
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

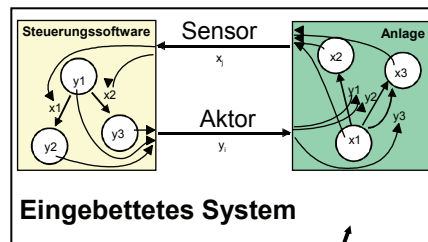
Symbolic Model Checking

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

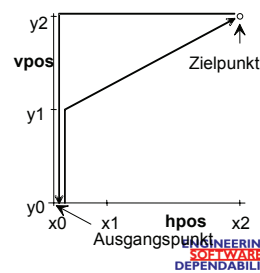
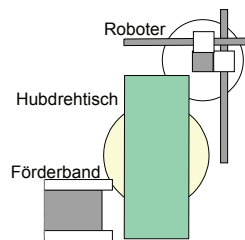
Symbolic Model Checking Beispiel: Der Hubdrehtisch (Hardware)

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

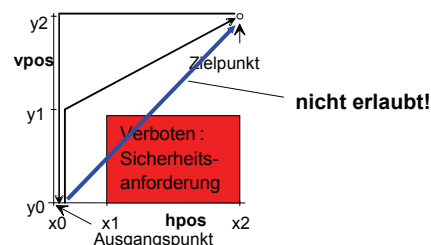
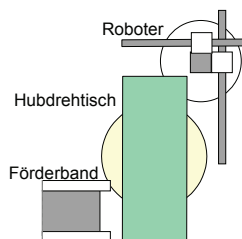
Symbolic Model Checking Beispiel: Der Hubdrehtisch (Steuerungsstrategie: Software)

Die Ausgangsposition des Tisches ist $hpos=x_0$, $vpos=y_0$. 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 $vpos=y_1$ erreicht wird, so wird auch die horizontale Bewegung gestartet ($hmov=plus$). Wenn der Tisch die Zielposition ($hpos=x_2$, $vpos=y_2$) 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 $hpos=x_0$ 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 $[vpos=y_0 \wedge (hpos=x_1 \vee hpos=x_2)]$ erfüllt ist. Aus diesem Grund wird bei der Aufwärtsbewegung die horizontale Bewegungsrichtung erst an der vertikalen Position y_1 gestartet und nicht direkt zu Beginn. Analog gilt dies auch für die Abwärtsbewegung.

