

boolesche Funktion

⇒ Abbildung $f : \{F, T\}^n \rightarrow \{F, T\}$

Warum sind boolesche Funktionen für die HW-Verifikation wichtig?

- Digitale Schaltungen verwenden zweiwertige boolesche Signale
- Schaltnetze lassen sich direkt durch boolesche Funktionen darstellen
- Boolesche Funktionen sind ein „Teil“ anderer Formalismen (z.B. endlicher Automaten)
- Komplexe Formalismen/Kalküle lassen sich zum Teil auf Aussagenlogik reduzieren (z.B. Prädikatenlogik: Satz von Gödel, Herbrand, Skolem [Fitt90])

Reale Chips/Schaltnetze sind oft sehr groß ($\sim 10^3$ Ein/Ausgänge, 10^3 - 10^6 Gatter)

- Auf aussagenlogischer Ebene sind Hierarchie und Abstraktionsmethoden zur Reduzierung der Problemgröße im allgemeinen nicht anwendbar.
 - ⇒ Effiziente Darstellung boolescher Funktionen nötig
 - ⇒ Darstellung muß gut für die Verifikation geeignet sein

Aufgaben

- Verknüpfung: $f := f_1 \otimes f_2 \otimes \dots \otimes f_n$
- Komposition: $f := f_{out}(f_1, \dots, f_n)$
- Gültigkeit
- Äquivalenz

hier betrachtete syntaktische Darstellungsformen

- Wertetabellen
- Aussagenlogik
- Binäre Entscheidungsdiagramme

Semantik

- *Grundbereich* (Menge der Wahrheitswerte) $\mathbb{B} = \{F, T\}$
- endliche Menge v von *Variablen* des Typs \mathbb{B}
- jeder *Belegung* der Variablen mit Wahrheitswerten aus \mathbb{B} wird ein *Ergebnis* aus \mathbb{B} zugeordnet

Eine solche Zuordnung ist gerade eine boolesche Funktion $f : \mathbb{B}^n \rightarrow \mathbb{B}$

Belegung: Abbildung $J : v \rightarrow \mathbb{B}$, die jeder Variablen x einen Wert aus \mathbb{B} zuweist

	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
J_0	F	F	F	F
J_1	F	F	T	T
J_2	F	T	F	T
J_3	F	T	T	F
J_4	T	F	F	T
J_5	T	F	T	T
J_6	T	T	F	T
J_7	T	T	T	T

- Wertetabelle mit n Variablen hat 2^n Einträge
- 2^{2^n} verschiedene Möglichkeiten, den Variablen Ergebniswerte zuzuordnen
 - ⇒ Wertetabellen für eine boolesche Funktion haben immer exponentielle Größe bezüglich der Zahl der Variablen

Abhilfe: reduzierte Werttabellen

	x_1	x_2	x_3
J_1, J_5	-	F	T
J_2, J_6	-	T	F
$J_4, J_5, J_6,$ J_7	T	-	-

aber: keine Normalform mehr!

Syntax

1. Alle atomaren Formeln aus \mathcal{V} sind Formeln.
2. Sind ϕ und ψ Formeln, so sind auch $\phi \wedge \psi$ und $\phi \vee \psi$ Formeln.
3. Ist ϕ eine Formel, so ist auch $\neg\phi$ eine Formel.

Trägermenge (Support)

Menge aller Variablen, die zur Bildung einer Formel ϕ beitragen

Interpretation val_J weist für die Belegung J einen Wahrheitswert aus \mathbb{B} wie folgt zu:

- $val_J(\varphi) := J(\varphi)$, falls $\varphi \in V$
- $val_J(\varphi \wedge \psi) := \begin{cases} T, & \text{falls } val_J(\varphi) = T \text{ und } val_J(\psi) = T \\ F, & \text{sonst} \end{cases}$
- $val_J(\varphi \vee \psi) := \begin{cases} T, & \text{falls } val_J(\varphi) = T \text{ oder } val_J(\psi) = T \\ F, & \text{sonst} \end{cases}$
- $val_J(\neg\varphi) := \begin{cases} T, & \text{falls } val_J(\varphi) = F \\ F, & \text{sonst} \end{cases}$

Satz:

Aussagenlogik ist funktional Vollständig!

