Einführung in die Technische Informatik

Validation und Verifikation Eingebetteter Systeme

Robert Wille

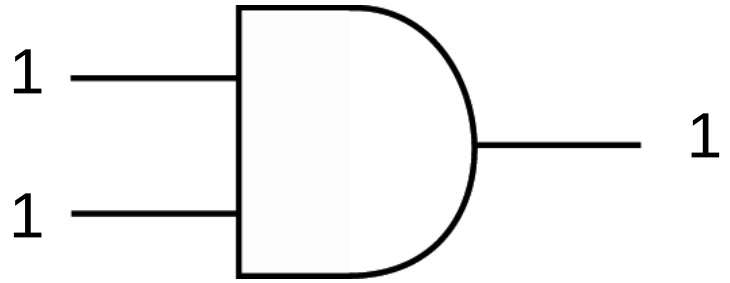
Motivation

- Verifikation wird immer wichtiger
  - Bis zu 80% der Entwurfskosten
- Schnell wachsender Markt
- Anforderung:
  - Effiziente Werkzeuge
  - Automatisierung (*„push-button tools“*)
  - Schnelle Algorithmen
- Systemebene

Existierende Techniken

- Simulation
  - Evaluation von gegebenen Stimuli
  - Deterministisch oder zufällig
- Emulation
  - Hardware Prototypen, z.B. FPGA
- Formale Verifikation
  - Exakte mathematische Modelle
  - Beweis der Korrektheit

Simulation



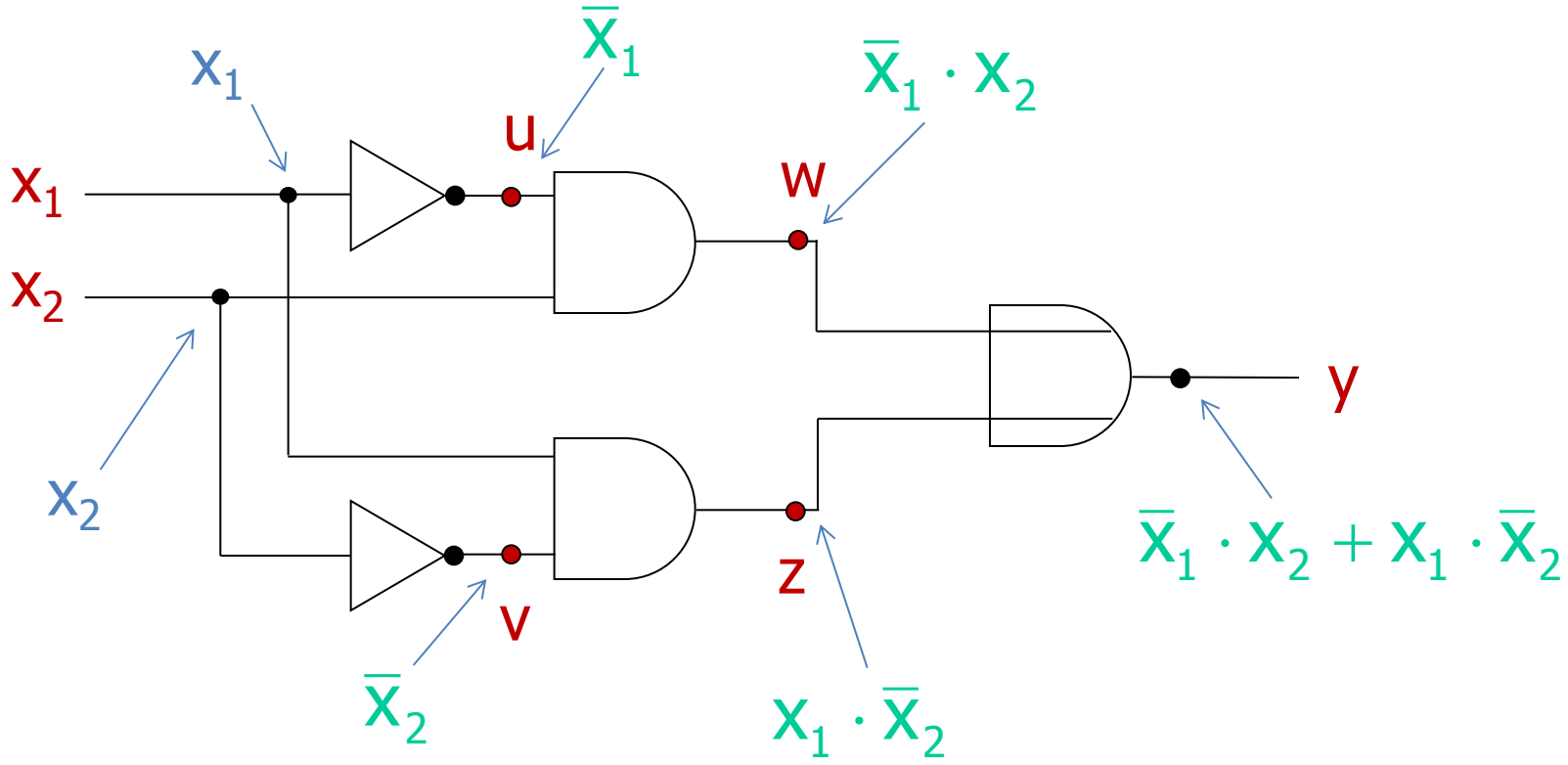
- Anwendung spezieller Werte
- Schnelle Berechnung
  (linear über Anzahl der Gatter)
- Aber:
  - Muss für jedes Eingabemuster neu durchgeführt werden
  - Vollständige Simulation nur für kleine Schaltungen möglich
    (exponentiell über Anzahl der Eingänge)

