Safety and Reliability of Embedded Systems

(Sicherheit und Zuverlässigkeit eingebetteter Systeme)

Symbolic Model Checking

Symbolic Model Checking
Inhalt

- Einleitung
- Beispiel: Der Hubdrehtisch
- Temporale Logik (CTL)
- Zustandsraum
- Nachweis von Sicherheitsanforderungen
- Exkurs OBDDs
- Kodierung von Zustandsautomaten mit OBDDs
- Analyse von Zustandsautomaten mit OBDDs
- Zusammenfassung

Symbolic Model Checking
Einleitung

- Symbolic Model Checking ist eine formale Verifikationstechnik
  - Sie ist mathematisch fundiert
  - Sie erfordert formale Dokumente als Ausgangsbasis
  - Sie liefert vollständige Aussagen
  - Die Ergebnisse sind verlässlich

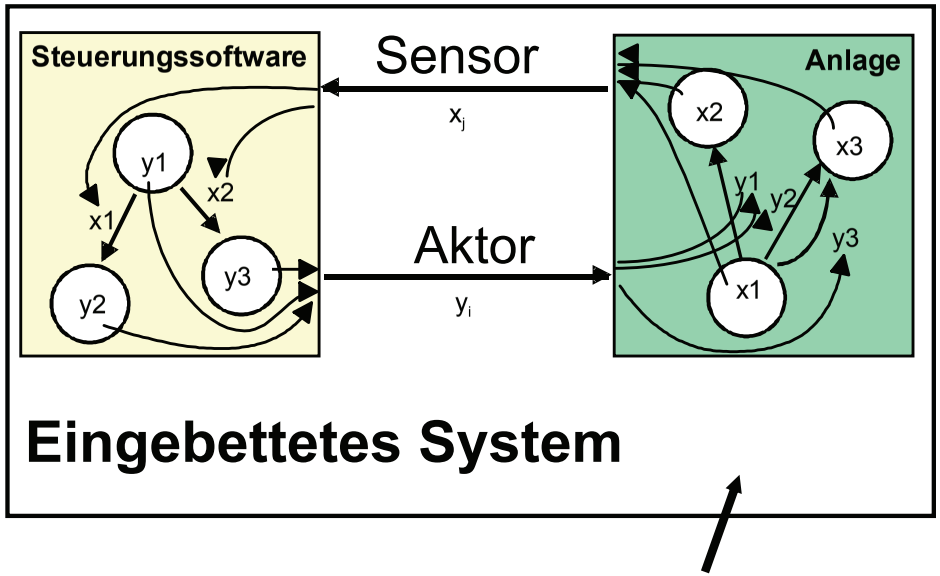
- Ziel:
  - Nachweis definierter Eigenschaften einer Betrachtungseinheit (Software, Spezifikation) oder ...
  - ... Erzeugung von Informationen dazu, wo die geforderte Eigenschaft nicht gilt

Symbolic Model Checking
Einleitung

- Symbolic Model Checking wird im folgenden anhand des Nachweises von Eigenschaften
  - ... zustandsendlicher Verhaltensbeschreibungen
  - ... gegen temporallogische Formeln verdeutlicht
- Zustandsendliche Beschreibungen sind interessant, weil sie in der Softwaretechnik aber auch in anderen Disziplinen (z.B. Elektrotechnik) weit verbreitet sind. Programmiersprachen für die Steuerungstechnik basieren in der Regel auf Zustandsautomaten.
- Temporallogik ist erforderlich, weil beim Traversieren von Zustandsautomaten Aussagen über die Reihenfolge erforderlich sind.

Symbolic Model Checking
Einleitung

- Zustandsendlich realisierte Steuerungssoftware
- Formale Beschreibung der gesteuerten Anlage
- Formale Spezifikation der (Sicherheits-) anforderung in temporaler Logik (z.B. CTL) für das eingebettete System bestehend aus Steuerungssoftware und der gesteuerten Anlage



Sicherheitsanforderung in Temporallogik

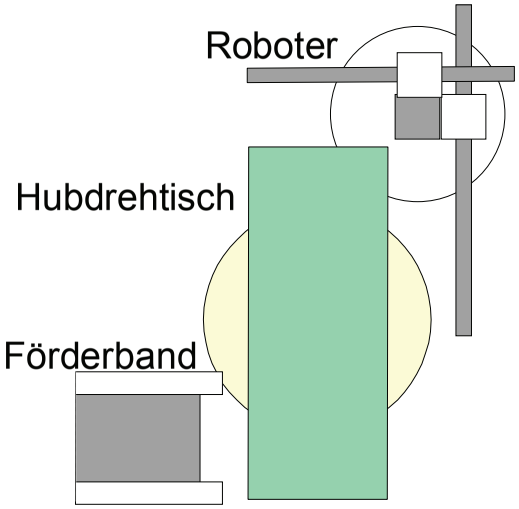
Symbolic Model Checking
Beispiel: Der Hubdrehtisch (Hardware)

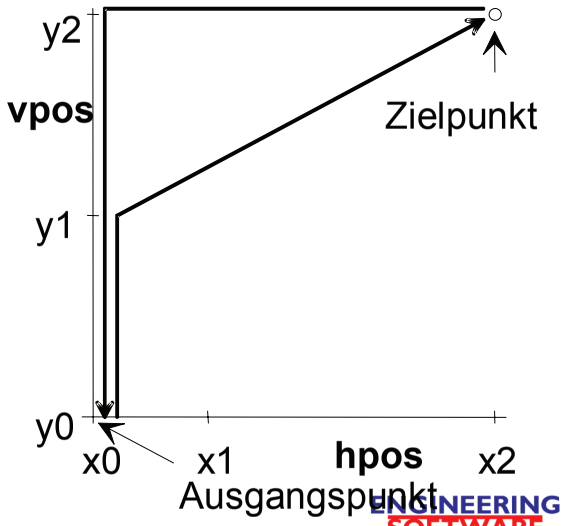
Eine Fertigungszelle verarbeitet Metall-Rohteile, die einer Presse durch einen Roboter zugeführt werden. Der Roboter greift das Rohteil von einem Hubdrehtisch, dem es durch ein Förderband zugeliefert wird. Die Aufgabe des Hubdrehtisches ist, durch eine horizontale und vertikale Positionsänderung ein Rohteil so zu platzieren, dass es vom Roboter gegriffen werden kann. Der Tisch besitzt zwei Antriebe (Aktuatoren) – einen für die horizontale und einen für die vertikale Richtung - und 3 Sensoren – jeweils einen für die horizontale und vertikale Position sowie einen für die Registrierung eines Werkstücks auf dem Tisch.

Aktuator	**Name**	**Werte**
Horizontale Bewegung	*hmov*	*stop, plus, minus*
Vertikale Bewegung	*vmov*	*stop, up, down*
Sensor	**Name**	**Werte**
Horizontale Position	*hpos*	x_0, x_1, x_2
Vertikale Position	*vpos*	y_0, y_1, y_2
Vorhandensein eines Werkstücks	*part_on_table*	*no, yes*

Symbolic Model Checking
Beispiel: Der Hubdrehtisch (Steuerungsstrategie: Software)

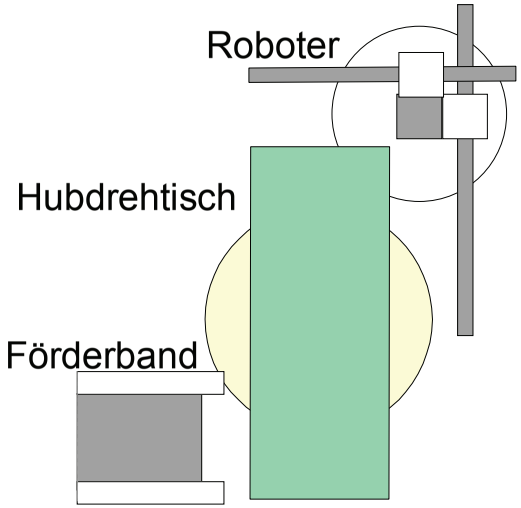
Die Ausgangsposition des Tisches ist hpos=x0, vpos=y0. Alle Bewegungen sind gestoppt (hmov=stop, vmov=stop). Die vertikale Bewegung wird gestartet (vmov=up) sobald sich ein Werkstück auf dem Tisch befindet (part_on_table=yes). Wenn vpos=y1 erreicht wird, so wird auch die horizontale Bewegung gestartet (hmov=plus). Wenn der Tisch die Zielposition (hpos=x2, vpos=y2) erreicht, so kann der Roboter das Werkstück greifen. Sobald das Werkstück den Tisch verlassen hat (part_on_table=no) beginnt die horizontale Bewegung (hmov=minus) bis die Position hpos=x0 erreicht ist. Die Abwärtsbewegung beginnt (vmov=down) und die horizontale Bewegung wird gestoppt (hmov=stop). Wenn der Tisch die Ausgangsposition erreicht, so wird auch die Abwärtsbewegung gestoppt (vmov=stop).

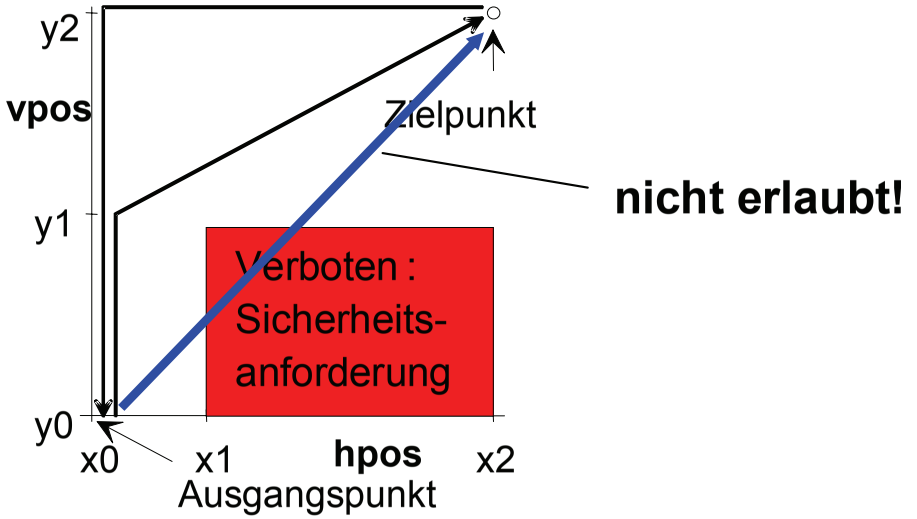




Symbolic Model Checking
Beispiel: Der Hubdrehtisch
Zu beweisende Eigenschaft

Es ist eine Sicherheitsanforderung zu beachten. Der Tisch darf sich nicht in den verbotenen Bereich bewegen. Dies ist sichergestellt, falls Zustände nicht erreichbar sind, für die [vpos=y0 /\ (hpos=x1 \/ hpos=x2)] erfüllt ist. Aus diesem Grund wird bei der Aufwärtsbewegung die horizontale Bewegungsrichtung erst an der vertikalen Position y1 gestartet und nicht direkt zu Beginn. Analog gilt dies auch für die Abwärtsbewegung.

