Model Checking with SMV

Biniam Gebremichael

`biniam@cs.ru.nl`

ITT – Computing Science Department

University of Nijmegen

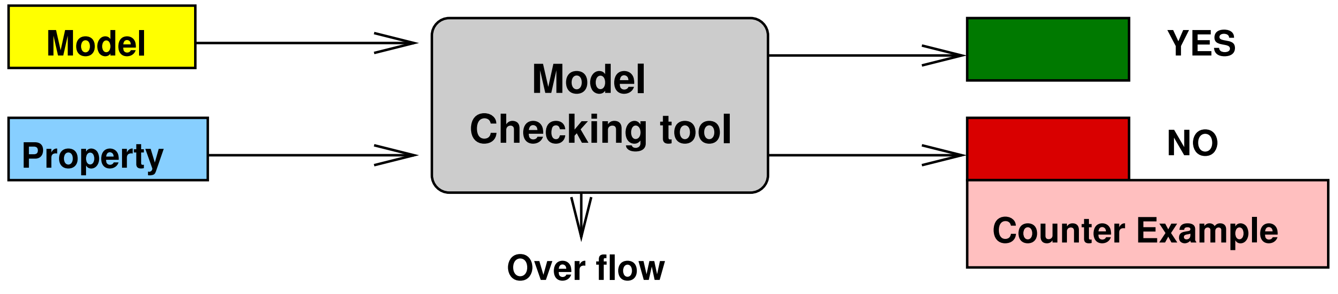
The Netherlands

March 2006

http://www.cs.ru.nl/~biniam/smv

Content

Model Checking



Model checking = An automatic technique for verifying properties of a finite model of a system.

General approach:

➜ Construct M= a model of the behavior of the system (given as kripke structure, finite automata, . . .). M must be finite.

➜ Specify ϕ = a property expected of the system (given as Temporal Logic)

➜ Check that M satisfies ϕ, if not , produce counter-example.

Examples of model checking tools: SMV, SPIN, Uppaal, Kronos . . .

Advantages and disadvantages of model checking (+/-):

+ **Completeness** : verification is fully exhaustive.

  Compare with simulation.

+ **Usability** :
  - ➜ Completely automatic,
  - ➜ can produce *counter-examples* that represent subtle errors or interesting execution paths.

  Compare with Theorem Proving

- **Applicability** : State explosion problem
  - ➜ The model should be finite and not too big.
  - ➜ Techniques to alleviate this problem: BDD data structure, partial reduction, symmetry, ...

Notable Examples

- Cache coherence protocol in the IEEE Futurebus standard in SMV [Clarke 1992]
- Cache coherence protocol of the IEEE Scalable Coherent Interface in Murφ [Dill 1992]
- High-level Data Link Controller (HDLC) in FormalCheck [$AT\&T$ 1996]
- Control Protocol used in Philips stereo components in Kronos [Bengtsson 1996]
- CCITT ISDN User Part Protocol [$AT\&T$ 92]
- Active structural control system, to make buildings more resistant to earthquakes [1995]

SMV: Symbolic Model Verifier

Ken McMillan, Symbolic Model Checking: An Approach to State Explosion Problem, 1993

- Modeling Language
  - ➜ Modularized and hierarchical descriptions
  - ➜ Finite data types: boolean, enum, int . . . etc
  - ➜ Array, loops, if-close . . . etc
  - ➜ Non-determinism, parallel execution
- Property specification Language
  - ➜ CTL and LTL
  - ➜ safety, lifeness, deadlock
  - ➜ Fairness
- Cadence SMV: command line and GUI for Windows / Linux / Sun-OS
- Other SMV versions: CMU-SMV, NuSMV

