content

1. Introducing smart card personalisation system
2. Super Single Mode
   ➜ Scheduling in the absence of error
3. Defective card treatment
   ➜ Scheduling in the presence of error – self-stabilizing
4. Conclusion and future work

Simplified Smart Card Personalisation System



Problem Definition

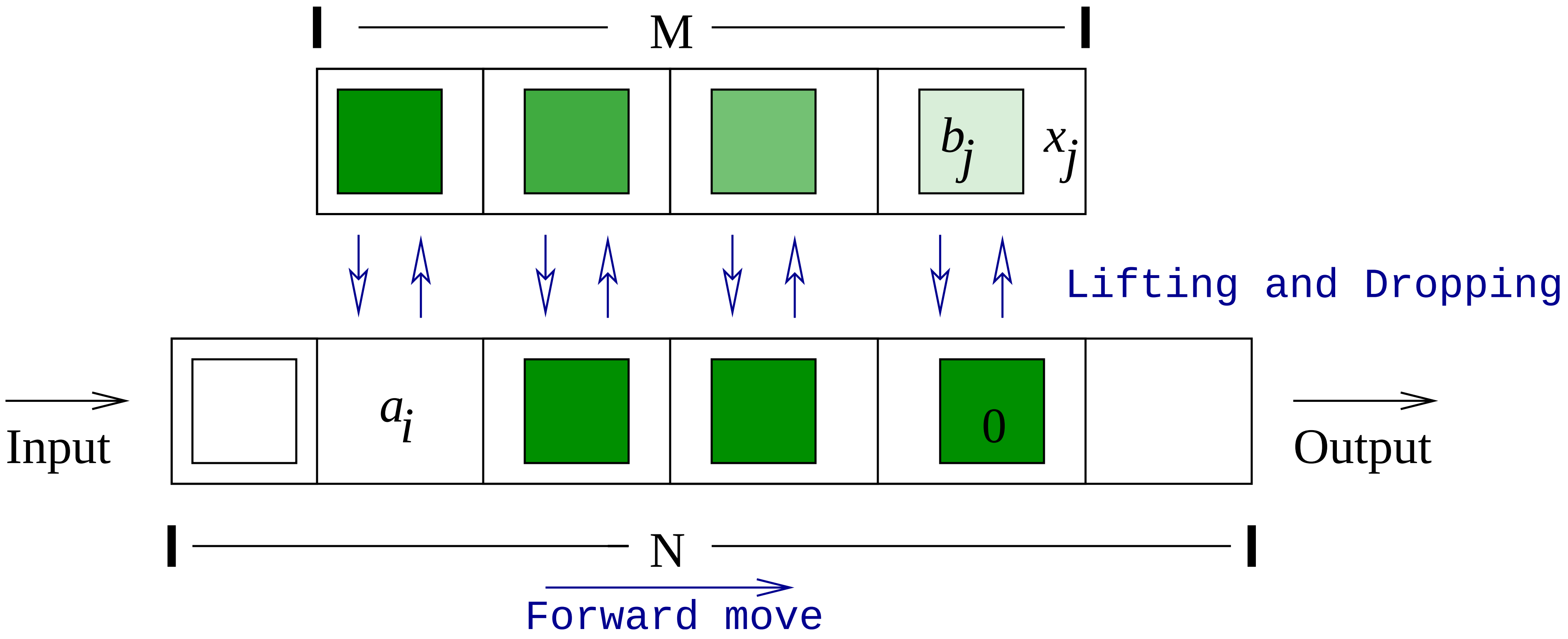
Given M personalisation stations can we find a scheduler/controller that can

1. produce personalised cards in the right order and with optimal throughput,
2. treats defective card automatically (self-stabilize)?

Super Single Mode – *a scheduler generated by SMV*

time	in put	personalisation stations 0	1	2	3	out put
1	□					
		0				
2	□					
		0				
3	□	□				
		0	1			
4	□	□				
		0	1			
5		□	□			
			1	2		
6	□	0	□			
		4	1	2		
7	□		0	□		
		4		2	3	
8	□	□	1	0		
		4	5	2	3	
9	□	□		1	0	
		4	5		3	
10		□	□	2	1	0
			5	6	3	
11	□	4	□		2	1
		8	5	6		
12	□		4	□	3	2
		8		6	7	
13	□	□	5	4		3
		8	9	6	7	
14	□	□		5	4	
		8	9		7	
15		□	□	6	5	4

Modelling



Parameters

M: Number of personalisation stations.
denoted as b_j: $(0 \leq j < M)$.