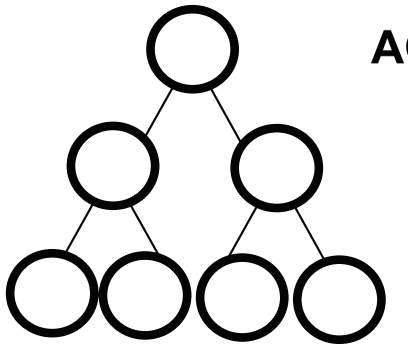




CTL (**C**omputation **T**ree **L**ogic):

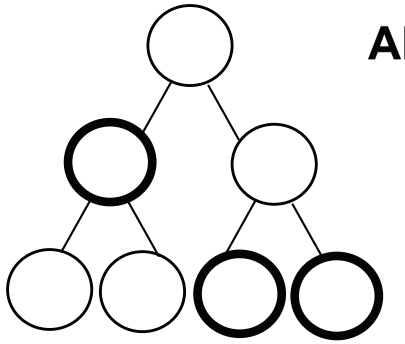
- Forward-time operators:
  - G: **g**lobally, invariantly
  - F: sometime in the **f**uture
  - X: ne**x**t time
  - U: **U**ntil
- Path quantifiers:
  - A: **a**ll computation paths
  - E: some computation path (**e**xists)
- In CTL muss direkt vor dem *forward-time operator* ein *path quantifier* stehen, also z.B. **AG** *f* oder **EF** *f*

Symbolic Model Checking
Exkurs: Temporale Logik; hier CTL



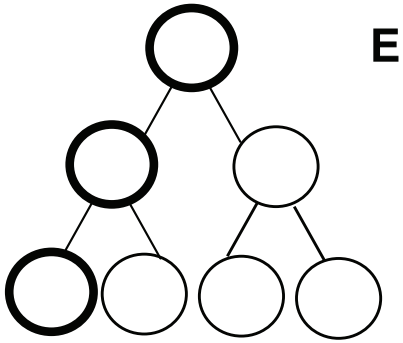
AG *f*

Alle Zustände auf allen Pfaden erfüllen *f*



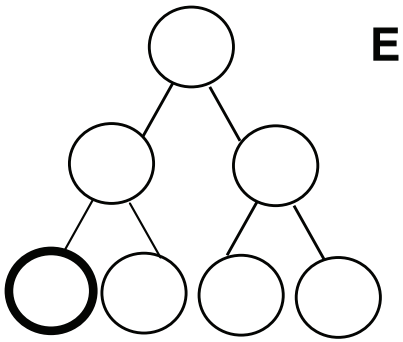
AF *f*

Auf allen Pfaden erfüllt mindestens ein Zustand *f*



EG *f*

Es gibt mindestens einen Pfad auf dem alle Zustände *f* erfüllen

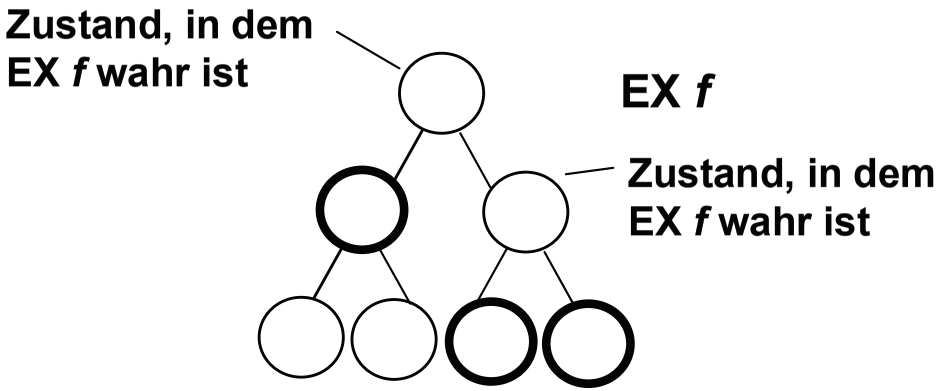


EF *f*

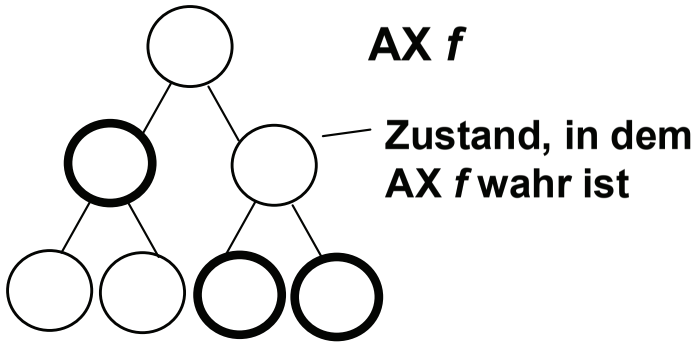
Es gibt mindestens einen Pfad auf dem Mindestens ein Zustand *f* erfüllt

Zustand, in dem *f* gilt

Symbolic Model Checking
Exkurs: Temporale Logik; hier CTL



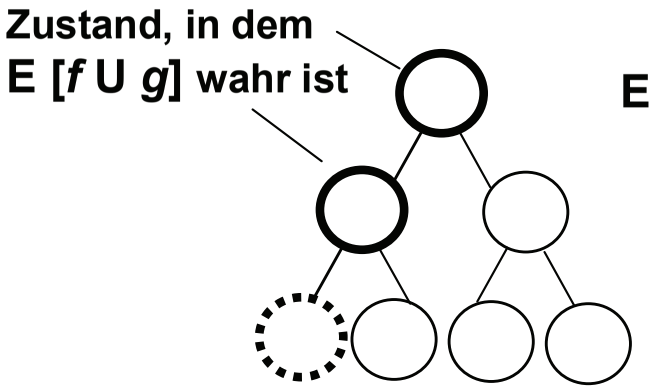
Es muss einen direkten Folgezustand geben,
In dem *f* gilt



f muss in allen direkten Folgezuständen gelten

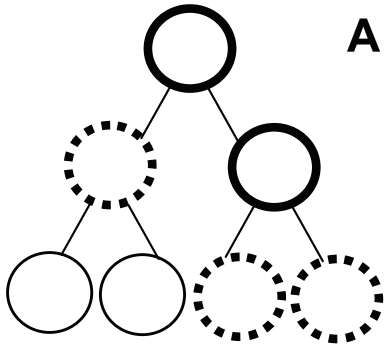
Zustand, in dem *f* gilt

Symbolic Model Checking
Exkurs: Temporale Logik; hier CTL



E [*f* U *g*]

f ist auf einem Pfad solange wahr bis *g* gilt.



A [*f* U *g*]

f ist entlang aller Pfade solange wahr bis *g* eintritt.

Zustand, in dem *f* gilt

Zustand, in dem *g* gilt

Symbolic Model Checking
Beispiel Hubdrehtisch: Sicherheitsanforderung

Die Sicherheitsanforderung ist:

Es darf niemals ein Zustand auftreten, für den gilt:

[vpos=y0 /\ (hpos=x1 \/ hpos=x2)]

In CTL:

AG ~[vpos=y0 /\ (hpos=x1 \/ hpos=x2)]

oder äquivalent:

~EF [vpos=y0 /\ (hpos=x1 \/ hpos=x2)]

~: Negation

Symbolic Model Checking
Ein Beispiel

- Steuerungssoftware in der zustandsmaschinenbasierten Programmiersprache CSL (Control Specification Language):

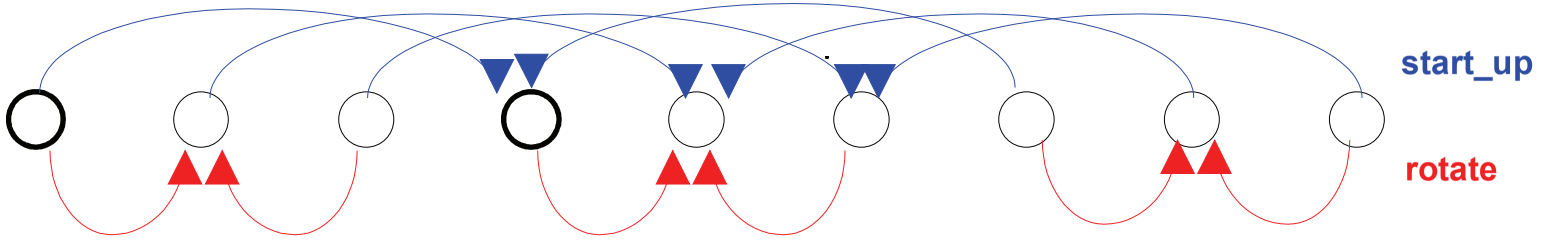
```
StateVariables
        input  vpos : [y0, y1, y2] default y0;
        input  hpos : [x0, x1, x2] default x0;
        input  part_on_table  : [no, yes]  default no;
        output vmov: [stop, up, down] default stop;
        output hmov: [stop, plus, minus] default stop;

Transitions
        start_up:= (part_on_table = yes /\ vpos = y0) ==> (** vmov = up);
        rotate:= (part_on_table = yes /\ vpos = y1 /\ hpos < x2) ==> (** hmov = plus);
        stophigh:= (part_on_table = yes /\ vpos = y2) ==> (** vmov = stop);
        stop_rot:= (part_on_table = yes /\ hpos = x2) ==> (** hmov = stop);
        rot_back:= (part_on_table = no /\ vpos = y2 /\ hpos = x2) ==> (** hmov = minus);
        start_down:= (part_on_table = no /\ hpos = x0 /\ vpos = y2)
                        ==> (** hmov = stop /\ ** vmov =    down);
        stoplow:= (part_on_table = no /\ vpos = y0) ==> (** vmov = stop);
```

Symbolic Model Checking
Zustandsraum des Beispiels

Der Zustandsraum der Steuerung ergibt sich aus der Kombination der Aktorwerte. Sensordaten lösen darin Zustandsübergänge aus.