Modeling the SMV way

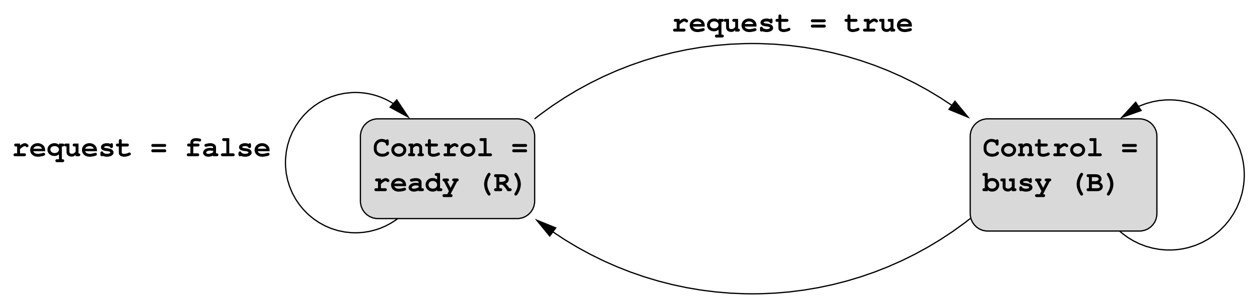


Figure 1: Simple SSH server

➜ **state variables:** control, request

4 states { (R,T), (R,F), (B,T), (B,F) }

➜ **state change:**

- ➜ initial value: eg. { (R,T), (R,F) }
- ➜ next value: when does a state variable change...

  eg.{ (R,T) –> (B,_), . . . }

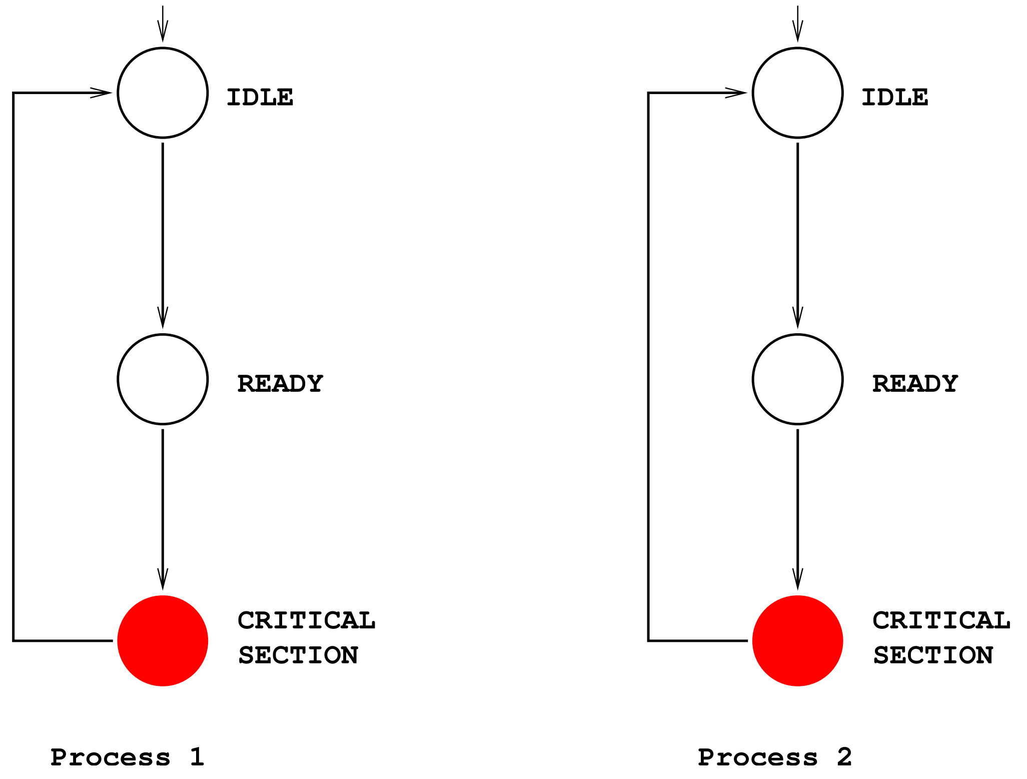
➜ **desired properties:** in CTL or LTL

eg. whenever a request is made it will be answered eventually.