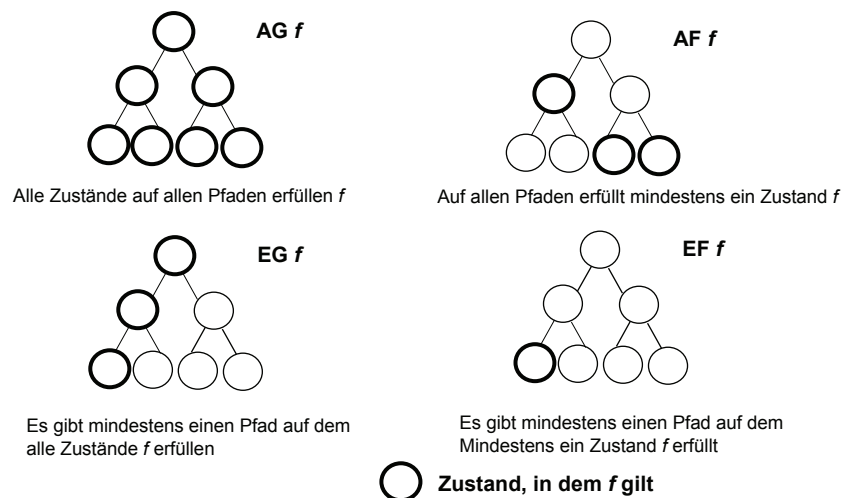


Symbolic Model Checking Exkurs: Temporale Logik; hier CTL

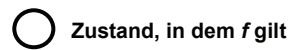
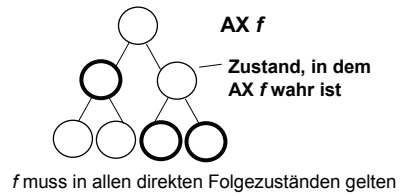
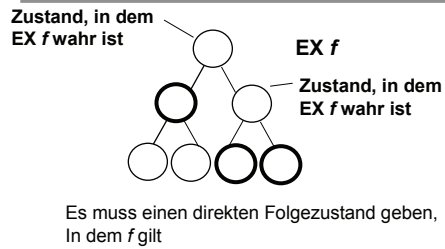
CTL (Computation Tree Logic):

- ☐ Forward-time operators:
 - G: globally, invariantly
 - F: sometime in the future
 - X: next time
 - U: Until
- ☐ Path quantifiers:
 - A: all computation paths
 - E: some computation path (exists)
- ☐ In CTL muss direkt vor dem *forward-time operator* ein *path quantifier* stehen, also z.B. **AG** f oder **EF** f

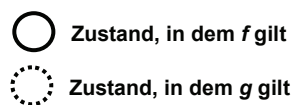
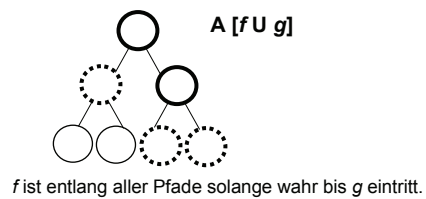
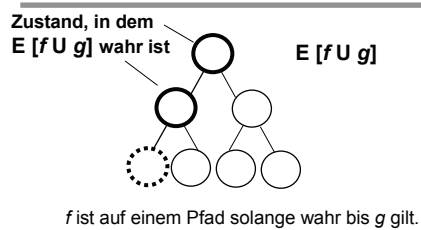
Symbolic Model Checking Exkurs: Temporale Logik; hier CTL



Symbolic Model Checking Exkurs: Temporale Logik; hier CTL



Symbolic Model Checking Exkurs: Temporale Logik; hier CTL



Symbolic Model Checking

Beispiel Hubdrehtisch: Sicherheitsanforderung



Die Sicherheitsanforderung ist:
Es darf niemals ein Zustand auftreten, für den gilt:

$$[\text{vpos}=\text{y0} \wedge (\text{hpos}=\text{x1} \vee \text{hpos}=\text{x2})]$$

In CTL:

$$\text{AG } \sim[\text{vpos}=\text{y0} \wedge (\text{hpos}=\text{x1} \vee \text{hpos}=\text{x2})]$$

oder äquivalent:

$$\sim\text{EF } [\text{vpos}=\text{y0} \wedge (\text{hpos}=\text{x1} \vee \text{hpos}=\text{x2})]$$

$\sim$: 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;
```

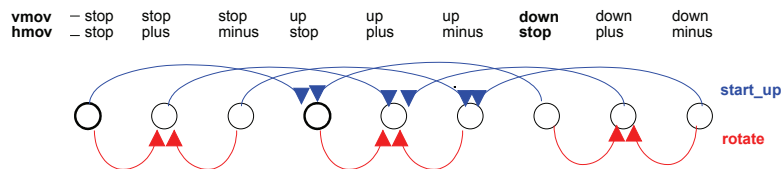
Transitions