In den Zustandsraum sind Transitionen *start_down* und *rotate* eingezeichnet.

vmov	stop	stop	stop	up	up	up	**down**	down	down
hmov	stop	plus	minus	stop	plus	minus	**stop**	plus	minus




```
start_up :=  (part_on_table = yes /\ vpos = y0) ==> (** vmov = up);
rotate:=  (part_on_table = yes /\ vpos = y1 /\ hpos < x2) ==> (** hmov = plus);
stophigh :=  (part_on_table = yes /\ vpos = y2)  ==> (** vmov = stop);
stop45 :=  (part_on_table = yes /\ hpos = x2)  ==> (** hmov = stop);
rot_back := (part_on_table = no /\ vpos = y2  /\ hpos = x2) ==> (** hmov = minus);
start_down :=  (part_on_table = no /\ hpos = x0 /\ vpos = y2) ==> (** hmov = stop /\ ** vmov = down);
stoplow :=  (part_on_table = no /\ vpos = y0) ==> (** vmov = stop);
```

Symbolic Model Checking
Beispiel Hubdrehtisch

- Die relevanten Eigenschaften sind für den Steuerungsautomaten allein nicht definiert und daher auch nicht nachweisbar.
- Es ist erforderlich, die Eigenschaften des gesteuerten Systems zu beachten, d.h. die Antworten des gesteuerten Systems auf Kommandos der Steuerungssoftware
- Diese ergeben sich aus meistens relativ simplen physikalischen Eigenschaften des gesteuerten Systems, z.B.
  - *Wenn ein Antrieb gestoppt ist, so bewegt sich das gesteuerte System in dieser Achse nicht.*
  - *Bei geöffnetem Einström- und geschlossenem Ausströmventil nimmt der Füllstand eines geschlossenen Kessels nicht ab.*
- Die Kombination aus Steuerungslogik und den Antworten des gesteuerten Systems ergibt das Verhalten der Gesamtanlage
  => formale Beschreibung des Verhaltens des gesteuerten Systems erforderlich

Symbolic Model Checking
Beispiel Hubdrehtisch

allMotorsStop:= ('table.hmov' = stop /\ 'table.vmov' = stop).
initialPosition:= ('table.hpos'=x0) /\ ('table.vpos'=y0).
finalPosition:= ('table.hpos'=x2) /\ ('table.vpos'=y2).
initialState:= initialPosition /\ ('table.part_on_table'=no).
finalState:= finalPosition /\ ('table.part_on_table'=yes).

Definitionen

fairprocess:=
(
(***In Abhängigkeit der Bewegungsrichtung müssen vertikale Positionen in einer bestimmten Reihenfolge eintreten***
('table.vmov' = stop /\ 'table.vpos' = y0) /\ x('table.vpos' = y0)
\/ ('table.vmov' = stop /\ 'table.vpos' = y1) /\ x('table.vpos' = y1)
\/ ('table.vmov' = stop /\ 'table.vpos' = y2) /\ x('table.vpos' = y2)

\/ ('table.vmov' = up /\ 'table.vpos' = y0) /\ until('table.vpos' = y0, 'table.vpos' = y1)
\/ ('table.vmov' = up /\ 'table.vpos' = y1) /\ until('table.vpos' = y1, 'table.vpos' = y2)
\/ ('table.vmov' = up /\ 'table.vpos' = y2) /\ x('table.vpos' = y2)

\/ ('table.vmov' = down /\ 'table.vpos' = y0) /\ x('table.vpos' = y0)
\/ ('table.vmov' = down /\ 'table.vpos' = y1) /\ until('table.vpos' = y1, 'table.vpos' = y0)
\/ ('table.vmov' = down /\ 'table.vpos' = y2) /\ until('table.vpos' = y2, 'table.vpos' = y1)
)
...

Symbolic Model Checking
Beispiel Hubdrehtisch

/\ (*In Abhängigkeit der Bewegungsrichtung müssen horizontale Positionen in einer bestimmten Reihenfolge eintreten*
('table.hmov' = stop /\ 'table.hpos' = x0) /\ x('table.hpos' = x0)
\/ ('table.hmov' = stop /\ 'table.hpos' = x1) /\ x('table.hpos' = x1)
\/ ('table.hmov' = stop /\ 'table.hpos' = x2) /\ x('table.hpos' = x2)

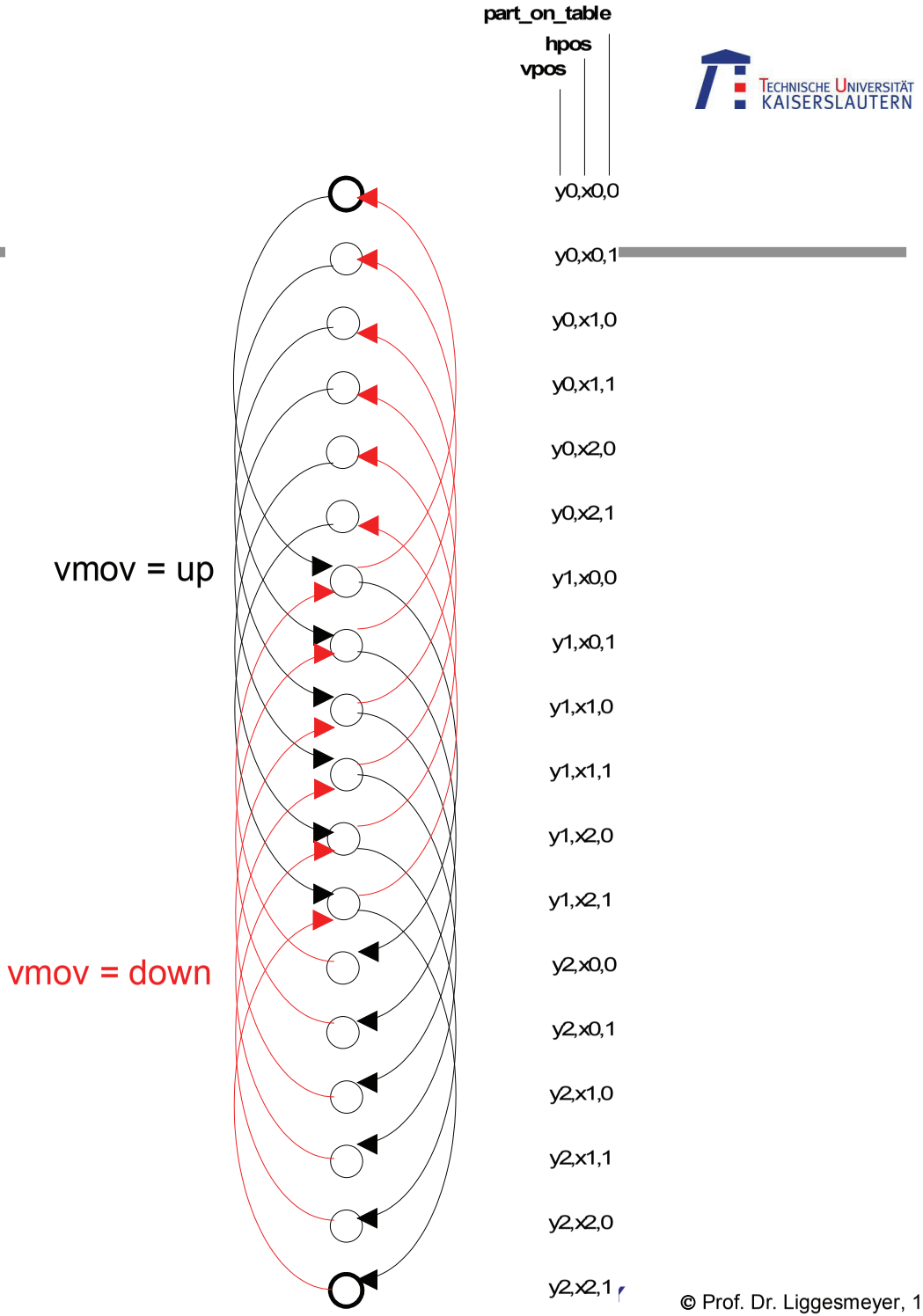
\/ ('table.hmov' = plus /\ 'table.hpos' = x0) /\ until('table.hpos' = x0, 'table.hpos' = x1)
\/ ('table.hmov' = plus /\ 'table.hpos' = x1) /\ until('table.hpos' = x1, 'table.hpos' = x2)
\/ ('table.hmov' = plus /\ 'table.hpos' = x2) /\ x('table.hpos' = x2)

\/ ('table.hmov' = minus /\ 'table.hpos' = x0) /\ x('table.hpos' = x0)
\/ ('table.hmov' = minus /\ 'table.hpos' = x1) /\ until('table.hpos' = x1, 'table.hpos' = x0)
\/ ('table.hmov' = minus /\ 'table.hpos' = x2) /\ until('table.hpos' = x2, 'table.hpos' = x1)
)
/\ (
(initialState /\ allMotorsStop /\ x('table.part_on_table' = yes)) *Werkstücke erscheinen im Startzustand*
\/ (finalState /\ allMotorsStop /\ x('table.part_on_table' = no)) *Werkstücke verschwinden im Endzustand*
\/ (*In allen anderen Zuständen ändert sich an dem Vorhandensein von Werkstücken nichts*
~(initialState /\ allMotorsStop \/ finalState /\ allMotorsStop)
/\ ('table.part_on_table' = no /\ x('table.part_on_table' = no) \/ 'table.part_on_table' = yes /\
x('table.part_on_table' = yes)
)
)
)
)