Modell

Gilt für eine Formel φ und eine Belegung J die Gleichung $val_J(\varphi) = \mathbf{T}$, so gilt φ unter der Belegung J oder J ist ein Modell für φ (geschrieben als $J \models \varphi$).

Erfüllbarkeit

Eine Formel φ heißt erfüllbar, falls φ mindestens ein Modell besitzt

Eine Formel φ heißt gültig (oder Tautologie), falls jede Belegung J ein Modell für φ ist, geschrieben als $\models \varphi$.

Folgerung

Eine Formel φ folgt aus einer Menge von Formeln Ψ ($\Psi \models \varphi$) falls gilt $J \models \varphi$ für jede Belegung J , für die gilt $J \models \psi$ für alle $\psi \in \Psi$

Satz

Gegeben eine Menge V von Variablen sowie eine Formel φ der Aussagenlogik. Der Nachweis, ob φ erfüllbar ist, ist ein NP-vollständiges Problem. Der Nachweis, ob φ eine Tautologie ist, ist co-NP-vollständig

Folgerungen aus der NP-Vollständigkeit

Problem ist NP-vollständig oder co-NP-vollständig

- ⇒ Alle bekannten Verfahren benötigen eine Rechenzeit, die (im schlimmsten Fall) exponentiell mit der Zahl der Variablen steigt.
- ⇒ **Suche nach effizienten Darstellungsformen und Algorithmen, die sich „in der Regel“ nicht exponentiell verhalten**
- Viele für die Hardware-Verifikation wichtige Funktionen führen zu exponentiellem Speicherplatzbedarf/Laufzeit bezüglich der Zahl der verwendeten Variablen

Definition (Kofaktor)

Der Ausdruck $f(x_1, \dots, x_i := \mathbf{T}, \dots, x_n)$ ist der *Kofaktor von f nach x_i* bzw. die *Entwicklung von f nach x_i* (geschrieben $f|_{x_i}$),

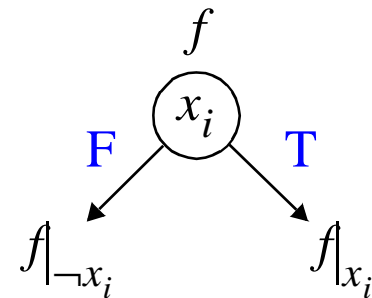
der Ausdruck $f(x_1, \dots, x_i := \mathbf{F}, \dots, x_n)$ ist der *Kofaktor von f nach $\neg x_i$* (geschrieben $f|_{\neg x_i}$).

Shannonscher Entwicklungssatz

$$f(x_1, \dots, x_i, \dots, x_n) = x_i f|_{x_i} \vee \neg x_i f|_{\neg x_i}$$

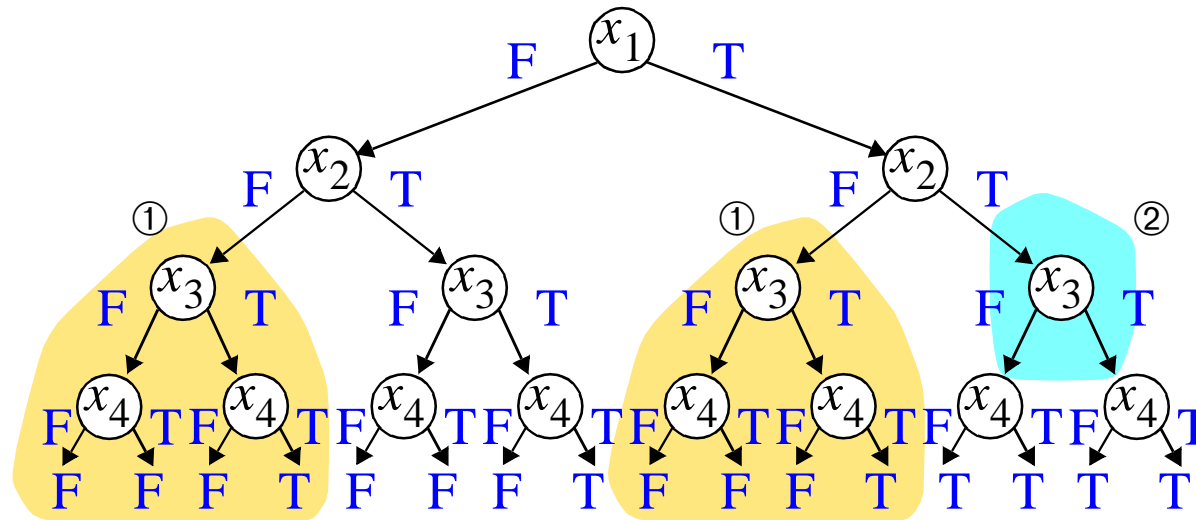
Schreibvereinbarung: $f|_{x_1 \neg x_2 x_3} := \left((f|_{x_1})|_{\neg x_2} \right)|_{x_3}$