```
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.



```

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' = y2)
    ∨ ('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)

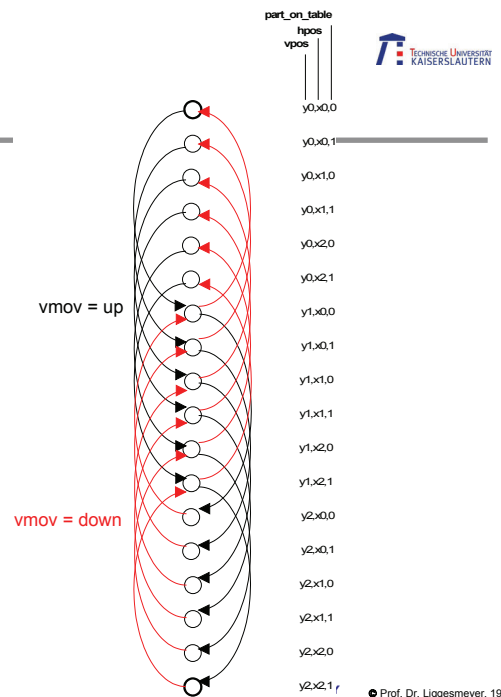
  ∨ ('table.hmov' = plus ∧ 'table.hpos' = x0) ∧ until('table.hpos' = x0, 'table.hpos' = x1)
  ∨ ('table.hmov' = plus ∧ 'table.hpos' = x1) ∧ until('table.hpos' = x1, 'table.hpos' = x2)
  ∨ ('table.hmov' = plus ∧ 'table.hpos' = x2) ∧ x('table.hpos' = x2)

  ∨ ('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

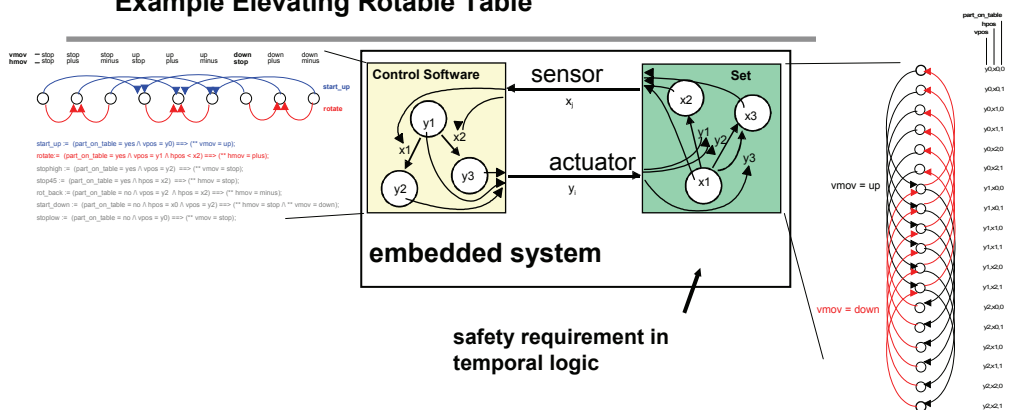
The state space of the elevating rotatable table is defined as the combination of the sensor values.
Aktuator data is triggering transitions in this state space.

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



Safety and Reliability of Embedded Systems

Symbolic Model Checking Example Elevating Rotatable Table



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

Safety and Reliability of Embedded Systems

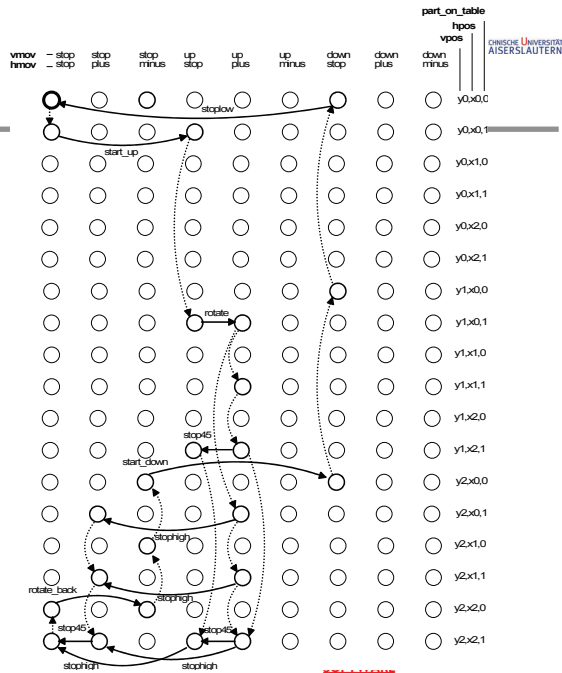
ENGINEERING
SOFTWARE
DEPENDABILITY

Prof. Dr. Uggemeyer, 20

Symbolic Model Checking State space of the example

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

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



Safety and Reliability of Embedded Systems

DEPENDABILITY