Symbolische Simulation

Symbolische Simulation benutzt zur Simulation eines Schaltkreises keine festen Booleschen Werte an den Inputs, sondern Boolesche Variablen.

Es wird dann zu jedem Signal die Boolesche Funktion, z.B. in Form eines Booleschen Ausdrucks oder eines BDDs, bestimmt, die das Signal berechnet.

Beispiel



Effiziente Darstellung der Funktionalität?

Funktionsrepräsentation

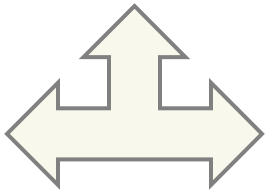
- Wahrheitstabelle
- DNF (SoP) und KNF (PoS)
  - Sum-of-products
    F = x1'x2x3 + x1x2'x3 + x1x2x3
  - Product-of-sums
    F = (x1+x2+x3) (x1+x2+x3') (x1+x2'+x3) (x1'+x2+x3) (x1'+x2'+x3)
- Entscheidungsdiagramme (z.B. BDDs)

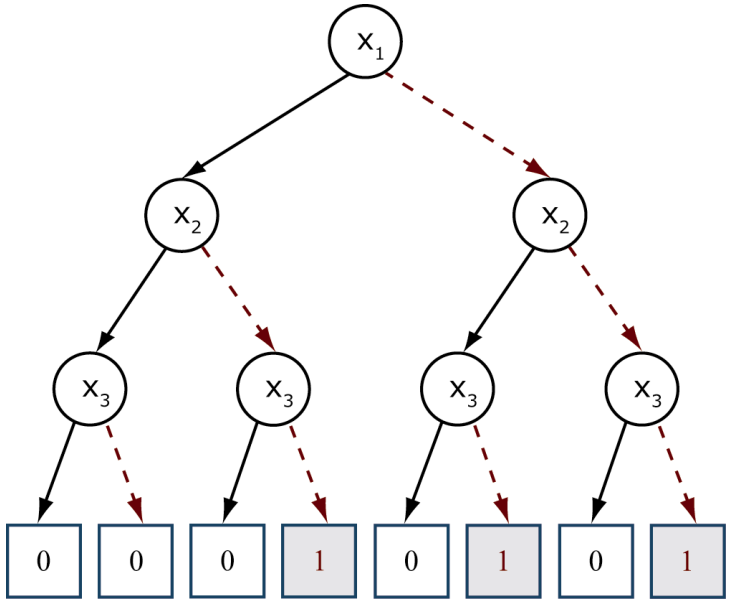
x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Entscheidungsdiagramme

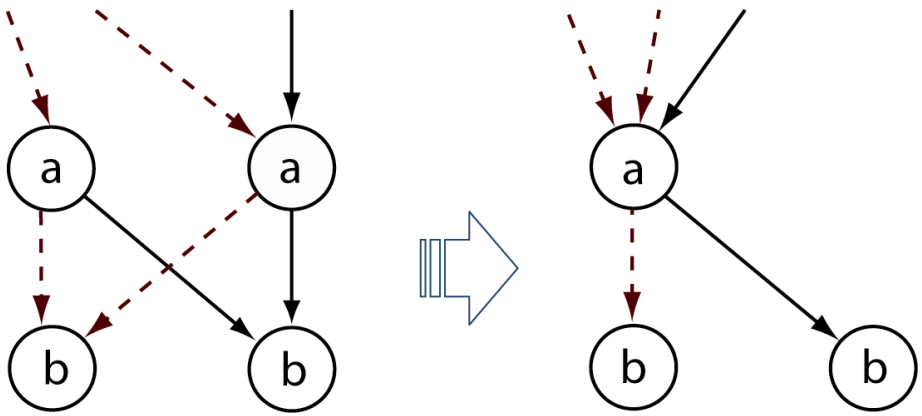
x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