Zurückführung einer Funktion f mit n Variablen auf zwei neue Funktionen, die Kofaktoren, mit jeweils höchstens $n-1$ Variablen
⇒ Teilweises Ausrechnen der Funktion



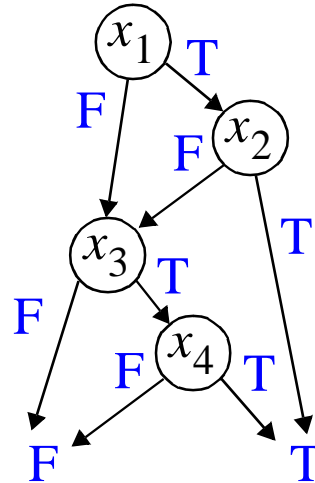
$$f = x_i f|_{x_i} \vee \neg x_i f|_{\neg x_i}$$

$$x_1x_2 \vee x_3x_4$$



Entfernen redundanter Teile

- identische Teilbäume (z.B. ①)
- Entscheidungsknoten, die nicht benötigt werden, da beide Unterbäume identisch sind (z.B. ②)



Darstellung ist sogar Normalform, falls Variablen feste Reihenfolge haben!

⇒ Reduced Ordered Binary Decision Diagram (ROBDD)

gerichteter zyklensfreier Graph $G := (V, E)$.

- Knoten $v \in V$
 - innerer Knoten:
 - zwei Söhne $\text{links} : V \rightarrow V$, $\text{rechts} : V \rightarrow V$
 - $\text{Index} : V \rightarrow \{1, \dots, n\}$
 - Blatt
 - zugeordneten Wert $\text{val}(v) : V \rightarrow \mathbb{B}$
 - keine Söhne
- Kanten $e \in E$
 - alle Paare $(v, \text{links}(v))$ und $(v, \text{rechts}(v))$.

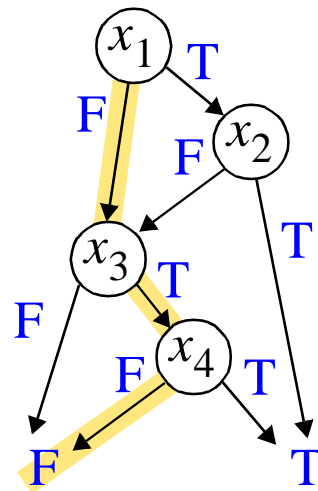
Der Graph ist geordnet, d.h. es gilt $\text{Index}(v) < \text{Index}(\text{links}(v))$ sowie $\text{Index}(v) < \text{Index}(\text{rechts}(v))$.