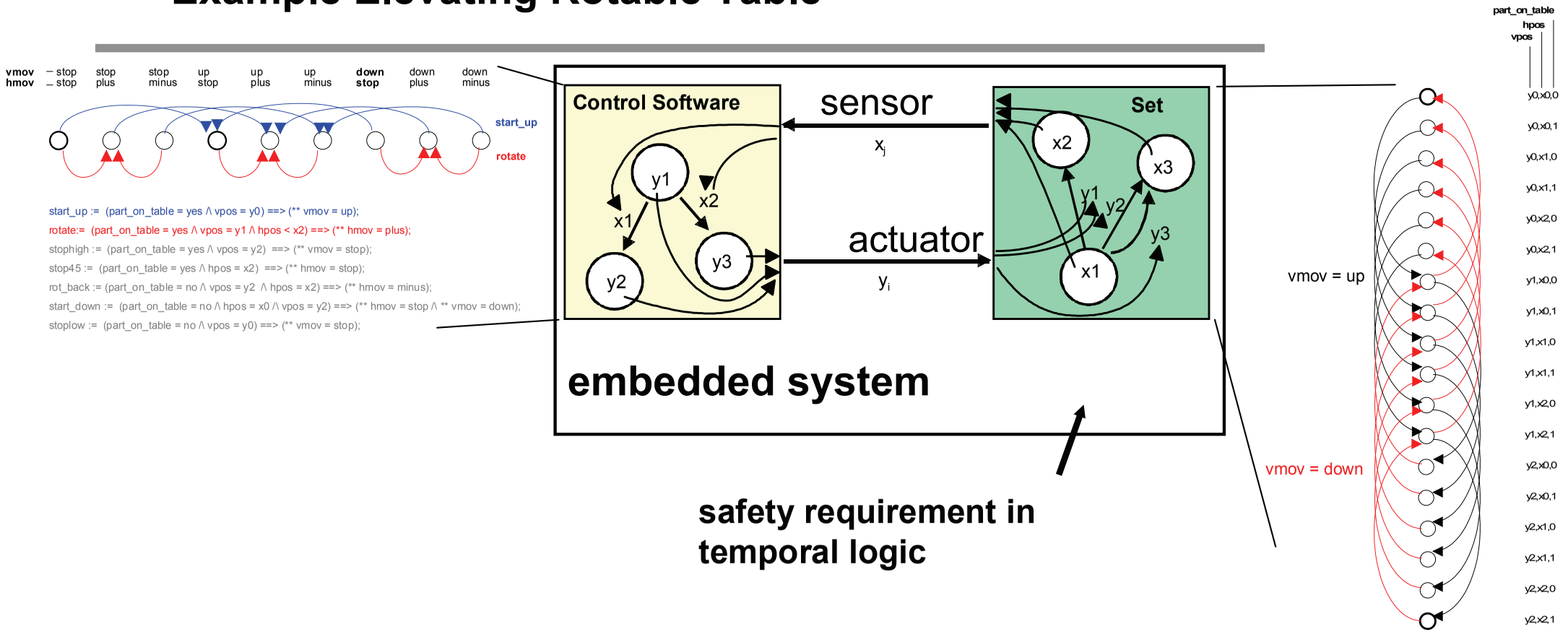
Symbolic Model Checking
State space of the example

The state space of the elevating rotable table is defined as the combination of the sensor values. Aktuator data is triggering transitions in this state space.

The diagram shows those transistions, that represent the correlation between vertical movement and vertical position



Symbolic Model Checking
Example Elevating Rotable Table

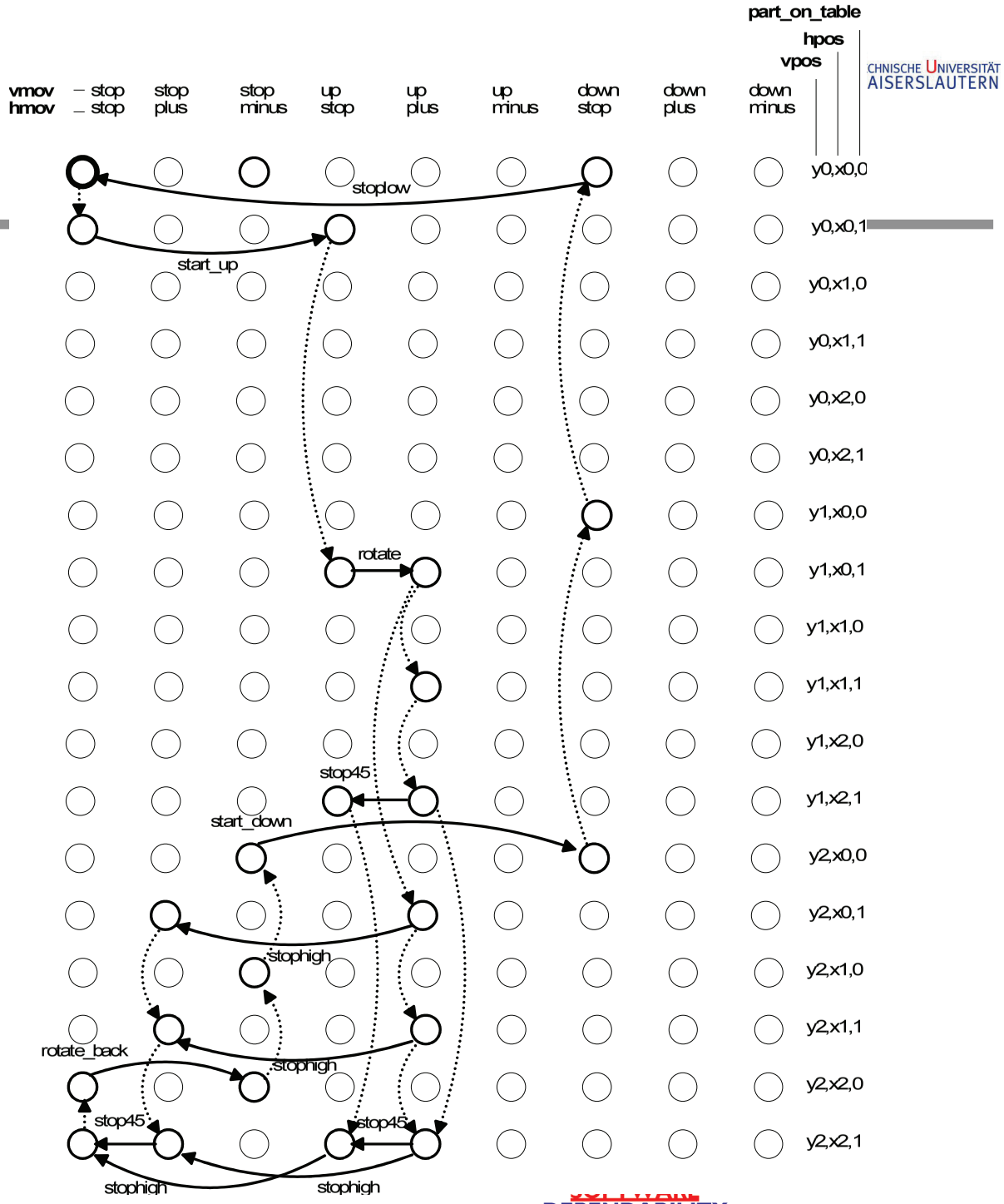


- Next Step: Combination of the state machine of the controller and of the state machine of the elevating rotable table to a so-called product fsm (finite state machine)

Symbolic Model Checking
State space of the example

The number of states of the product automat is the product of the number of states of the control automat and of the automat of the controlled system.

Here, those transitions are displayed that can actually be passed through in the product automat. Steps of the controller are drawn as solid lines and steps of the controlled system are represented as dashed lines.



Symbolic Model Checking
State space of the example

The safety requirement:

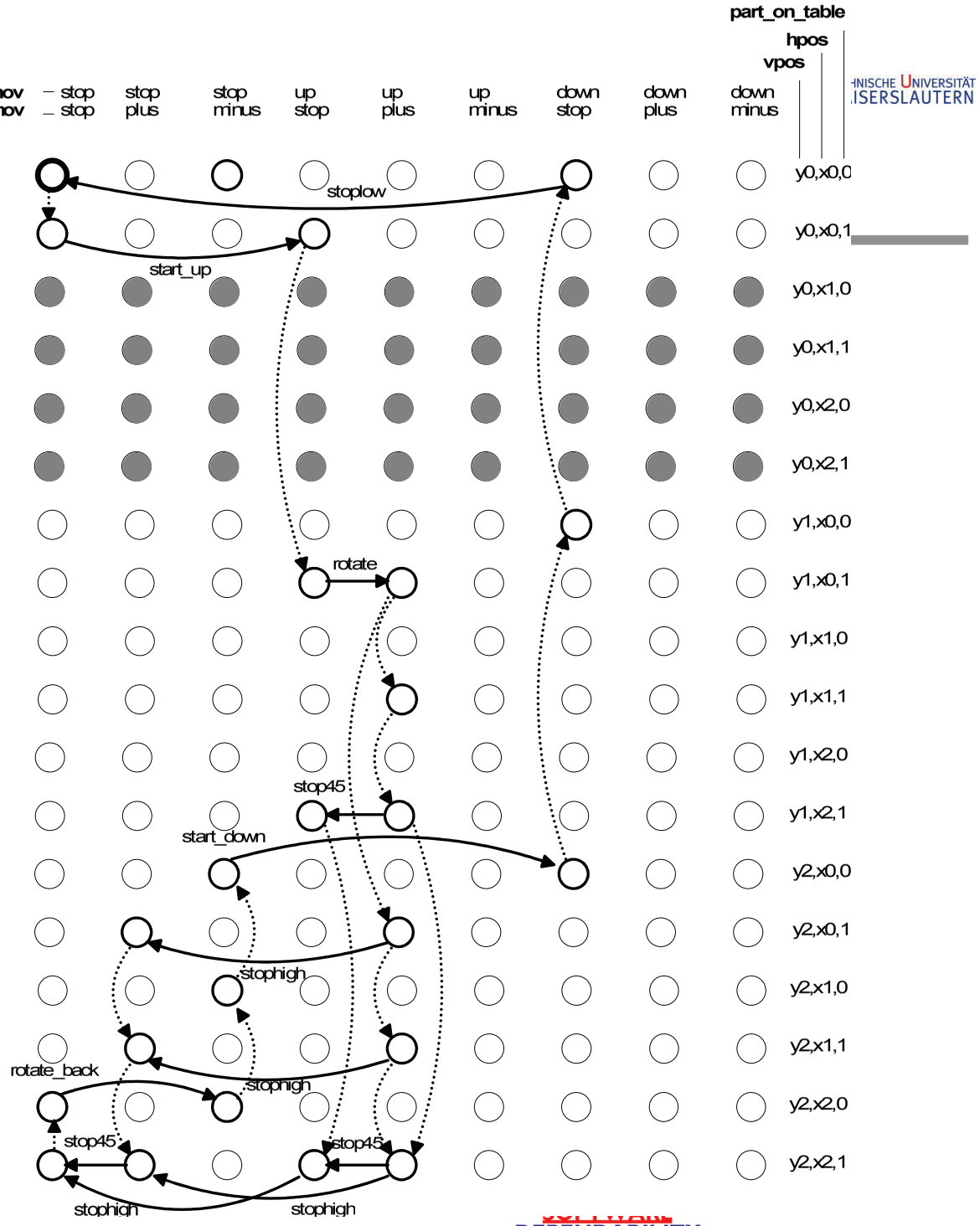
AG ~[vpos=y0 /\ (hpos=x1 \/ hpos=x2)]

resp.:

~EF [vpos=y0 /\ (hpos=x1 \/ hpos=x2)]

States, for which [vpos=y0 /\ (hpos=x1 \/ hpos=x2)] holds, are marked in grey. The safety requirement demands that none of the possible paths contains such a state

=> Reachability analysis



Symbolic Model Checking
Proof of safety requirements

- Safety requirements can often be checked by perfoming reachability analyses of the state space: “A system is safe, if unsafe states are not reachable.”
- Algorithm:

Initialize the set of reachale states E with the initial state Z

Calculate F as the set of direct successor states of set E.

$E = E \cup F$

until ***E*** *stays unchanged (so-called fix point iteration)*

Calulate the intersection S of E and of the set of the unsafe states ***U;*** $S = E \cap U$

$S = \emptyset$	
T	F
system is safe	*system is unsafe*
	Determine a path from the initial state into an unsafe state an example for unsafe behavior

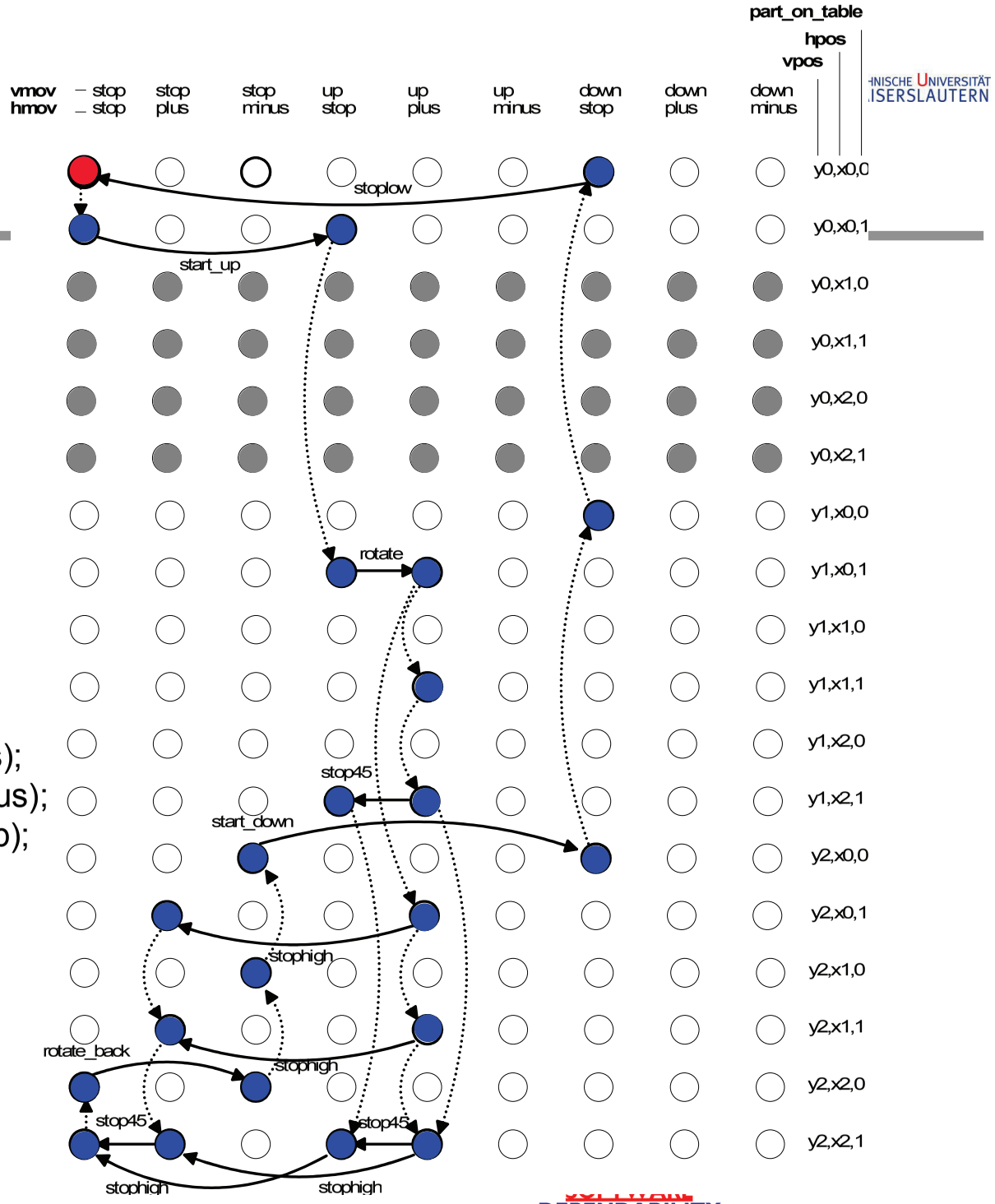
Symbolic Model Checking Proof of safety requirements

- Algorithm safety analysis:

E= {(y0,x0,0,stop,stop); (y0,x0,1,stop,stop);
(y0,x0,1,up,stop); (y1,x0,1,up,stop);
(y1,x0,1,up,plus); (y1,x1,1,up,plus);
(y1,x2,1,up,plus); (y2,x2,1,up,plus);
(y2,x0,1,up,plus); (y2,x1,1,up,plus);
(y1,x2,1,up,stop); (y2,x2,1,up,stop);
(y2,x0,1,stop,plus); (y2,x1,1,stop,plus);
(y2,x2,1,stop,plus); (y2,x2,1,stop,stop);
(y2,x2,0,stop,stop); (y2,x2,0,stop,minus);
(y2,x1,0,stop,minus); (y2,x0,0,stop,minus);
(y2,x0,0,down,stop); (y1,x0,0,down,stop);
(y0,x0,0,down,stop)}

U= {(y0,x1,-,-,-); (y0,x2,-,-,-)}

E ∩ **U**= ∅ => safety requirement is fulfilled

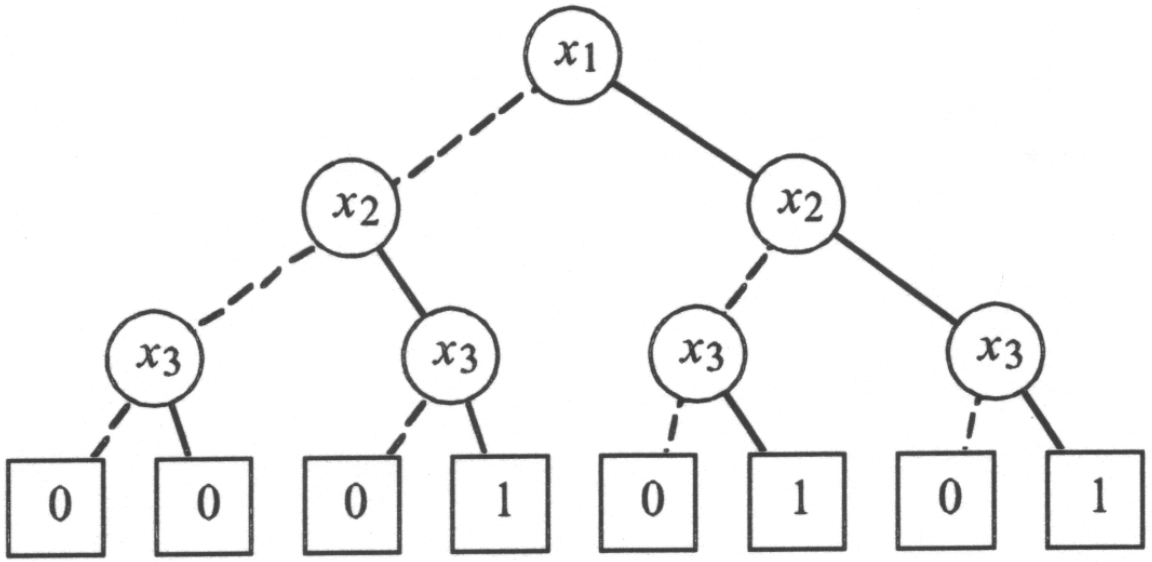


Symbolic Model Checking
Efficient implementation

- The size of the state space grows exponentially with the number of state variables => an efficient implementation is necessary in order to apply model checking to real, large systems.
- Common implementation of Symbolic Model Checking: Use of so-called OBDDs *(Ordered Binary Decision Diagrams)*:
  - Often very compact representation of the so-called characteristical function of those state machines that occur in practical applications
  - Efficient evaluation algorithms

- OBDDs are a notation for the description of Boolean function
- OBDDs are a so-called canonical representation (with a given variable order).