Prof. Dr. Uggemeyer, 21

Symbolic Model Checking State space of the example

The safety requirement:

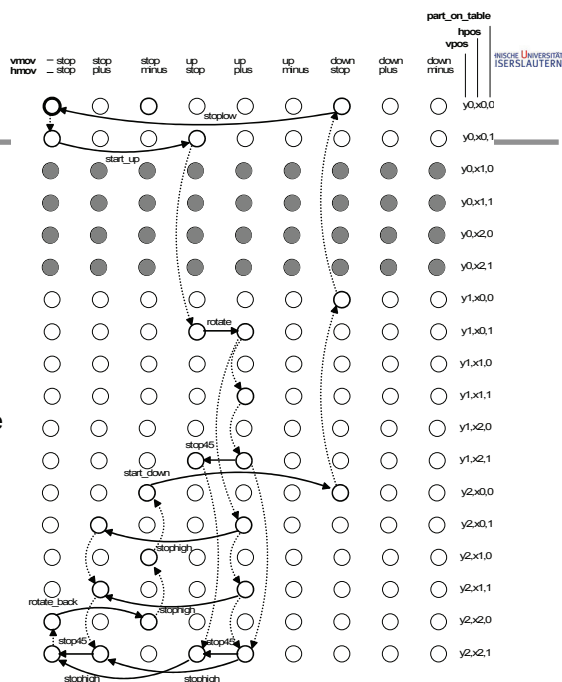
AG $\sim[vpos=y0 \wedge (hpos=x1 \vee hpos=x2)]$

resp.:

$\sim EF$ $[vpos=y0 \wedge (hpos=x1 \vee hpos=x2)]$

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

=> Reachability analysis



Safety and Reliability of Embedded Systems

DEPENDABILITY

Prof. Dr. Uggemeyer, 22

Symbolic Model Checking Proof of safety requirements

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

Initialize the set of reachable 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)	
Calculate the intersection S of E and of the set of the unsafe states U ; $S = E \cap U$	
T	F
system is safe	system is unsafe
	Determine a path from the initial state into an unsafe state an example for unsafe behavior

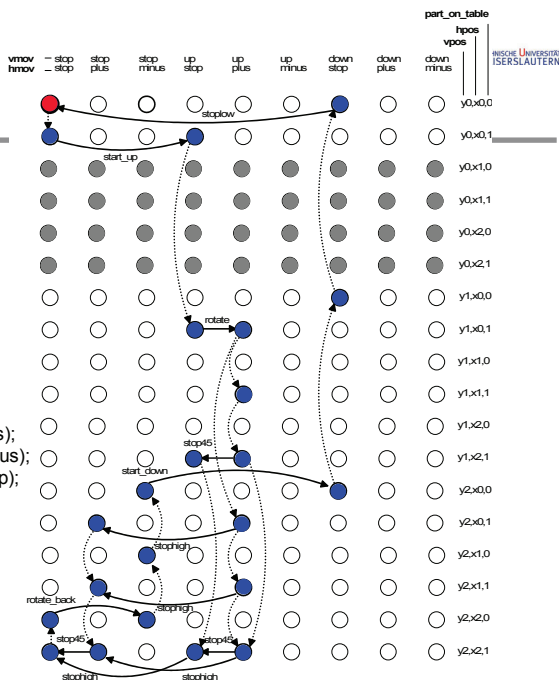
Symbolic Model Checking Proof of safety requirements

- Algorithm safety analysis:

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

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

$E \cap U = \emptyset \Rightarrow$ safety requirement is fulfilled



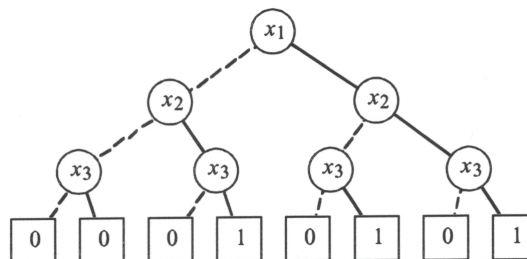
Symbolic Model Checking Efficient implementation

- 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 characteristic function of those state machines that occur in practical applications