Größe eines ROBDDs: Zahl der Knoten, geschrieben $|G|$

ROBDD mit Wurzel $v \in V$ definiert rekursiv eine boolesche Funktion f :

$$f(v) := \begin{cases} \text{F, falls } v \text{ ein Blatt ist mit } \text{val}(v) = \text{F} \\ \text{T, falls } v \text{ ein Blatt ist mit } \text{val}(v) = \text{T} \\ \neg \text{var}(\text{Index}(v)) \wedge f(\text{links}(v)) \vee \text{var}(\text{Index}(v)) \wedge f(\text{rechts}(v)) \quad , \text{ sonst} \end{cases}$$

wobei $\text{var} : \{1, \dots, n\} \rightarrow V$ eine Abbildung der Indexzahlen auf die entsprechenden Variablen von v ist



**Auswerten der Funktion
unter der Belegung J**

$$J(x_1) = \text{F}$$

$$J(x_2) = \text{T}$$

$$J(x_3) = \text{T}$$

$$J(x_4) = \text{F}$$

Isomorphie

Zwei ROBDDs $G := (V, E)$ und $G' := (V', E')$ heißen *isomorph* genau dann, wenn es eine bijektive Funktion $\text{map} : V \rightarrow V'$ gibt, so daß gilt

- für alle Blätter $\text{val}(v) = \text{val}(\text{map}(v))$
- für innere Knoten $\text{Index}(v) = \text{Index}(\text{map}(v))$, sowie
 $\text{map}(\text{links}(v)) = \text{links}(\text{map}(v))$ und $\text{map}(\text{rechts}(v)) = \text{rechts}(\text{map}(v))$

reduzierter ROBDD

Ein ROBDD heißt reduziert genau dann, wenn kein Knoten existiert, für den gilt $\text{links}(v) = \text{rechts}(v)$ sowie keine zwei Knoten v und v' existieren, so daß die Teilbäume mit den Wurzeln v und v' isomorph sind

Satz

Für jede Boolesche Funktion f und eine vorgegebene Variablenordnung gibt es einen – bis auf Isomorphie – eindeutigen ROBDD

Algorithmus zur Reduktion

```
1 fasse Blätter mit gleichen Werten zusammen;  
2 for  $i := n$  downto 1 do  
3   for alle Knoten  $v$  mit  $\text{Index}(v) = i$  do  
4     if  $\text{links}(v) = \text{rechts}(v)$  then lösche  $v$ ;  
5     if  $\exists v' . (\text{Index}(v') = i$  und  
6        $\text{links}(v') = \text{links}(v)$  und  $\text{rechts}(v') = \text{rechts}(v))$   
7     then ersetze  $v$  durch  $v'$ ;
```

Satz (Distributivität)