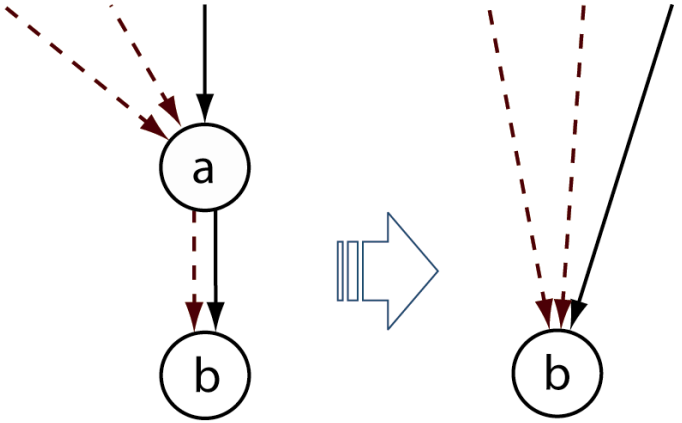
x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1
F	0	0	0	1	0	1	0	1



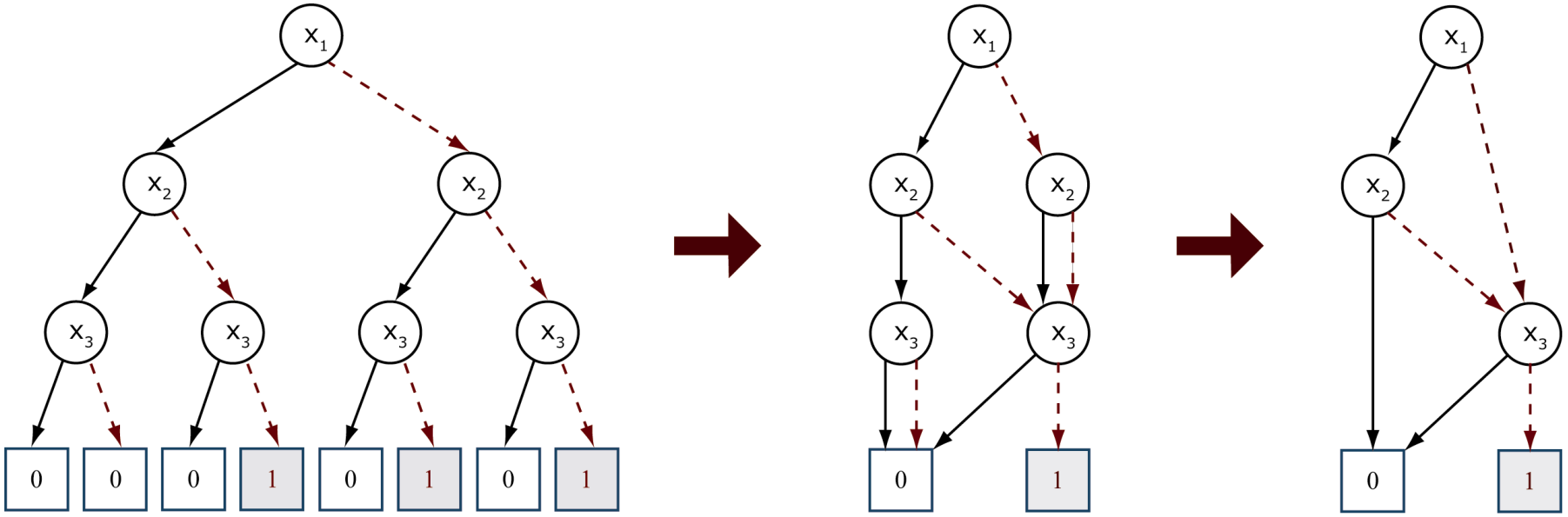


Reduktionsregeln

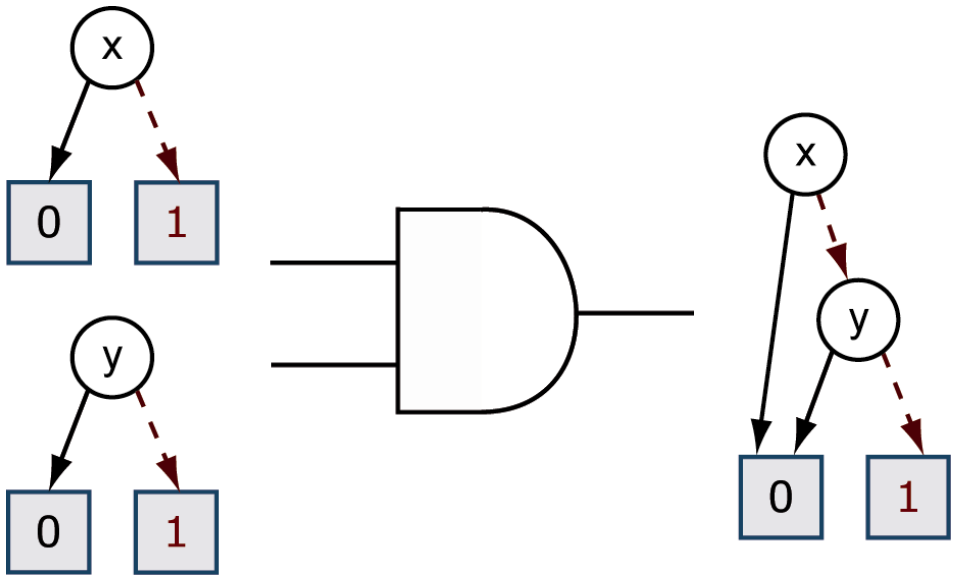




Beispiel



Symbolische Simulation mit BDDs



Existierende Techniken

- Simulation
  - Evaluation von gegebenen Stimuli