Simple SSH server

```
module main() {
 -- state variables and their possible values
  request: boolean;
  control: {ready, busy};
 -- initial value
  init(control):= ready;
 -- next value
  if(control = ready & request)   next(control) := busy;
  else  next(control) := {ready, busy};     -- non deterministic choice
 -- request is not initialized and no next value is given
 -- it means request can be T or F (it is not upto the system to control it)

  -- whenever (AG) a request arrives it will be processed eventually (AF)
  -- eventually, some time in the future, (AF)
  p1: SPEC AG (request -> AF (control=busy));
}
```

Mutual Exclusion



➜ Both processes should not be in their critical section at the same time

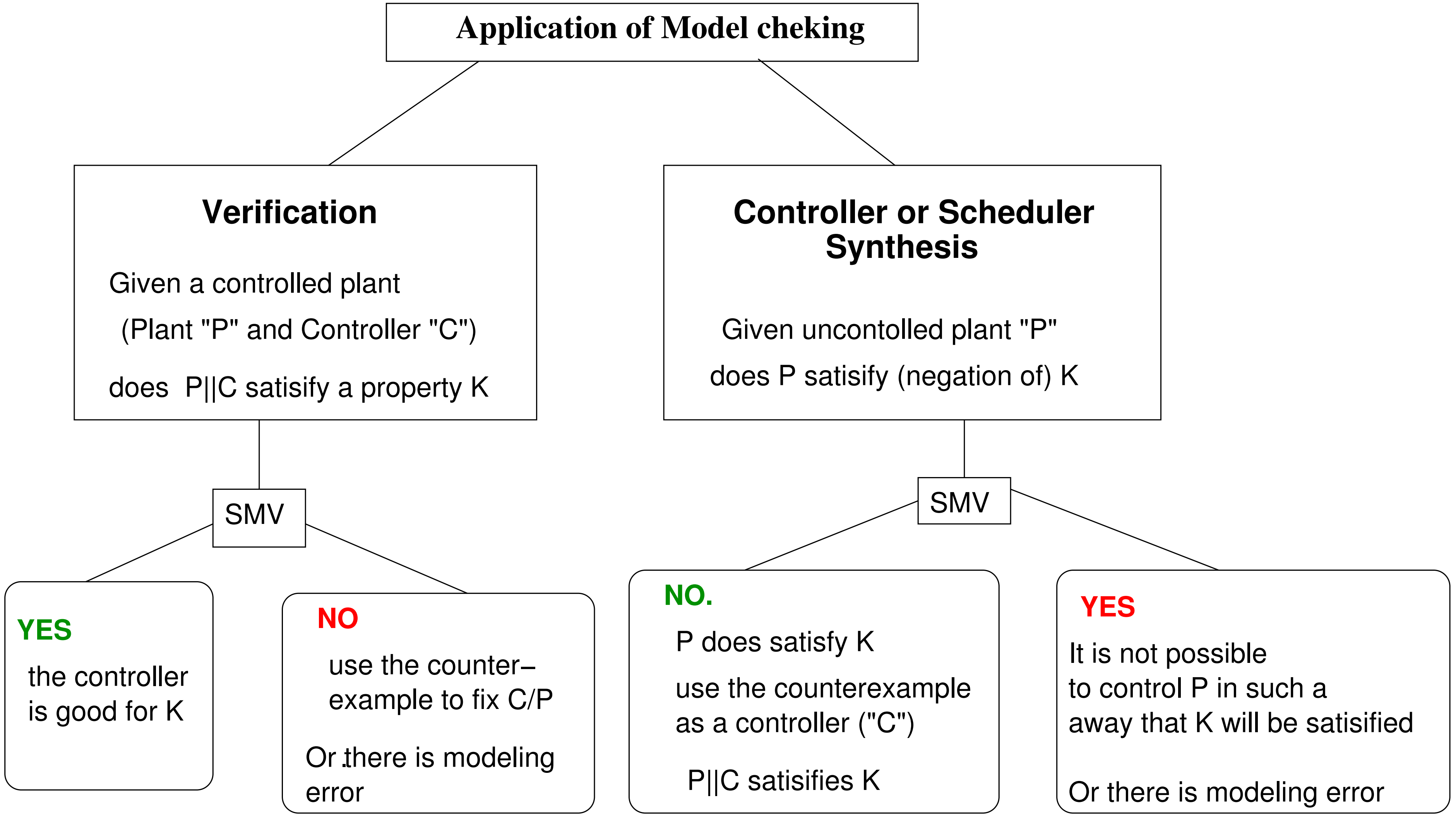
➜ A process must be allowed to enter its critical section