N = M+2: Number of slots in the conveyor belt.
denoted as a_j: $(0 \leq j < N)$.

K: The number of different personalisations.

S: The number of time units spent by a station to personalise a card. ($S = M$, in super single mode)
x_j: Clock for the j^{th} p.station $(0 \leq j < M)$. Ranges from *0* to S. One time unit is equal to one step move of the belt.

Parameter values

slot (a_j) and station (b_j) values

- INI: initial value
- MTY: empty
- NEW: new card
- j: personalised with j, $(0 \leq j < K)$
- K: defective card

Uncontrolled plant – Cadence SMV code

```
module main(){

/* initialize */
...

/* As minimum processes as possible - reduces state blow up
 * input, move belt, clock tick, output */
tk: process forward(a,b,x,out,tl);

/* lift, drop card */
for(j=0;j<=M-1;j=j+1)
   ld[j]:process lift_drop(a,b,x,j);
}
```

forward

```
module forward(a,b,x){

  /* Input */
  next(a[0]):={NEW,MTY};

  /* move belt forward */
  for(j=1;j<=N-1;j=j+1)     next(a[j]):=a[j-1];

  /* tick */
  for(j=0;j<=M-1;j=j+1){
     if(x[j]<S & b[j]>=0)   next(x[j]):= x[j]+1;
  }
}
```

lift and drop

```
module lift_drop(a,b,x,j){

  /* lift */
  if(b[j] <= MTY & a[j+1] = NEW){
    next(b[j]):= 0..K;
    next(a[j+1]):=b[j];
    next(x[j]):=0;
  }

  /* drop */
  else if(b[j] >= 0 & a[j+1] = MTY & x[j] = S){
    next(b[j]):= MTY;
    next(a[j+1]):=b[j];
  }
}
```

Observer

- correctness: produce cards in the right order

  If a card is produced non-sequentially, set `out` *to* `K`*,*
  otherwise increment `out`*.*

```
if(out = a[N-1])        next(out):= (out+1) mod K;
  else if(a[N-1]>MTY)  next(out):= K; /* error */
```

- optimization: minimize the number of blank slots

  If `S` *cards are produced, increment* `tl` *by one. If no card*
  is produce decrement `tl`*.*

```
if(a[N-1]=MTY)                                  next(tl):=tl-1;
else if(a[N-1]>=0 & (a[N-1] mod S)=S-1) next(tl):=tl+1;
```

- CTL formula: There is no correct and optimal run.
  $\texttt{AF ~(out} < \texttt{K} \wedge \texttt{tl} \geq \texttt{0)}$.

Observer

- correctness: produce cards in the right order

  If a card is produced non-sequentially, set out to K, otherwise increment out.

```
if(out = a[N-1])        next(out):= (out+1) mod K;
  else if(a[N-1]>MTY)   next(out):= K; /* error */
```

- optimization: minimize the number of blank slots

  If S cards are produced, increment tl by one. If no card is produce decrement tl.

```
if(a[N-1]=MTY)                                next(tl):=tl-1;
else if(a[N-1]>=0 & (a[N-1] mod S)=S-1)  next(tl):=tl+1;
```

- CTL formula: There is no correct and optimal run.

  `AF ~(out < K` $\wedge$ `tl` $\geq$ `0)`.

  SMV – *There is, super single mode*

Observation – super single mode is optimal

- In the super single mode – $S = M$ (a machine with M p.stations produce M cards prior to a blank slot.)
  Why?

$S < M$: slow belt movement, stations under utilized.
not optimal.

$S > M$: fast belt movement, needs extra blank positions.
not optimal.