  - Deterministisch oder zufällig
- Emulation
  - Hardware Prototypen, z.B. FPGA
- **Formale Verifikation**
  - Exakte mathematische Modelle
  - Beweis der Korrektheit

Formale Verifikation

- Äquivalenzprüfung
  - Sind zwei Schaltungen äquivalent?

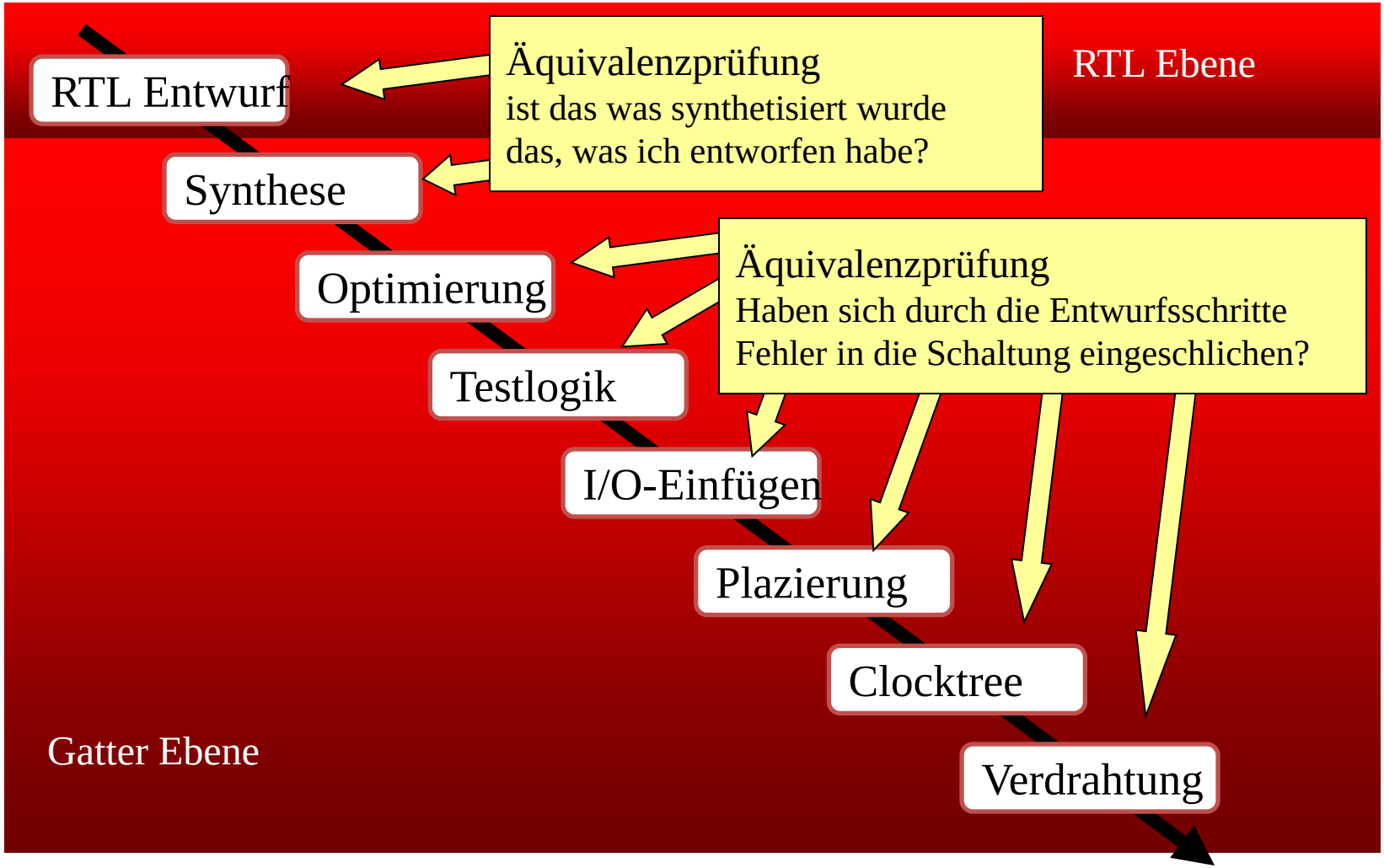
- Modellprüfung
  - Erfüllt eine Schaltung bestimmte (sequentielle) Eigenschaften?