Mutual Exclusion (the controller)

```
MODULE main(){

turn : boolean;
pr0 : process prc(turn, 0);
pr1 : process prc(turn, 1);

init(turn) := 0;
next(turn):=   case {
(pr1.control = critical || pr0.control = critical )    : turn;
default                                                : {0,1};
};
   --safety
  SPEC AG ~((pr0.control = critical) & (pr1.control = critical));
   --liveness
  SPEC AG((pr1.control = ready) -> AF (pr1.control = critical));
}
```

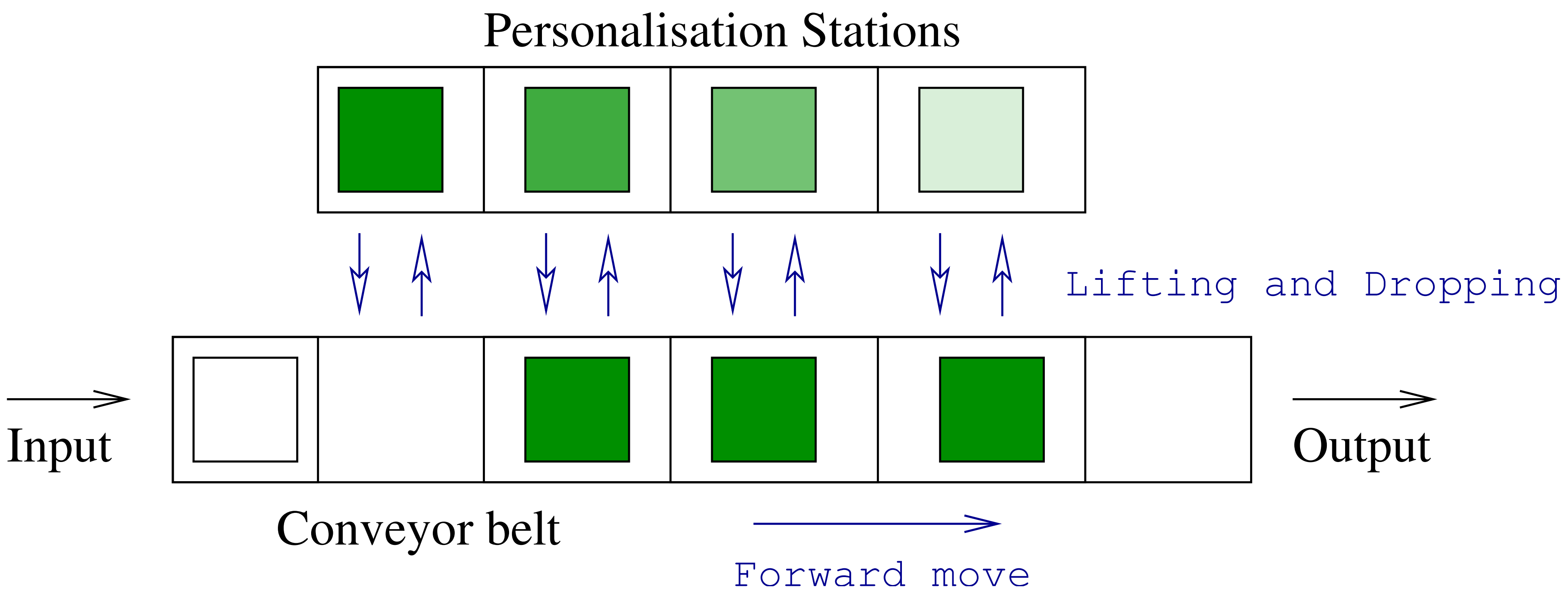
Mutual Exclusion (the processes)

```
MODULE prc(turn, myID){
control : {idle, ready, critical};
init(control) := idle;

next(control) :=  case {
            (control = idle)                           : {ready, idle};
            (control = ready) & (turn = myID)          : critical;
            (control = critical)                       : {critical, idle};
            default                                    : control;
    };

FAIRNESS running;               -- allow this process to run infinitely often
FAIRNESS ~(control = critical);  -- do not stay in critical section forever

}
```