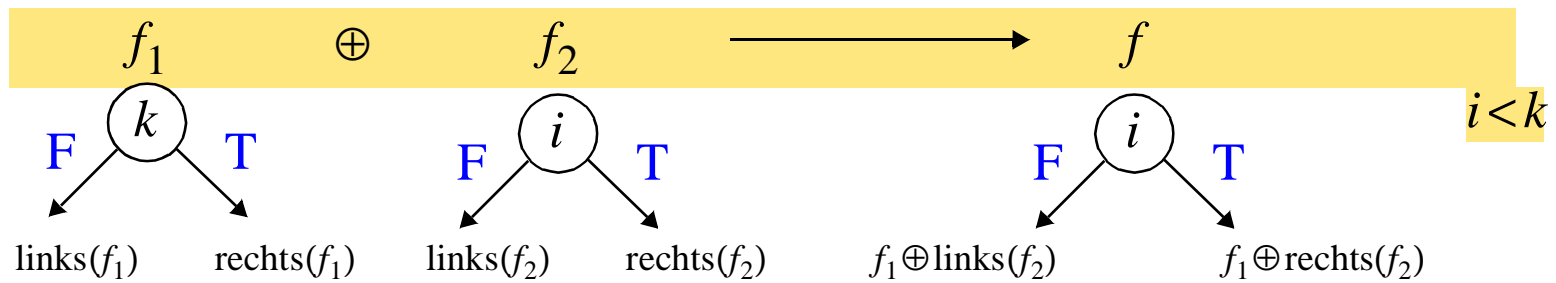
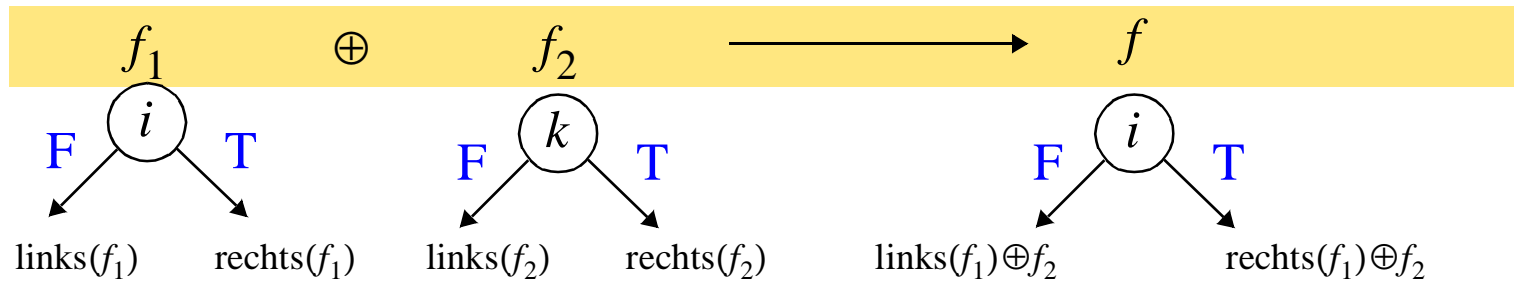
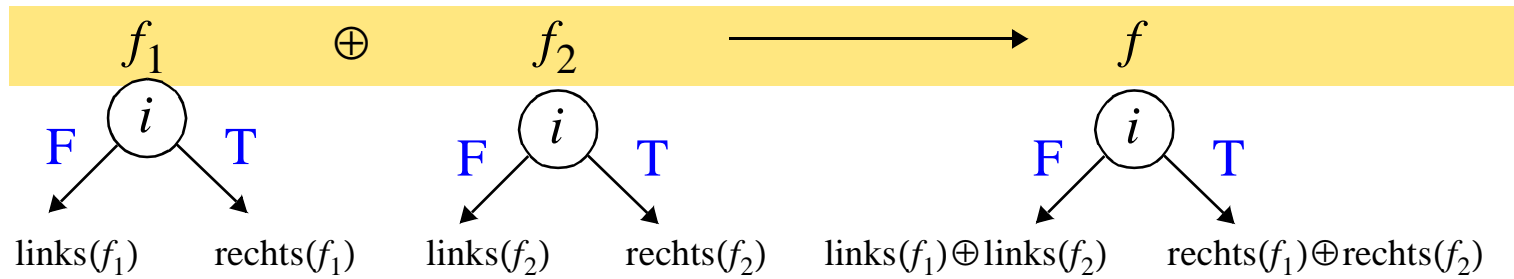
$$(f \oplus g)|_{x_i} = f|_{x_i} \oplus g|_{x_i}$$

\oplus beliebiger zweistelliger Boolescher Operator

Das bedeutet, aus $f = f_1 \oplus f_2$ folgt:

$$\begin{aligned} f &= x_i f|_{x_i} \vee \neg x_i f|_{\neg x_i} \\ &= x_i (f_1 \oplus f_2)|_{x_i} \vee \neg x_i (f_1 \oplus f_2)|_{\neg x_i} \\ &= x_i \left(f_1|_{x_i} \oplus f_2|_{x_i} \right) \vee \neg x_i \left(f_1|_{\neg x_i} \oplus f_2|_{\neg x_i} \right) \end{aligned}$$

\Rightarrow Die Operation \oplus kann auf die Söhne von x_i verlagert werden



Aufwand für die Verknüpfung: $O(2^n)$ (n ist maximaler Index)

⇒ Einsatz spezieller Datenstrukturen resultiert in einem Aufwand
 $O(|V_1||V_2|)$

1. unique Table

Die unique table ist eine Hashtabelle, in der alle Knoten mit ihren Nachfolgern abgelegt werden