Äquivalenzprüfung

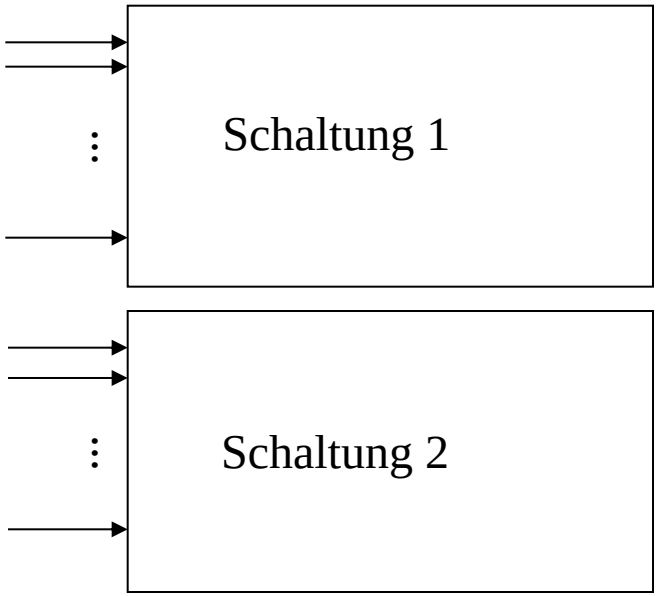
Engl. „*equivalence checking*“ (EC)

- Gegeben: Zwei digitale Schaltungen
- Gefragt: Haben beide die gleiche Funktionalität?
  - Bei kombinatorischen Schaltungen:
    Sind die Ausgänge bei gleichen Eingangsbelegungen gleich?
  - Bei sequentiellen Schaltungen:
    Sind die Ausgänge zu allen Zeitpunkten bei gleichen Eingabefolgen identisch?

Entwurfsflow mit Äquivalenzprüfung



Äquivalenzprüfung von Schaltkreisen

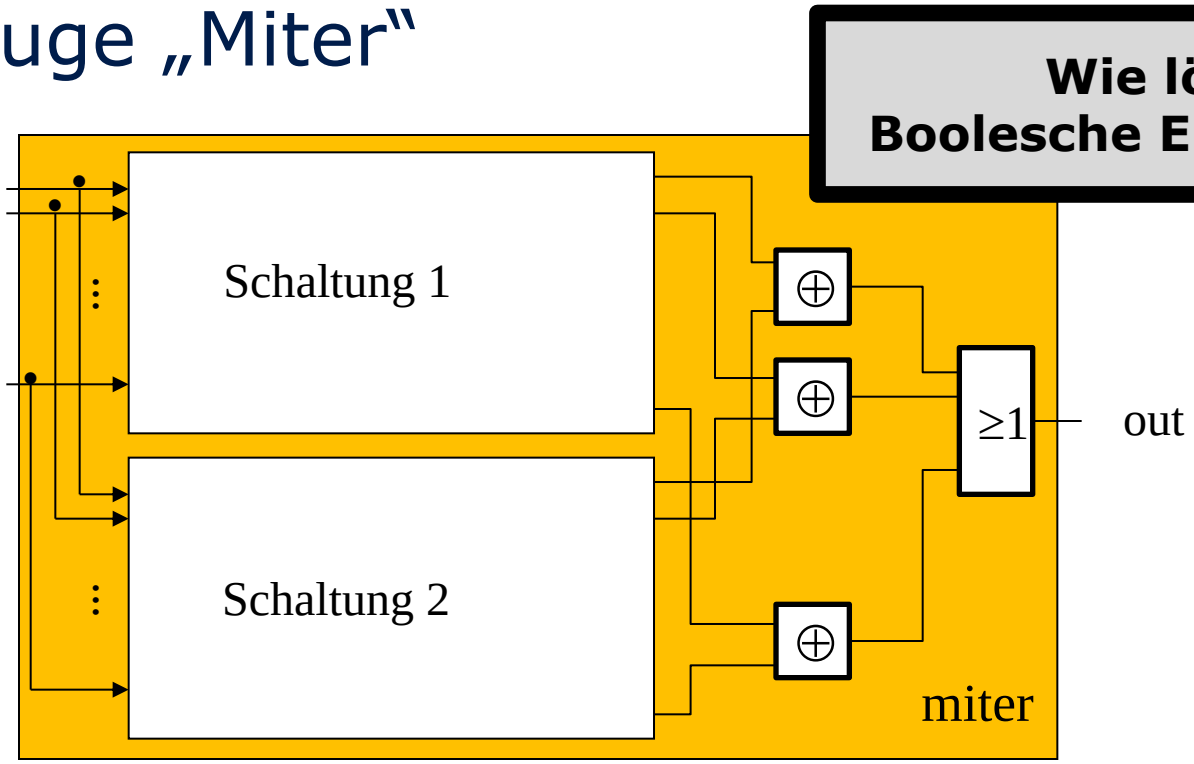


Vorgehen

- Transformation der Schaltnetze in eine Normalformdarstellung (z.B. KNF, BDDs,...)
- Liegt Gleichheit der Normalformen vor?