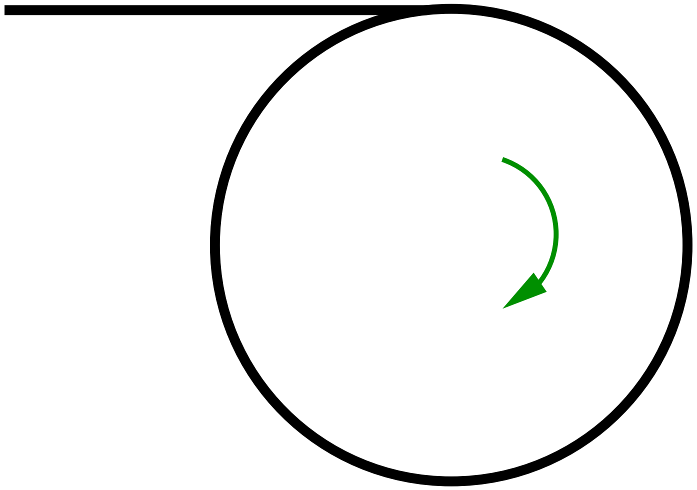
- It is not possible to produce $M+1$ consecutive cards.
- Thus, the super single mode is optimal in the absence of error.

II

Defective Card Treatment

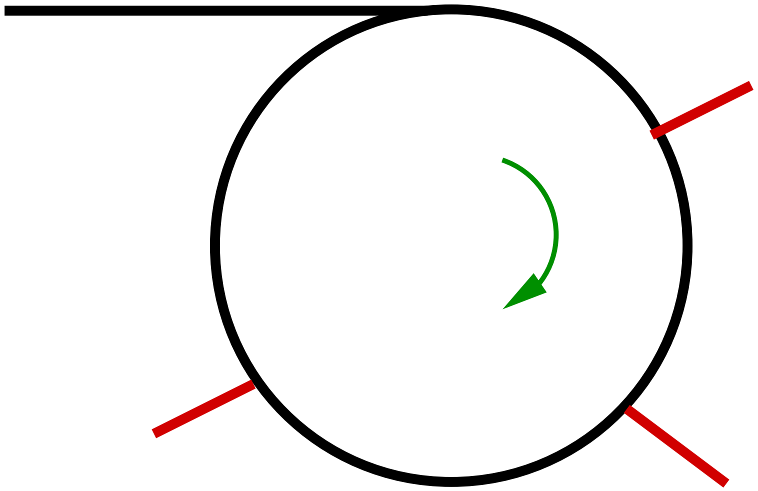
Defective card treatment

The super single mode loop



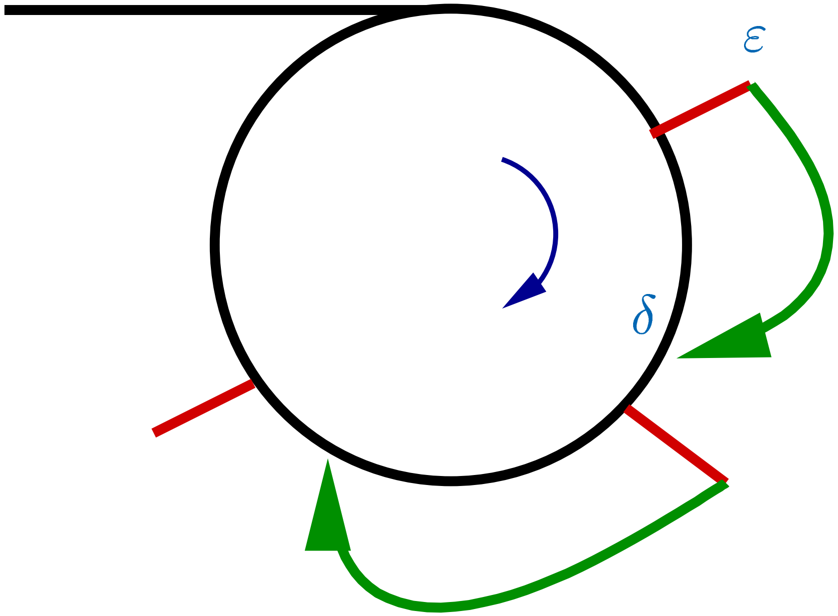
Defective card treatment

Occurrence of defective card.