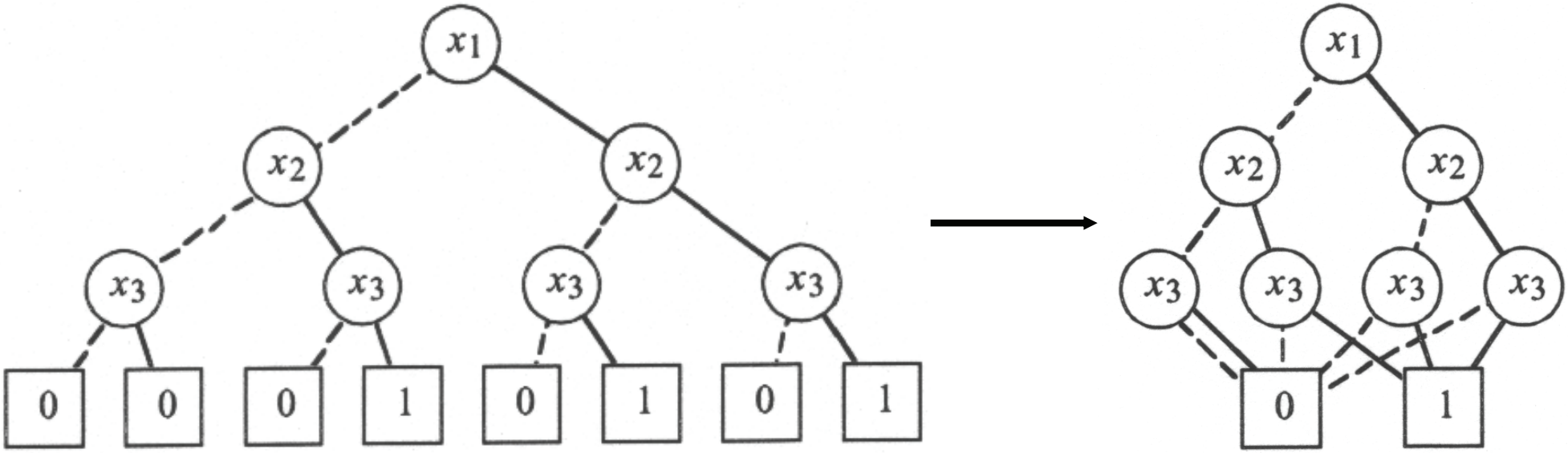
x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



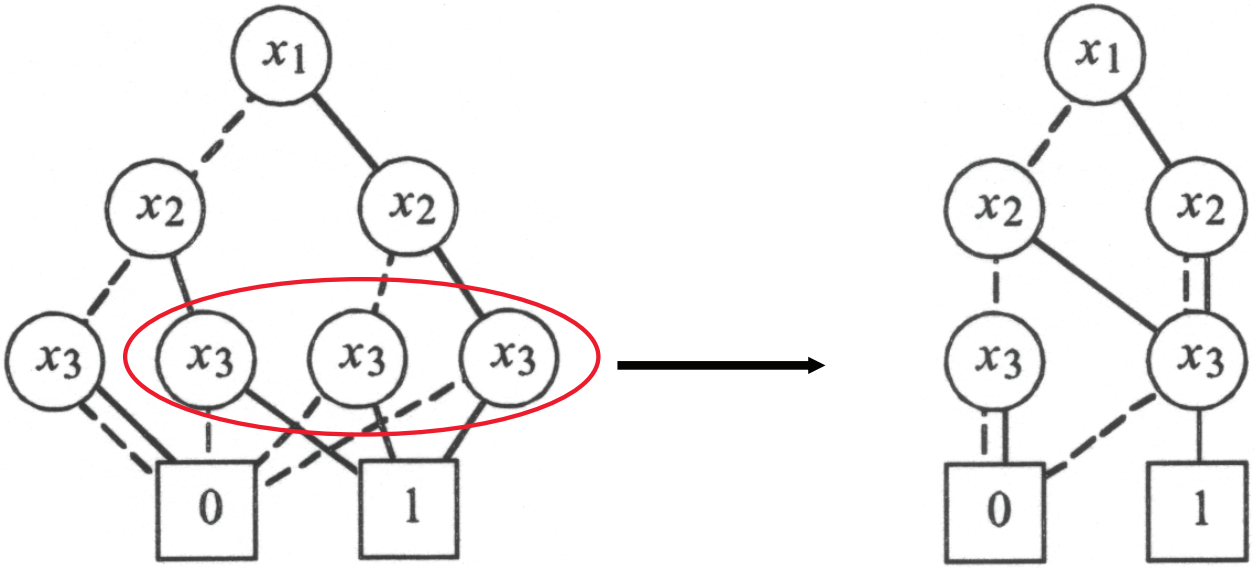
Decision table and decision tree as basis for the generation of an OBDD