 - Efficient evaluation algorithms

Symbolic Model Checking Excursus: OBDDs

- 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

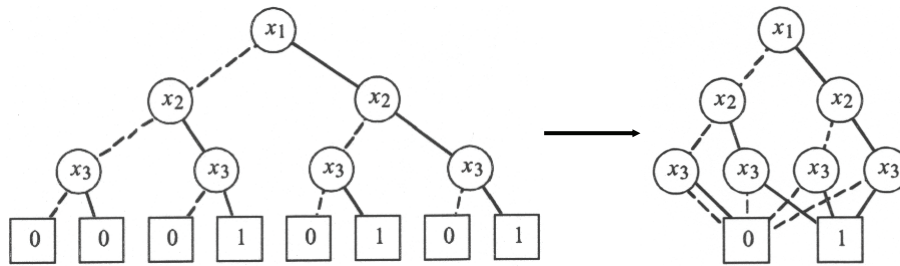


Decision table and decision tree as basis for the generation of an OBDD

0: dashed lines; 1: solid lines

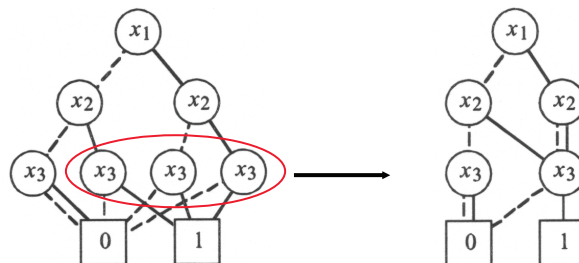
Symbolic Model Checking Excursus: OBDDs

- Removal of redundant terminal knots



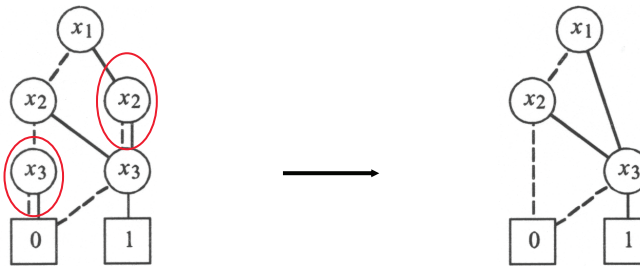
Symbolic Model Checking OBDDs

- Removal of identical non-terminal knots



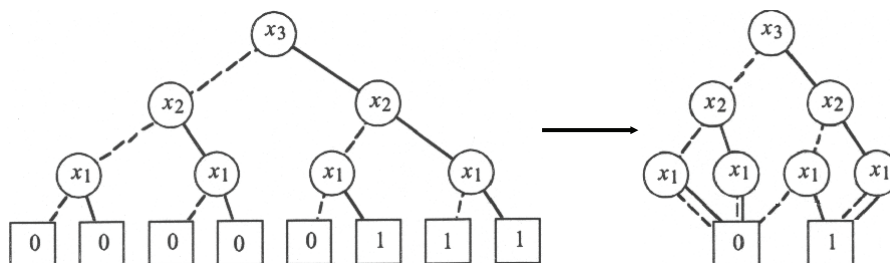
Symbolic Model Checking OBDDs

- Removal of redundant tests



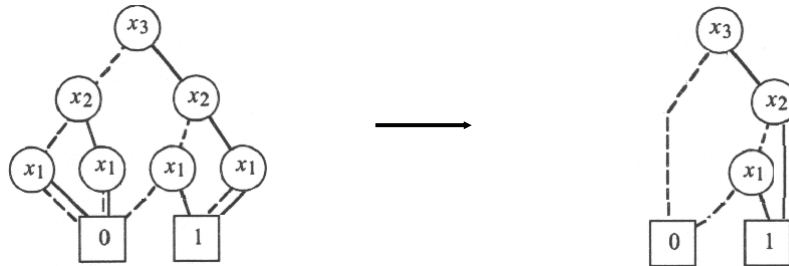
Symbolic Model Checking OBDDs

- Same function, different variable order



Symbolic Model Checking OBDDs

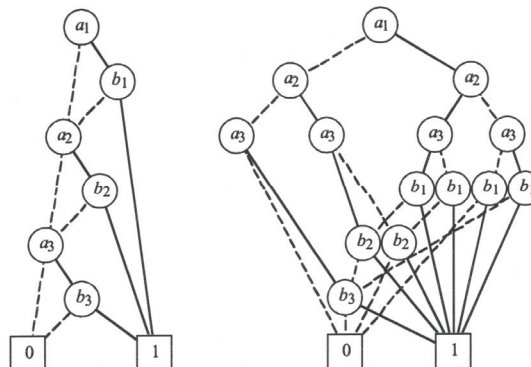
- Same function, different variable order



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

Symbolic Model Checking OBDDs

- The variable order may have considerable influence on the size of the OBDDs.
- The function $a_1 b_1 + a_2 b_2 + a_3 b_3$ with two different variable orders



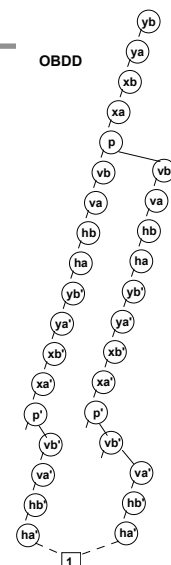
- | Y | yb ya | X | xb xa | part_on_table | p | vmove | vb va | hmove | hb ha |
|----|-------|----|-------|---------------|---|-------|-------|-------|-------|