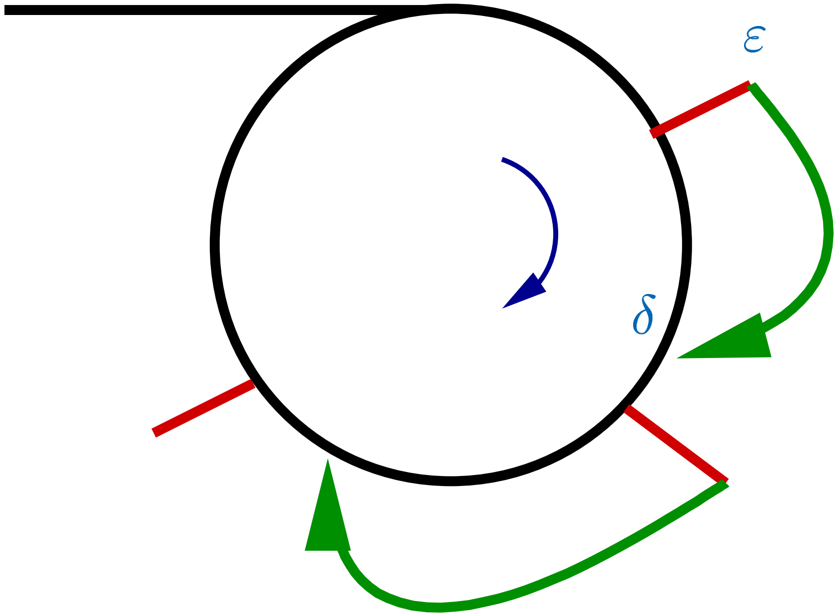
Defective card treatment

Stabilizing back to the loop.



Defective card treatment

Stabilizing back to the loop.



$\texttt{AG} \ (\ \varepsilon \ \rightarrow \ \texttt{EF} \ \delta)$

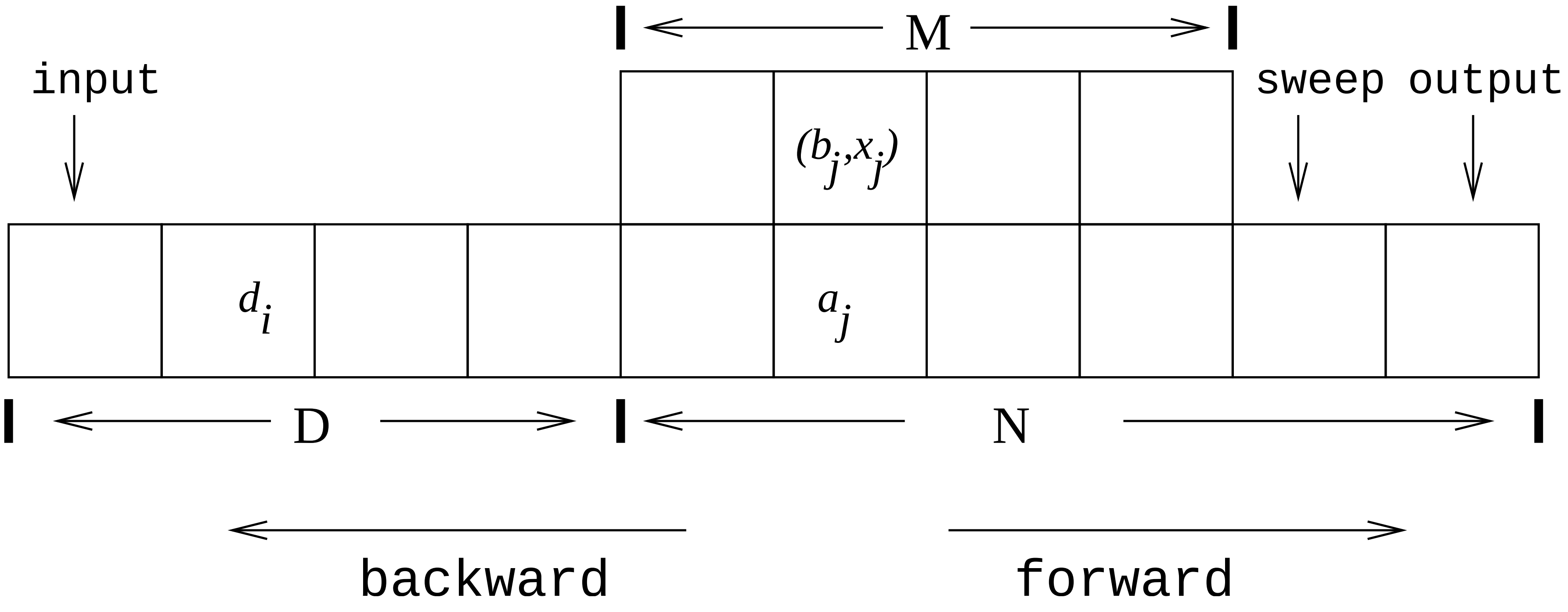
> *It is always possible to find a path from an error state ε to a state in the super single mode δ.*