$(Index(v), inks(v), rechts(v), bdd)$

Sie dient zum schnellen Auffinden von isomorphen Teilgraphen

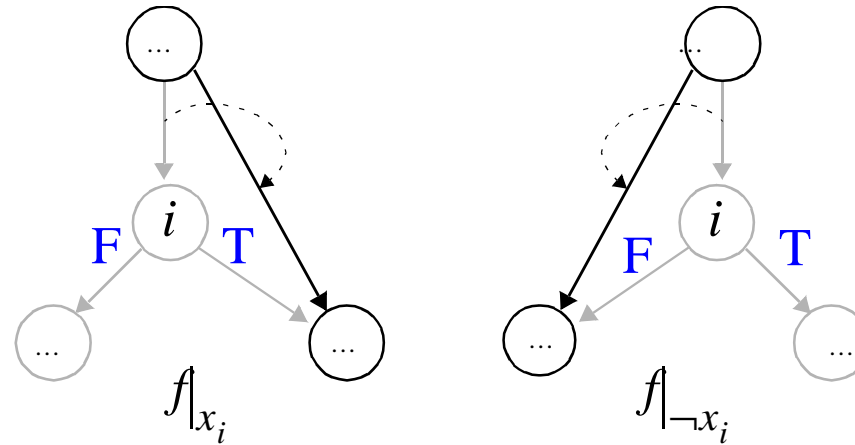
⇒ Isomoprphieprüfung bei idealer Hashtabelle in $O(1)$

2. computed table

Die computed table ist eine Hashtabelle in der bereits berechnete (Zwischen-) Ergebnisse abgelegt werden:

$(operation, op_1, op_2, result)$

```
1 bdd apply(operation op, bdd f, bdd g)
2
3   bdd res, tmp1, tmp2
4   index var
5   res = computedTable.lookup(op,f,g)
6
7   if (res == NULL) then
8     var = topIndex(f,g)
9     tmp1 = apply(op, posCofact(var,f), posCofact(var,g))
10    tmp2 = apply(op, negCofact(var,f), negCofact(var,g))
11
12    res = uniqueTable.lookup(var, tmp1, tmp2)
13    if (res == NULL) then
14      res = ite(var,tmp1,tmp2)
15      uniqueTable.insert(var,tmp1,tmp2,res)
16
17    computedTable.insert(op,f,g,res)
18
19  return res
```



Komposition

$$f_2|_{x_i:=f_1} = \left(f_1 \wedge f_2|_{x_i} \right) \vee \left(\neg f_1 \wedge f_2|_{\neg x_i} \right)$$

Aufwand $O(|G_1| \cdot |G_2|^2)$ bei spezieller Implementierung

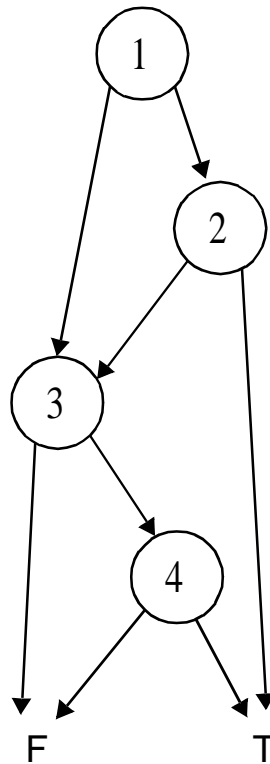
Variablenordnung 1

$\text{var}_1(1) := x_1$

$\text{var}_1(2) := x_2$

$\text{var}_1(3) := x_3$

$\text{var}_1(4) := x_4$



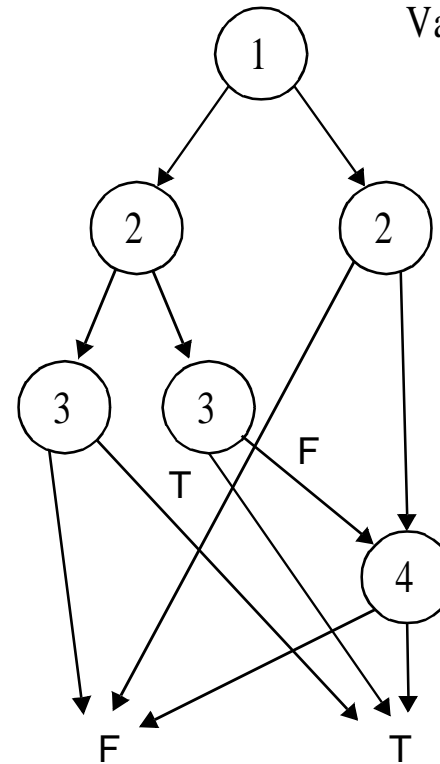
Variablenordnung 2

$\text{var}_2(1) := x_1$

$\text{var}_2(2) := x_3$

$\text{var}_2(3) := x_2$

$\text{var}_2(4) := x_4$



Satz

Das Finden der optimalen Variablenordnung ist ein NP-vollständiges Problem.