| y0 | 0 0 | x0 | 0 0 | no | 0 | stop | 0 0 | stop | 0 0 |
| y1 | 0 1 | x1 | 0 1 | yes | 1 | down | 0 1 | minus | 0 1 |
| y2 | 1 0 | x2 | 1 0 | | | up | 1 0 | plus | 1 0 |
| - | 1 1 | - | 1 1 | | | - | 1 1 | - | 1 1 |

- ENGINEERING
SOFTWARE
DEPENDABILITY

● Prof. Dr. Liggesmeyer, 33

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

-
- The diagram shows a sequence of states (circles) connected by transitions (arrows). The transitions are labeled with instructions: 'vmov', 'hmov', 'stop', 'plus', 'minus', 'up', 'down', 'stoplow', 'startup', and 'part_on_table'. The diagram illustrates the execution flow, including a loop and a final state labeled 'part_on_table'.

$$\begin{aligned} (y_0, x_0, 0, \text{stop}, \text{stop}) \rightarrow (y_0, x_0, 1, \text{stop}, \text{stop}) &\Rightarrow \overline{y_b} y_a \overline{x_b} \overline{x_a} p \overline{v_b} \overline{v_a} \overline{h_b} \overline{h_a} y_b' y_a' \overline{x_b'} \overline{x_a'} p' \overline{v_b'} \overline{v_a'} \overline{h_b'} \overline{h_a'} \\ &+ \\ (y_0, x_0, 1, \text{stop}, \text{stop}) \rightarrow (y_0, x_0, 1, \text{up}, \text{stop}) &\Rightarrow \overline{y_b} y_a \overline{x_b} \overline{x_a} p \overline{v_b} \overline{v_a} \overline{h_b} \overline{h_a} y_b' y_a' \overline{x_b'} \overline{x_a'} p' \overline{v_b'} \overline{v_a'} \overline{h_b'} \overline{h_a'} \\ &+ \end{aligned}$$