Types of errors – restricting ε

- ➜ Error Type I (E1): Detected by the personalisation station. (physical damage)
- ➜ Error Type II (E2): Detected by the tester station. (wrong personalisation)

time	input	personalisation stations 0	1	2	3	4	5	6	7	output (tester)
4	□		1							
15	□			10						
16	□								7	
17	□				E1					
18										0
19	□					12?				E2
20	□	16?								2

Expanded personalisation machine



backward

```
module backward(d,a,b,x){

  if(d[0]<0){
    next(a[N-1]):= MTY;
    next(d[D-1]):= a[0];

    for(j=N-2;j>=0;j=j-1)                next(a[j]):= a[j+1];
    for(j=D-2;j>=0;j=j-1)                next(d[j]):= d[j+1];

    for(j=0;j<=M-1;j=j+1)
      if(x[j]<S & b[j]> -1)               next(x[j]):= x[j]+1;
  }
}
```

sweep

```
module sweep(a){
  if(a[M] = K)
     next(a[M]) := MTY;
}
```

Self Stabilization Property

It is always possible to find a path from an error state ε to a state in the super single mode δ.

$\mathtt{AG}\ (\ \varepsilon \rightarrow \mathtt{EF}\ \delta),$

where

- ε : a state that has E1 or E2 error.
- δ : a state (or a CNF of all states) in super single mode.

Self Stabilization Property

It is always possible to find a path from an error state ε to a state in the super single mode δ.

$\texttt{AG (} \varepsilon \rightarrow \texttt{ EF } \delta\texttt{)}$,

where

- ε : a state that has E1 or E2 error.
- δ : a state (or a CNF of all states) in super single mode.

Script generated CTL formula for all possible errors, from $\forall\delta$

```
AG (    ( a[0] = NEW & a[1] = MTY & b[2] = 4 /* = K */ & ...
          x[1] = 2 & x[2] = 0 & ... & d[0] = NEW & ...)
  -> EF ( a[0] = NEW & a[1] = MTY & b[2] = 3 & ...
          x[1] = 1 & x[2] = 0  & d[0] = NEW & ...)
   )
```

Example: Error treatment scheduler/controller

time	input	personalisation stations								tester
		0	1	2	3	4	5	6	7	
12	□						5			
21	□						13			3
22	□		17							4
23	□							14		5
24	□			6						E2
25	□	□		10	9	8	□	7		
26	□		10	9	8	□	7			
32	8	□	7	6						
38	□									6

Defective card treatments

Error type	First Half (FH) b_i: $0 \le i \le \lfloor \frac{M}{2} \rfloor$	Second Half (SH) b_i: $\lfloor \frac{M}{2} \rfloor < i \le M - 1$
E1	- Produce in the next station.	- Ignore. - Evolves to E2.
E2	- Backward to a ready station in the FH. - Reproduce the missing. - Forward.	- Forward one step. - Backward to a ready station in the FH. - Reproduce the missing. - Forward.

Example 2

Treating E1 in the first half of the stations

time	input	personalisation stations								output
		0	1	2	3	4	5	6	7	
11	□	8								
12	□						5			
13	□		E1							
14	□							6		
15	□			9						

Example 3

time	input	personalisation stations								tester
		0	1	2	3	4	5	6	7	
12	□						5			
21	□						13			3
22	□		17							4
23	□							14		E2
24	□			5						6
25	□	□		10	9	8	□	7	6	
26	□		10	9	8	□	7	6		
32	□	7	6	5						
32	□							14		5

Conclusion

- The super single mode is optimal scheduler for error free personalisation system.
- The controller can stabilize the machine automatically.
- Model checking can be useful to solve problems in the area of scheduling and control synthesis.

Future work

- Multiple error treatment.
- Other types of errors.
- Modeling the complete smart card personalisation system including printers and other stations.