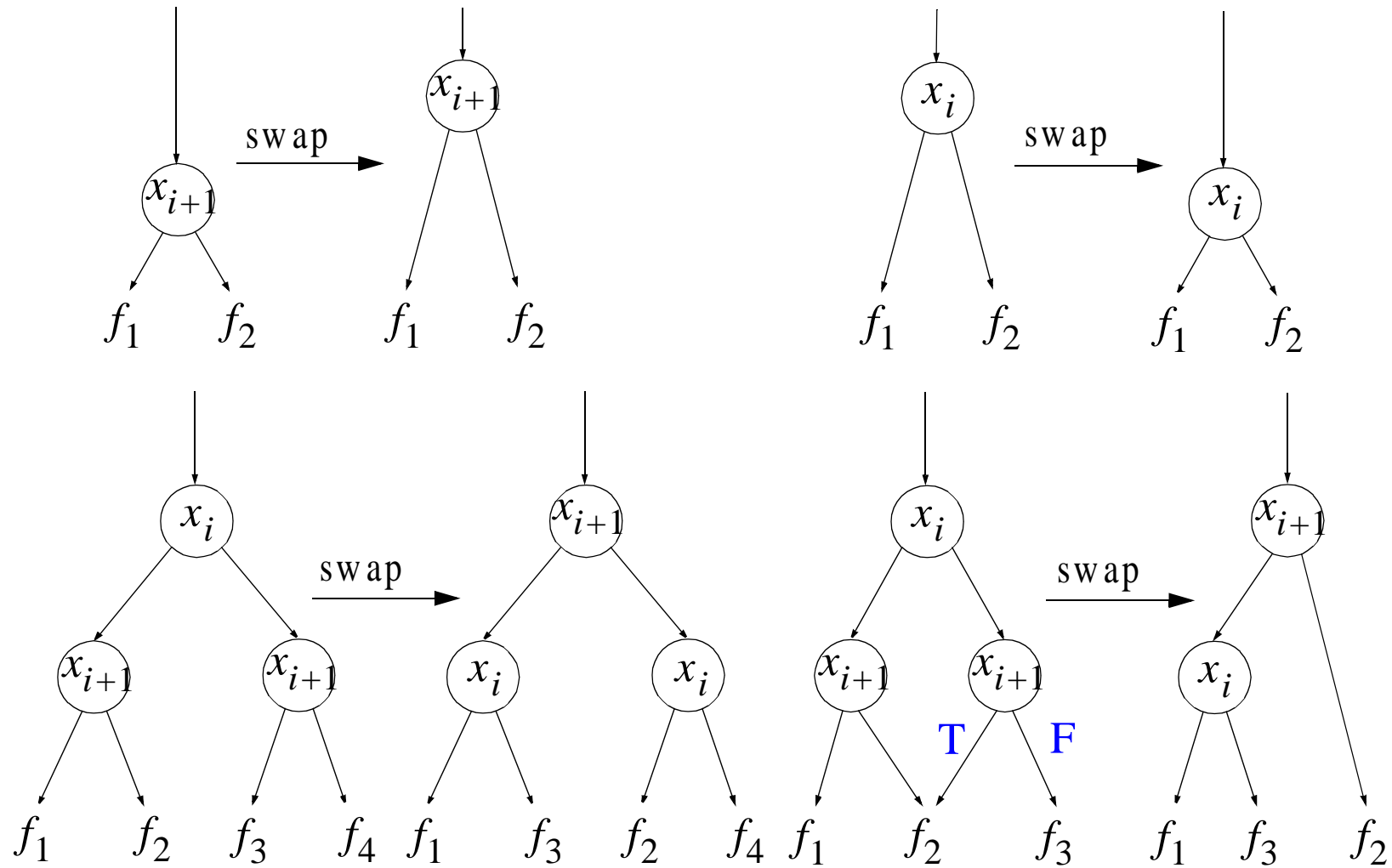
⇒ Heuristische Verfahren