Alternative für die Äquivalenzprüfung

- Erzeuge „Miter“



- Ist der Ausgang für alle Eingangsbelegungen „false“?

Erfüllbarkeitsproblem (SAT)

- **SAT-Problem (Erfüllbarkeitsproblem):**
  Für eine gegebene Boolesche Funktion f finde eine Belegung a, so dass f(a)=1 oder zeige, dass keine solche Belegung existiert

- SAT-Beweiser:
  Algorithmus zum Lösen einer SAT Instanz

Erfüllbarkeitsproblem (SAT)

- SAT-Instanz repräsentiert in **konjunktiver Normalform**
  - Literal: Variable oder ihre Negation
  - Klausel: Disjunktion von Literalen
  - KNF: Konjunktion von Klauseln

$$(a+b+\overline{c})\cdot(\overline{a}+c)\cdot(\overline{b}+c)$$

- SAT ist NP-vollständig

- SAT-Beweiser können viele praktisch relevante Probleme lösen

Anwendung

Reales Problem

Reale Lösung

$$(\overline{a}+b+\overline{c})\cdot(\overline{a}+c)\cdot(\overline{b}+c)\cdot(c+\overline{d})\cdot(\overline{c}+d)\cdot(\overline{d}+\overline{e}+f)\cdot(d+\overline{f})\cdot(e+\overline{f})\cdot f$$

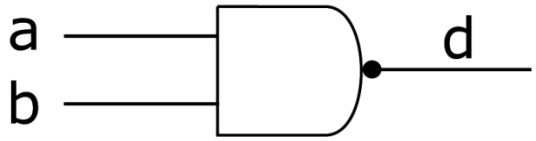
KNF

SAT Beweiser

$$a=0, b=0, c=0, d=1, e=1, f=1$$

SAT Lösung

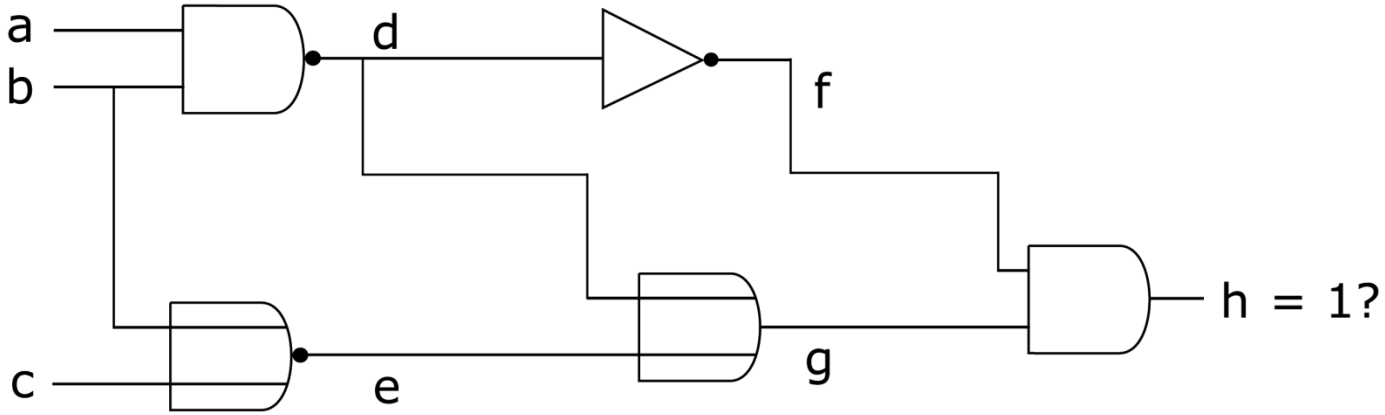
KNF eines Gatters



$$\varphi_d = [d = \neg(a\ b)]$$

$$= \neg[d \oplus \neg(a\ b)]$$
$$= \neg[\neg(a\ b)\neg d + a\ b\ d]$$
$$= \neg[\neg a\ \neg d + \neg b\ \neg d + a\ b\ d]$$
$$= (a + d)(b + d)(\neg a + \neg b + \neg d)$$

KNF eines Schaltkreises



$\varphi = h\,[d = (ab)]\,[e = \neg(b + c)]\,[f = \neg d]\,[g = d + e]\,[h = fg]$
$= h$
$(a + d)(b + d)(\neg a + \neg b + \neg d)$
$(\neg b + \neg e)(\neg c + \neg e)(b + c + e)$
$(\neg d + \neg f)(d + f)$
$(\neg d + g)(\neg e + g)(d + e + \neg g)$
$(f + \neg h)(g + \neg h)(\neg f + \neg g + h)$