0: dashed lines; 1: solid lines

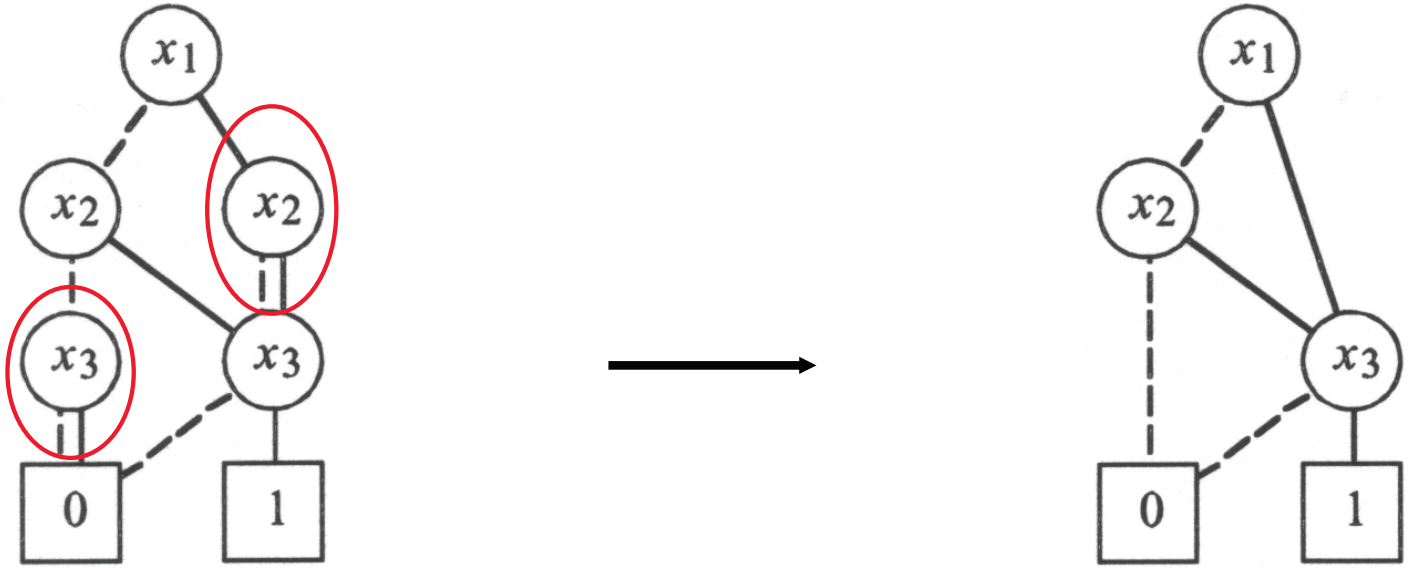
- Removal of redundant terminal knots



- Removal of identical non-terminal knots

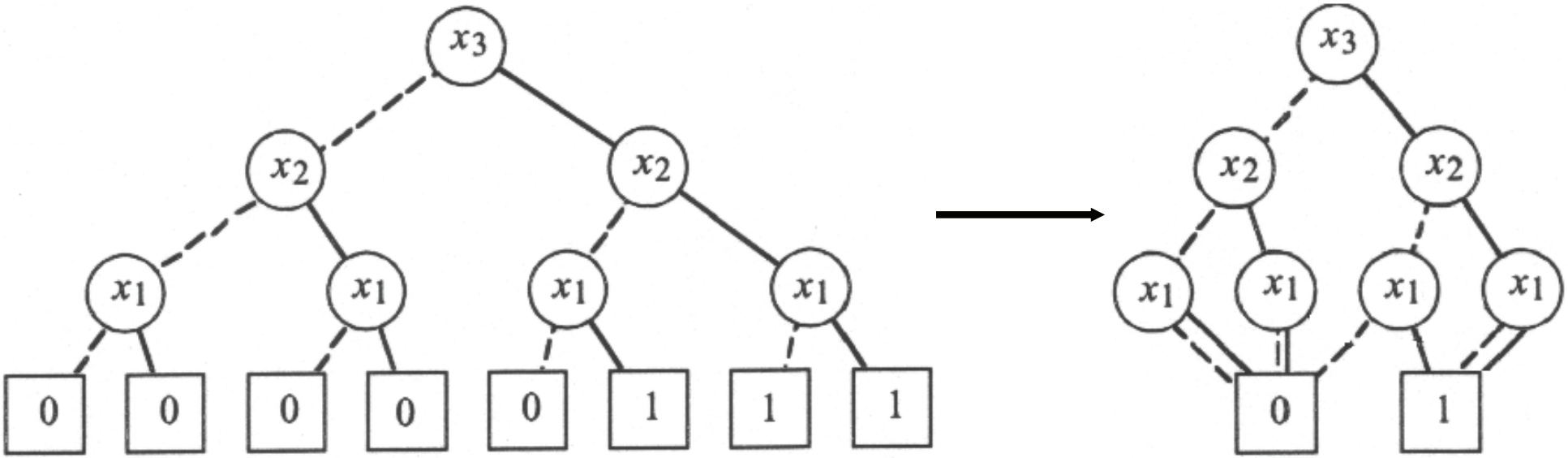


- Removal of redundant tests



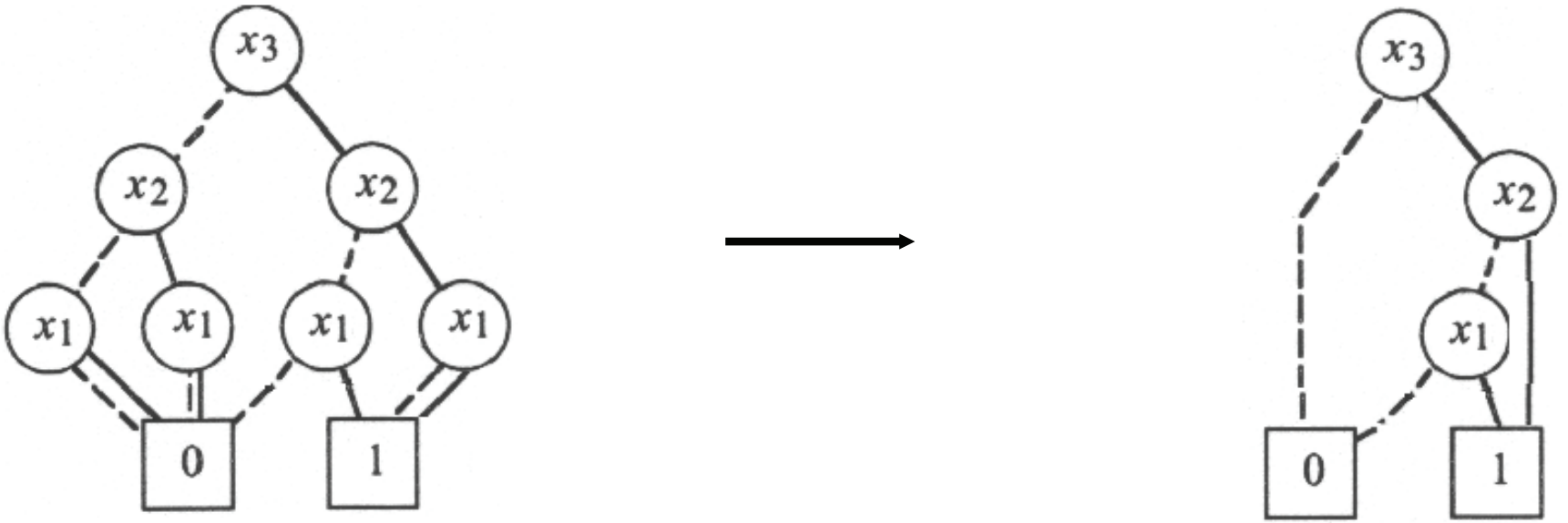
Symbolic Model Checking
OBDDs

- Same function, different variable order



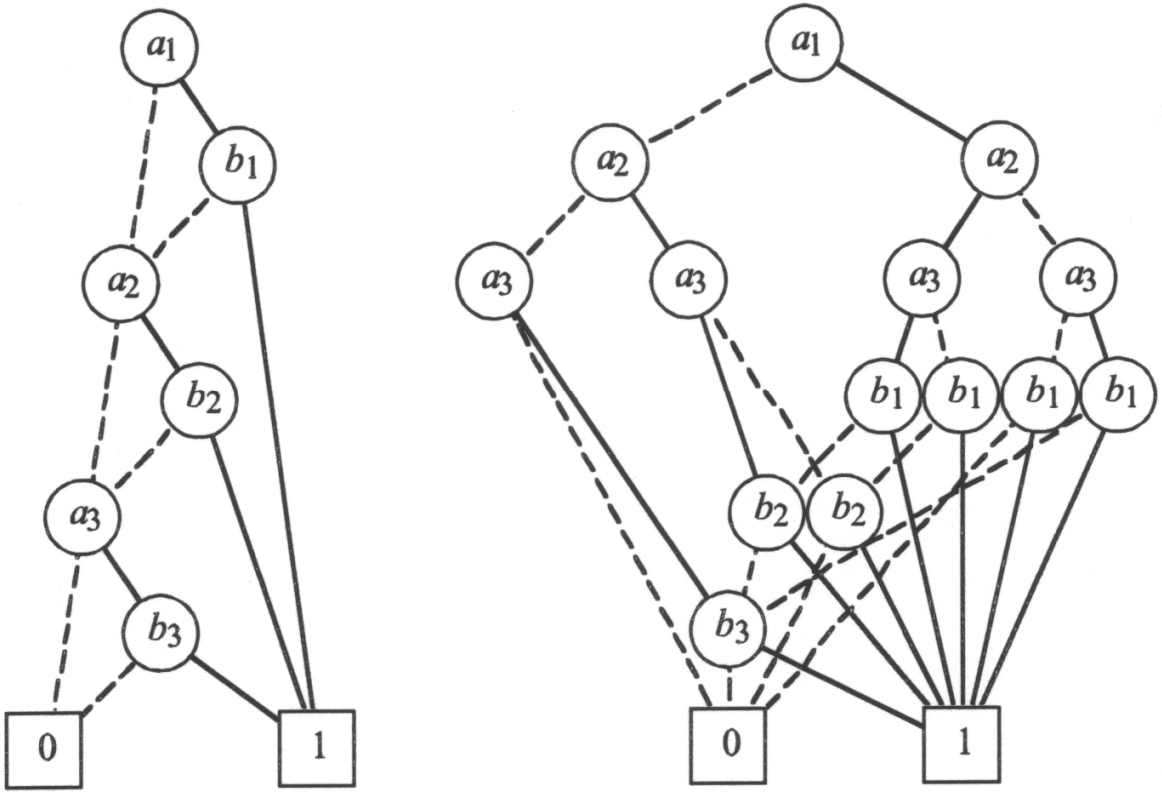
Symbolic Model Checking
OBDDs

- Same function, different variable order



- Result: Different OBDD
- The variable order may have considerable influence on the size of the OBDDs.

- The variable order may have considerable influence on the size of the OBDDs.
- The function a1 b1 + a2 b2 + a3 b3 with two different variable orders



Symbolic Model Checking
Representing state machines with OBDDs

- Representation of the characteristical function of the fsm as an OBDD
- Selection of a binary coding for all state variables and, if necessary, events, e.g.,:

Y	yb	ya
y0	0	0
y1	0	1
y2	1	0
-	1	1

X	xb	xa
x0	0	0
x1	0	1
x2	1	0
-	1	1

part_on_table	p
no	0
yes	1

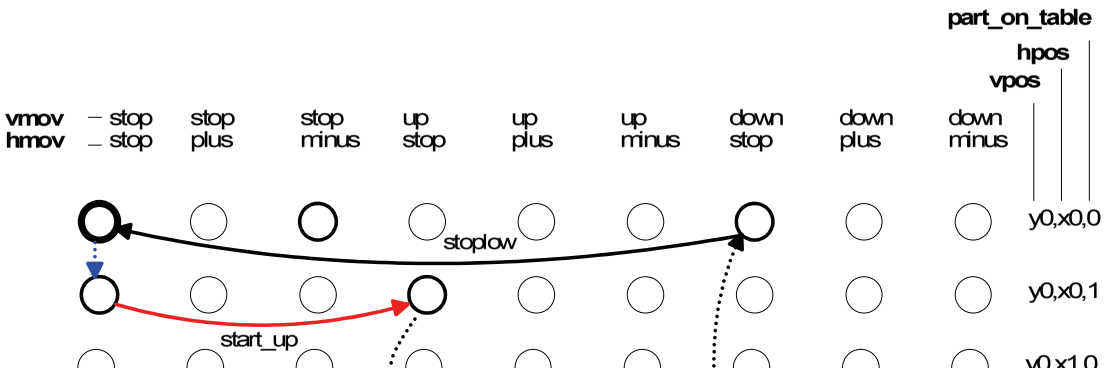
vmov	vb	va
stop	0	0
down	0	1
up	1	0
-	1	1

hmov	hb	ha
stop	0	0
minus	0	1
plus	1	0
-	1	1

- Duplication of state variables for the description of present and successor (next) states, e.g. yb, ya (binary coding of the state variable Y in the initial state); yb‘, ya‘ (binary coding of the state variable Y in the next state)

Symbolic Model Checking
Coding of state machines with OBDDs

- If the state space is coded as described, the characteristical function has the value 1, if there exists a transition between two states under consideration, otherwise it has the value 0.



Present state **Next state (')**

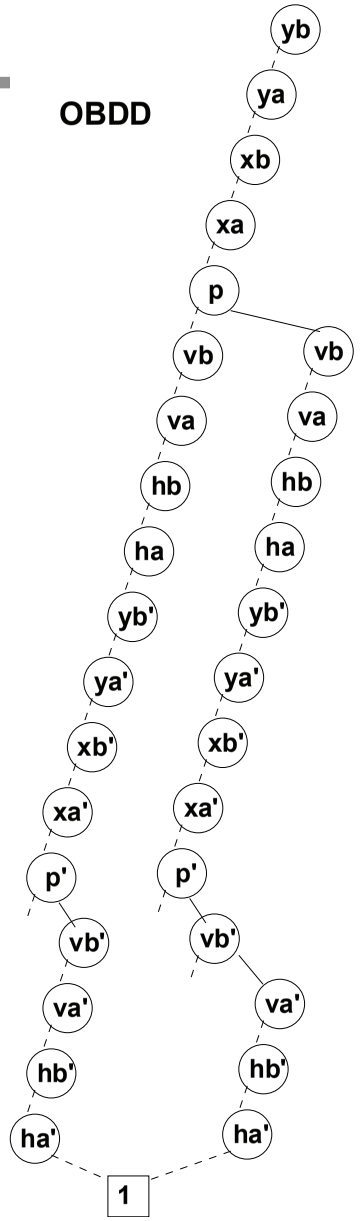
(y0,x0,0,stop,stop) -> (y0,x0,1,stop,stop) =>
yb ya xb xa p vb va hb ha yb' ya' xb' xa' p' vb' va' hb' ha'

+

(y0,x0,1,stop,stop) -> (y0,x0,1,up,stop) =>
yb ya xb xa p vb va hb ha yb' ya' xb' xa' p' vb' va' hb' ha'

+

...