ENGINEERING
SOFTWARE
DEPENDABILITY

● Prof. Dr. Liigesmeyer, 34

Symbolic Model Checking Analysis of state machines with OBDDs



- 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: $f \text{ op } g$
- The apply-function on OBDDs uses the following rule:

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

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.

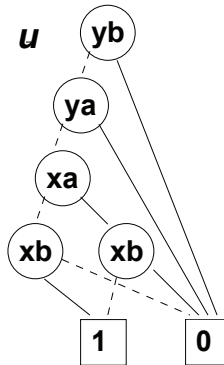
Symbolic Model Checking Analysis of state machines with OBDDs



- Example: Checking the safety requirement of the elevating rotatable 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 ,
 - $(e \text{ AND } u)$ must be reducible to the Boolean constant FALSE .
- i.e., the state is either available or unsafe, but not *available and unsafe*.

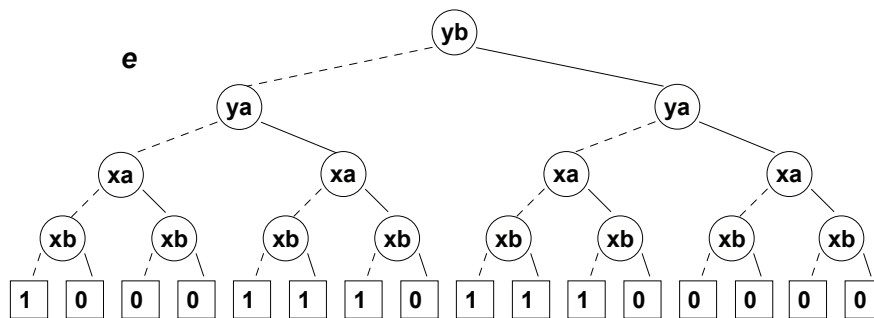
Symbolic Model Checking Analysis of state machines with OBDDs

- 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:



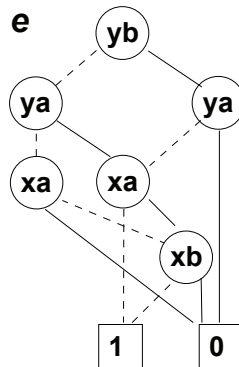
Symbolic Model Checking Analysis of state machines with OBDDs

- 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:



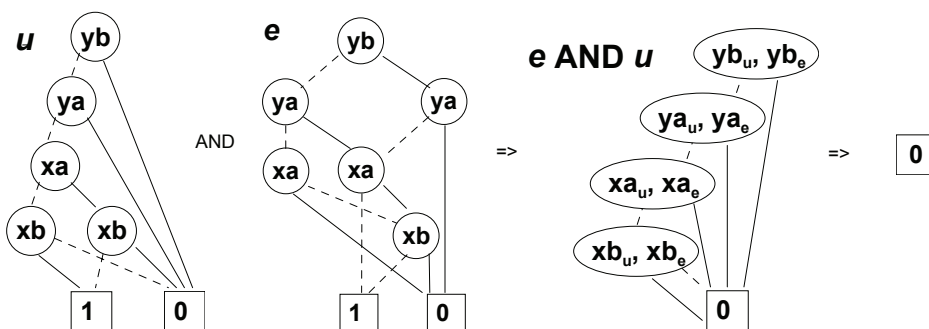
Symbolic Model Checking Analysis of state machines with OBDDs

- ☐ 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 a powerful formal technique for proving 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.