- KNF für Schaltkreis mit h=1

Model Checking

- Gegeben:
  - HW-Implementierung $Impl$
  - Spezifikation $Spec$ eines Entwurfs (Eigenschaften)

- Verifikationsaufgaben:
  - Beweis, dass $Impl$ der $Spec$ entspricht
  - Notation: $Impl \rightarrow Spec$ oder $Impl \models Spec$

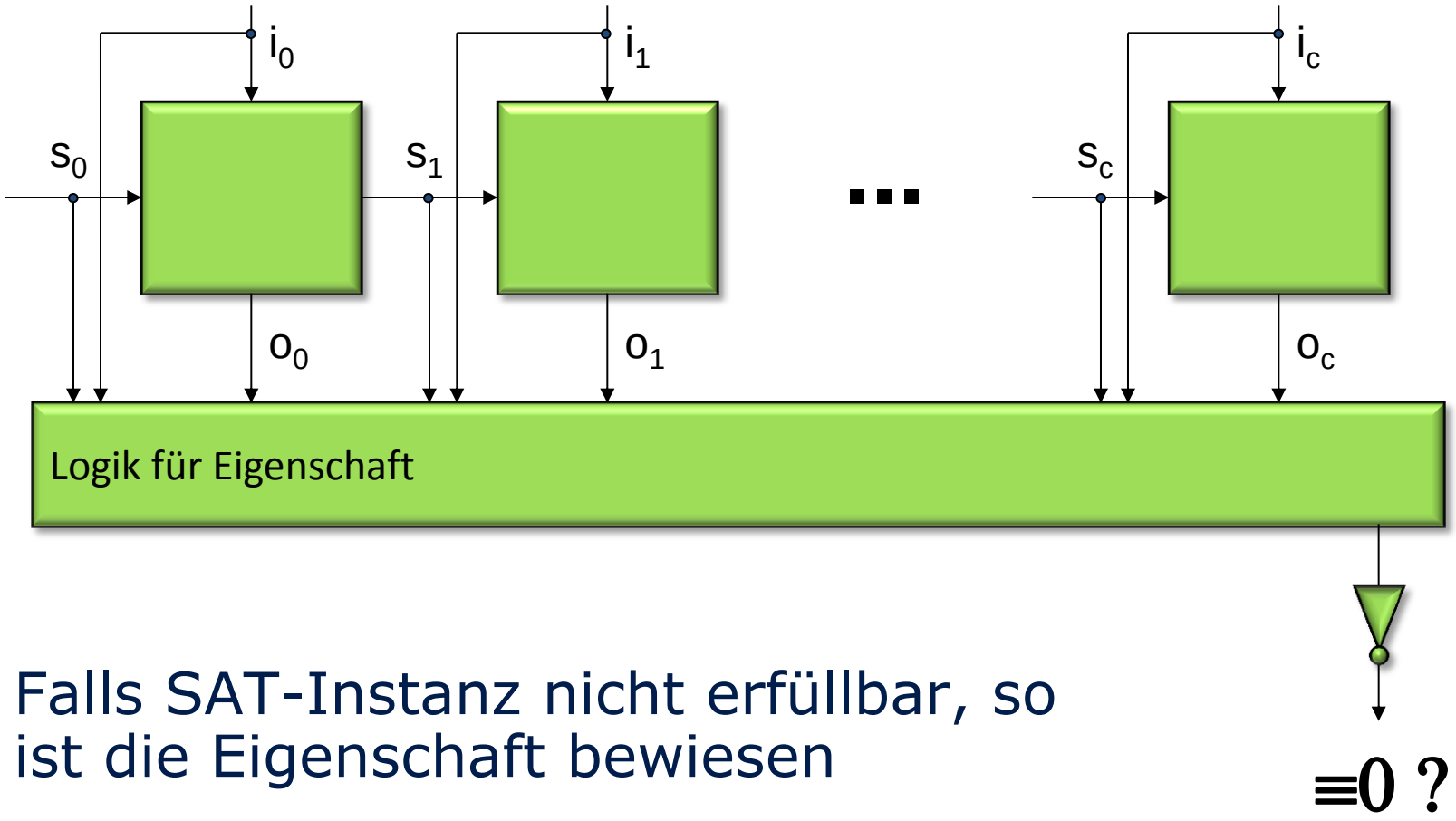
Bounded Model Checking (BMC)

- Eigenschaft argumentiert über ein endliches Zeitintervall [0,c]
- Abrollen des Modells:

$$\bigwedge_{i=0}^{c-1} T_\delta(s_i, s_{i+1}) \ \wedge \neg\, p$$

- Keine Einschränkungen für den Zustand s_0
- Auftreten von „false negatives“, Vermeidung durch zusätzliche Annahmen