- Efficient analysis algorithms exist for OBDDs, e.g., the apply-function, that combines two functions f and g given as OBDDs with an operator *op* to a new function: *f op g*
- The apply-function on OBDDs uses the following rule:

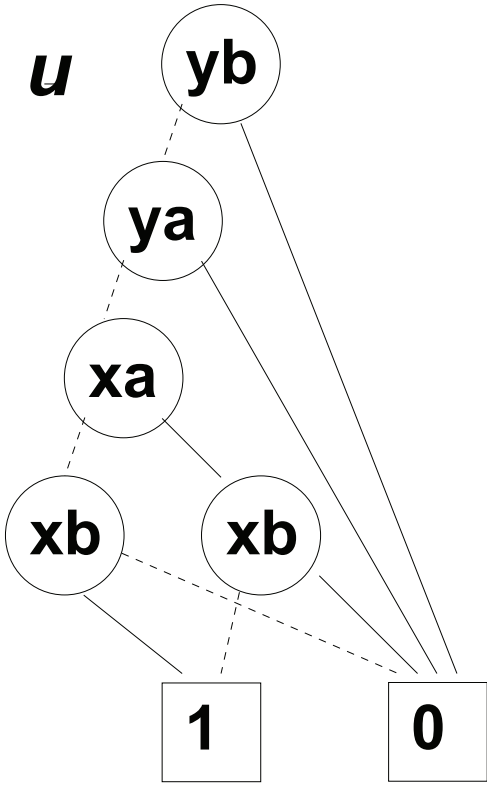
$$f \langle op \rangle g = \overline{x} \cdot \left(f|_{x \leftarrow 0} \langle op \rangle g|_{x \leftarrow 0} \right) + x \cdot \left(f|_{x \leftarrow 1} \langle op \rangle g|_{x \leftarrow 1} \right)$$

The application of the operation on the OBDDs thus can be recursively applied to the single knots. If – doing this - a dominant terminal value occurs in one of the combined OBDDs (e.g., 1 with the OR-operator, 0 with the AND-operator), the appropriate terminal value can be directly inserted and the recursion stops.

Symbolic Model Checking
Analysis of state machines with OBDDs

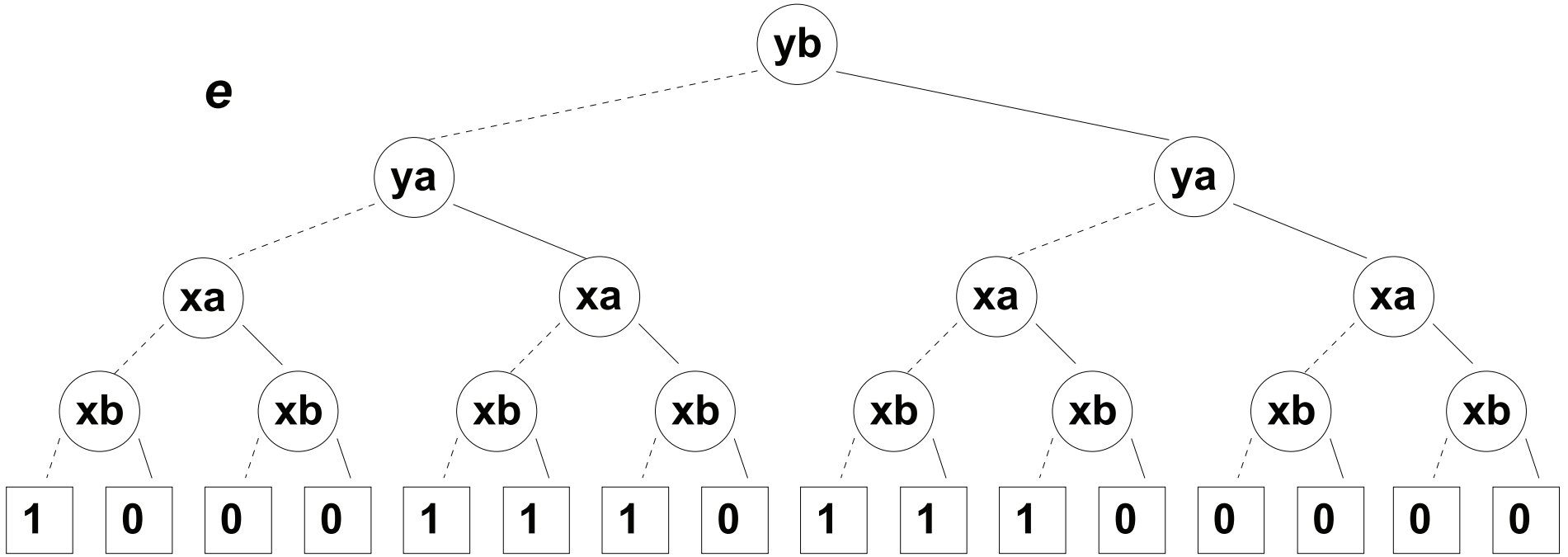
- Example: Checking the safety requirement of the elevating rotable table:
  the intersection of the reachable states and the unsafe states has to be empty, i.e.,
  - if *e* is a Boolean function represented as an OBDD, which has the value 1 for all available states **E** and
  - if *u* is a Boolean function represented as an OBDD, which has the value 1 for all unsafe states **U**,
  - (*e* AND *u*) must be reducible to the Boolean constant FALSE .
- i.e., the state is either available or unsafe, but not *available and unsafe*.

- The function *u,* that describes the set **U** of unsafe states:
  - For unsafe states only position variables are interesting, all other variables can be seen as „don‘t cares“.
  - Based on the given binary coding the following OBDD is generated:

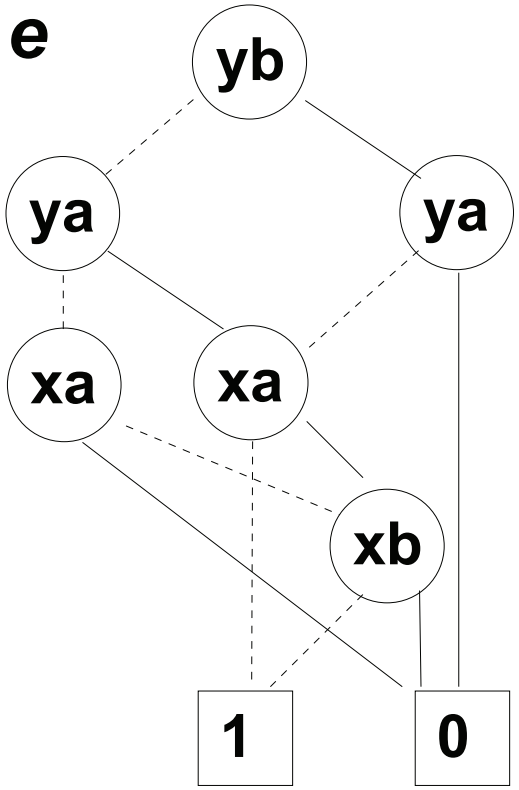


Symbolic Model Checking
Analysis of state machines with OBDDs

- The function *e*, that describes the set **E** of reachable states:
  - Only position variables must be considered; all other variables are regarded as „don‘t cares“.
  - Based on the given binary coding the following tree is generated:

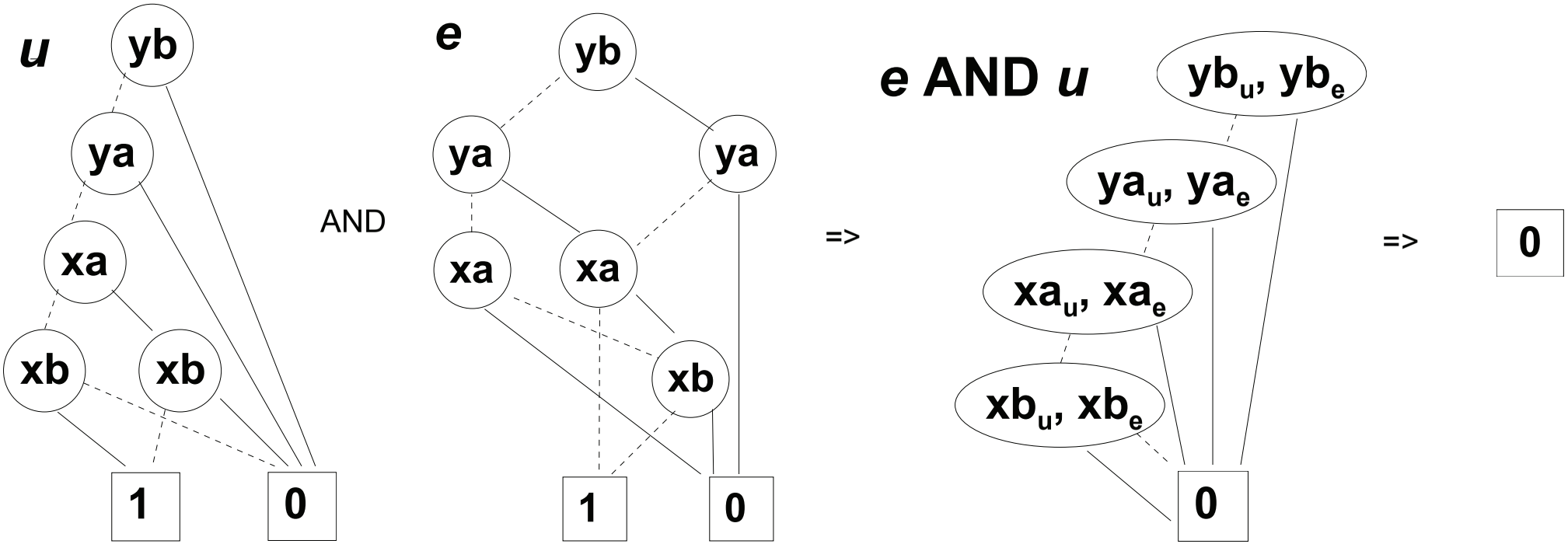


- The following OBDD describes the set **E** (reachable states):



Symbolic Model Checking
Analysis of state machines with OBDDs

- The AND-operation:



- There are no reachable states that are unsafe, because the AND-operation applied to both sets of states produces the Boolean constant FALSE. The system is safe with regard to the defined safety requirement.

Symbolic Model Checking
Summary

- Symbolic Model Checking is an powerful formal technique for prooving properties, that requires a finite state description of behavior.
- In many practical applications, OBDDs are appropriate, efficient descriptions of state machines.
- Symbolic Model Checking produces either a validation of required properties or a counter example.
- Due to the widespread use of finite state machines in software engineering, the importance of Symbolic Model Checking is growing.