Application of Model cheking
Verification
Given a controlled plant
(Plant "P" and Controller "C")
does P||C satisify a property K
Controller or Scheduler Synthesis
Given uncontolled plant "P"
does P satisify (negation of) K
SMV
SMV
YES
the controller is good for K
NO
use the counter–example to fix C/P
Or there is modeling error
NO.
P does satisfy K
use the counterexample as a controller ("C")
P||C satisifies K
YES
It is not possible to control P in such a away that K will be satisified
Or there is modeling error

Case Study

Smart Card Personalisation System – Cybernetics, France



Personalizing a card

1. New card inserted through INPUT
2. belt moves forward one step from left to right
3. card is lifted to any idle station
4. card spend at least S time units in the station to be personalised (with arbitrary value). Belt moves during this time as well.
5. personalized card is dropped to a slot beneath the station. This slot should be empty
6. The personalized card reaches the output position.

Problem Definition

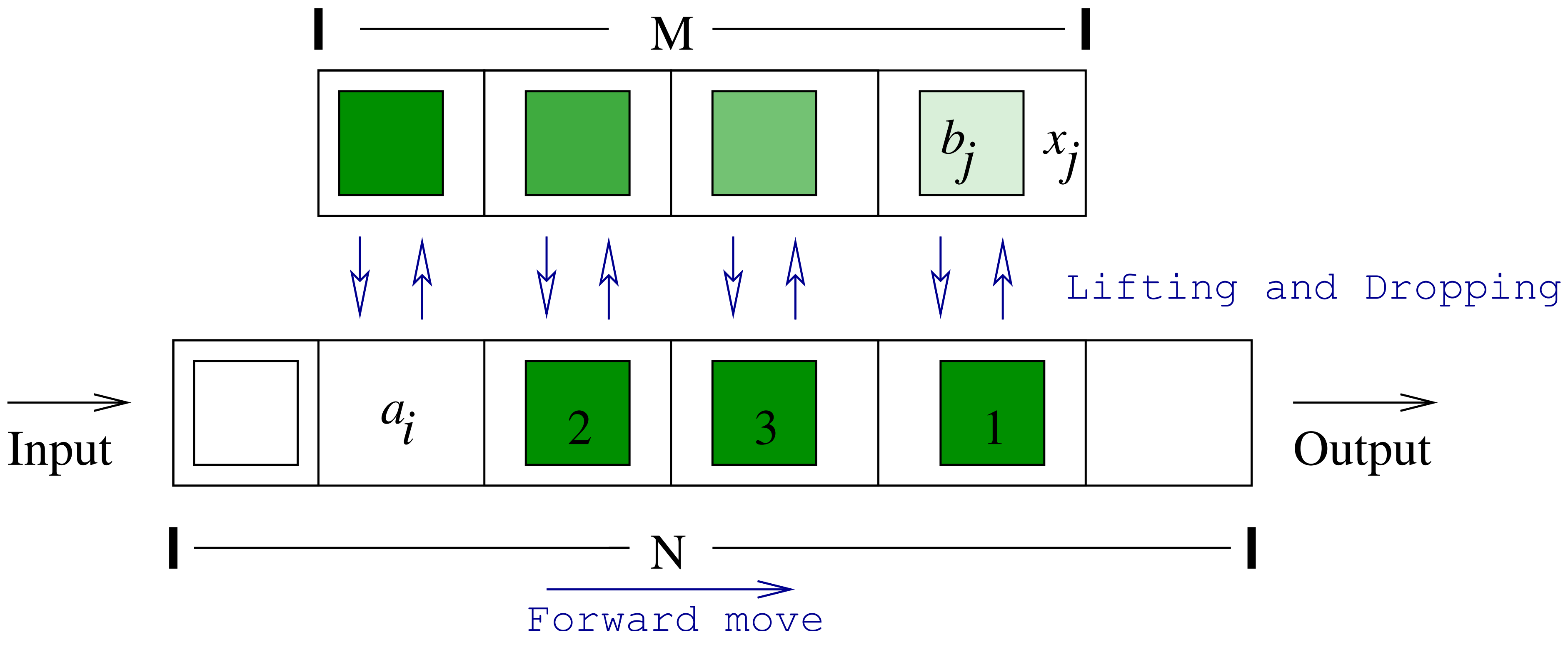
Given M personalisation stations find a scheduler/controller that produce personalised cards