Abrollen



Falls SAT-Instanz nicht erfüllbar, so ist die Eigenschaft bewiesen

Property Specification Language

- **P**roperty **S**pecification **L**anguage
- **PSL** – Formalismus zur Verhaltensbeschreibung über die Zeit
- Seit 2005 IEEE Standard (IEEE1850)
- Wenn *Annahmen* gelten, müssen *Zusicherungen* auch gelten

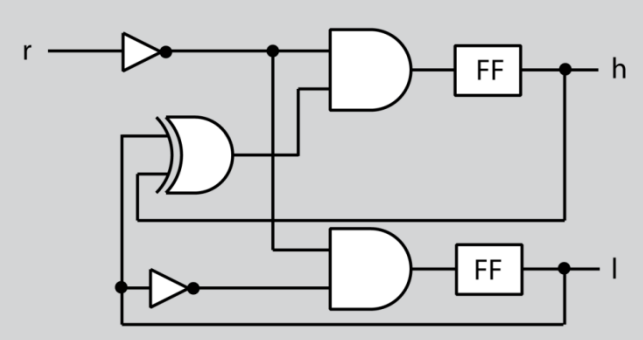
```
property test =
always(
   // Annahme
   x == 1
) -> (
   // Zusicherung
   next[2](y == 2)
);
```

Eigenschaften

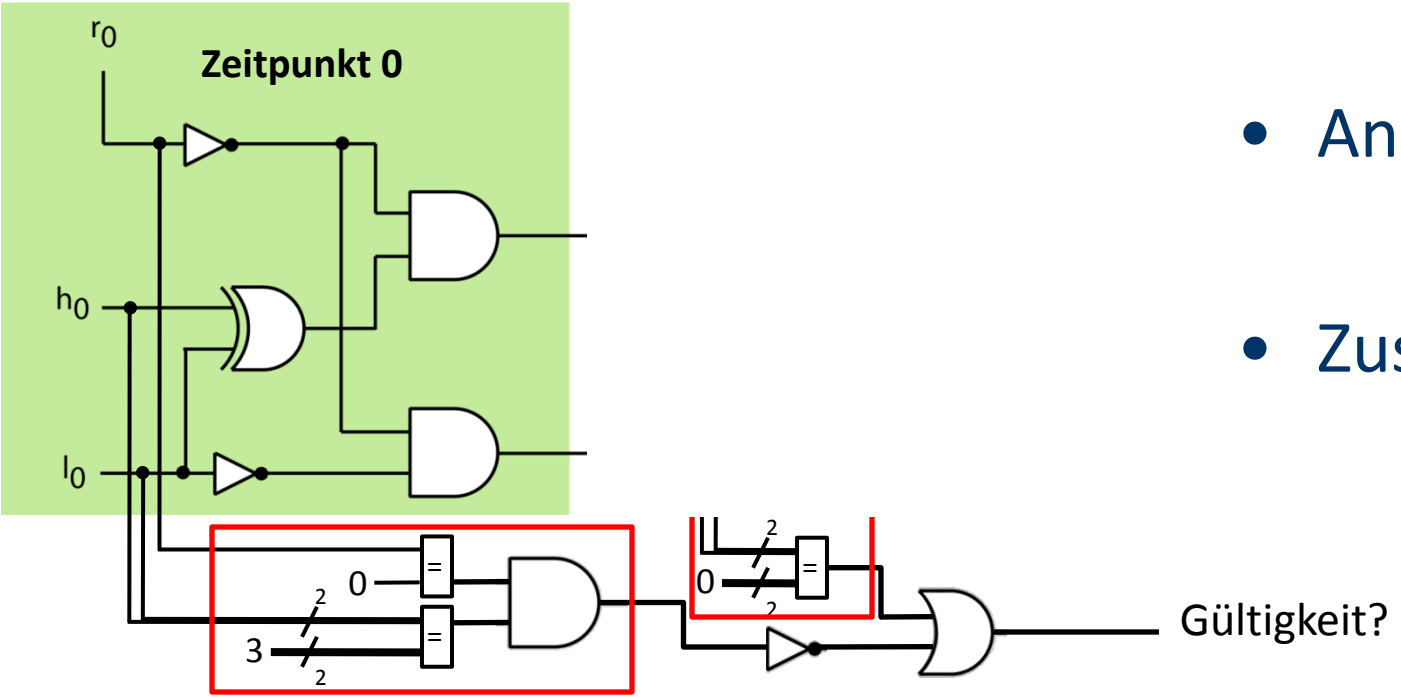
- Eigenschaft: $A \Rightarrow Z$

- A/Z aufgebaut aus:
  - next[i](expr): expr gilt zum Zeitpunkt i
  - next_e[i..j](expr): expr gilt **irgendwann** im Zeitintervall i..j
  - next_a[i..j](expr): expr gilt **immer** im Zeitintervall i..j

Beispiel: Zähler




```
property modulo =
always(
   (reset == 0) &&
   (counter == 3)
) -> (
   next[1](counter == 0)
);
```



- Annahmen
- Zusicherungen

Zusammenfassung