1. in the right order and
2. with optimal throughput

Trivial solution: one mode approach

Modeling

- ➜ Model the machine as an uncontrolled plant. (any thing that is physically allowed should also be allowed by the model).
- ➜ Model the negation of the properties that need to be satisfied (or called the observer)



state variables

Plant parameters

$b[M]$:	personalisation stations.	-3 ... K
$x[M]$:	clocks for each of the p.stations.	0 ... S
$a[N]$:	slots in the conveyor belt.	-3 ... K

Property/Observer parameters

`out:`	the next expected value in OUTPUT	-2 ... K
`tl:`	blank space tolerance.	-1 ... 1

Where

- ➜ `M` is the number of personalization stations
- ➜ `N` is the number of slots in the conveyor belt. $N = M + 2$
- ➜ `K` is the number of different personalisations
- ➜ `S` is the number of time units needed to personalise a card.
  one time unit is defined as equal to one step move of the belt.

Processes

the number of non-determinism

s

1. lifting a card (for each personalisation station),
2. dropping a card (for each personalisation station),
3. moving belt forward,
4. inserting new card,
5. ticking the timer,
6. observing order of delivery and optimality

Processes

1. lifting a card (for each personalisation station),
2. dropping a card (for each personalisation station),
3. moving belt forward,
4. inserting new card,
5. ticking the timer,
6. observing order of delivery and optimality

Grouping: decreases the number of non-determinism

- ➜ forward: 3,4,5 and 6
- ➜ lift_drop: 1 and 2

SMV model – top level specification

```
module main(){
 a: array 0..N-1 of -3..K;    -- conveyor belt
 b: array 0..M-1 of -3..K;    -- personalisation station
 x: array 0..M-1 of -2..S;    -- tick counter (discrete clock)
 out: -2..K;                  -- correctness observer
 tl: -1..1;                   -- optimality observer

 tk: process forward(a,b,x,out,tl);   -- forward process
 for(j=0;j<=M-1;j=j+1)
   ld[j]:process lift_drop(a,b,x,j);  -- lift_drop processes

 for(j=0;j<=N-1;j=j+1)   init(a[j]):=-3;   -- initialization
 for(j=0;j<=M-1;j=j+1){  init(b[j]):=-3;
                         init(x[j]):=0; }
 init(out):=0;  init(tl):=0;
}
```

forward

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

  -- Input: new card or nothing
  next(a[0]):={NEW,MTY};  -- NEW = -3, MTY = -2

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

  -- advance the clock of all busy stations
  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 the station is idle and if there is new card beneath
  if(b[j] <= MTY & a[j+1] = NEW){
      next(b[j]):= 0..K;    -- choose arbitrary
                            -- personalisation value
      next(a[j+1]):=b[j];   -- empty the slot beneath
      next(x[j]):=0;        -- reset the timer
   }
    -- drop if the card is fully personalised and
    -- if the slot beneath is empty
  else if(b[j] >= 0 & a[j+1] = MTY & x[j] = S){
      next(b[j]):= MTY;     -- the station is emptied
      next(a[j+1]):=b[j];   -- the card is dropped to the slot
   }
}
```

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.

  $\mathtt{AF}\ \tilde{}\,(\mathtt{out} < \mathtt{K} \wedge \mathtt{tl} \geq 0)$.

Super Single Mode – *a scheduler generated by SMV*

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

The Barbershop Problem

The barber shop has 3 barbers, 3 barber chairs, and a waiting area with a sofa, There is a limitation of m customers in the shop at a time. The barbers divide their time between cutting hair, accepting payment, and sleeping in their chair waiting for a customer. A customer will not enter the shop if it is filled to capacity. Once inside, the customer takes a seat on the sofa or stands if the sofa is full. When a barber is free, the customer that has been waiting the longest on the sofa is served and the customer that has been standing the longest takes its place on the sofa. When a haircut is finished any barber can accept payment, but payment can be accepted for only one customer at a time as there is only one cash register.

- ➜ List the state variables, their possible values and initial values
- ➜ what are actions that define the next value of the state variables
- ➜ give CTL/LTL formula for “A costumer admitted to the shop will be served eventually”