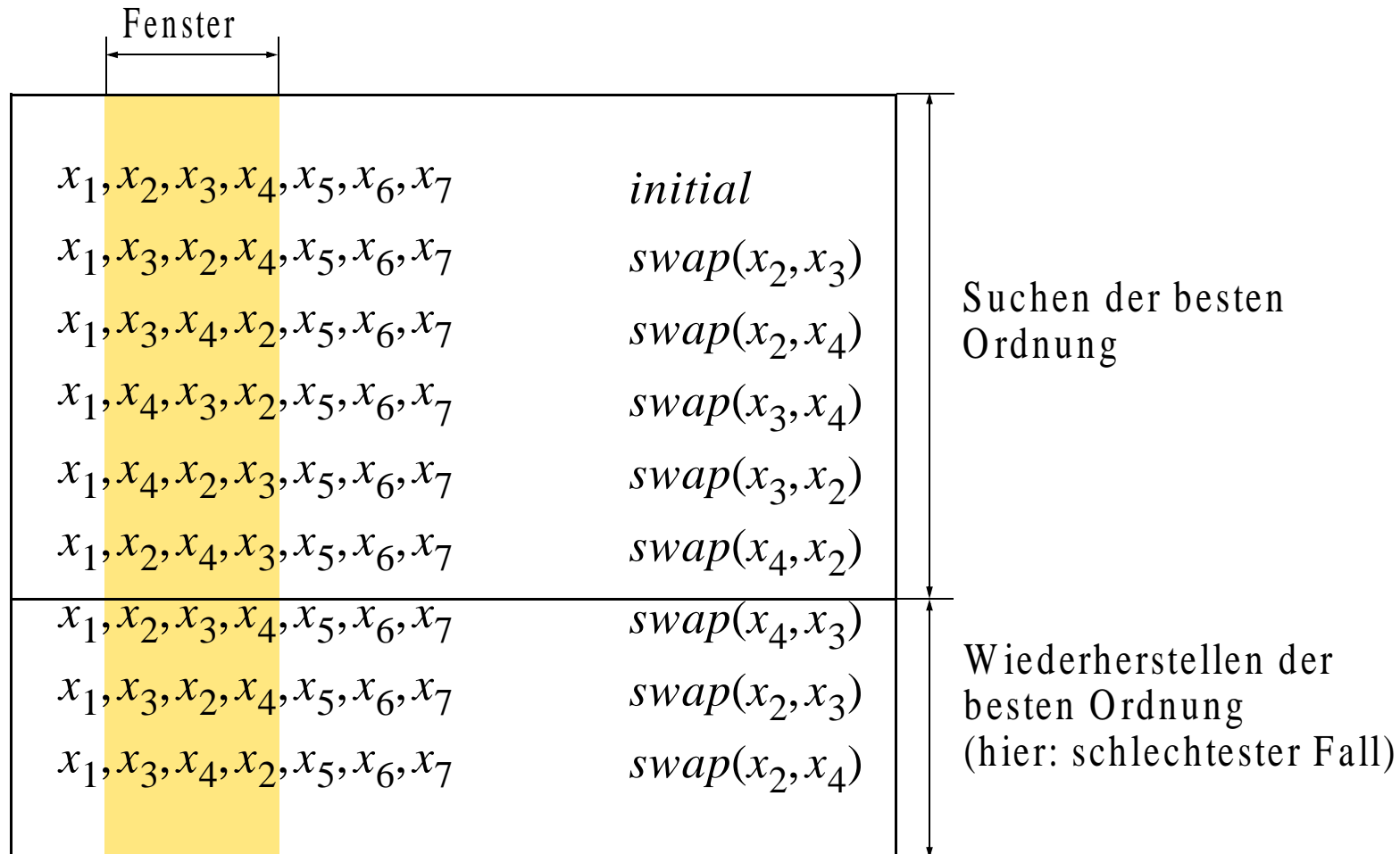
statische Verfahren

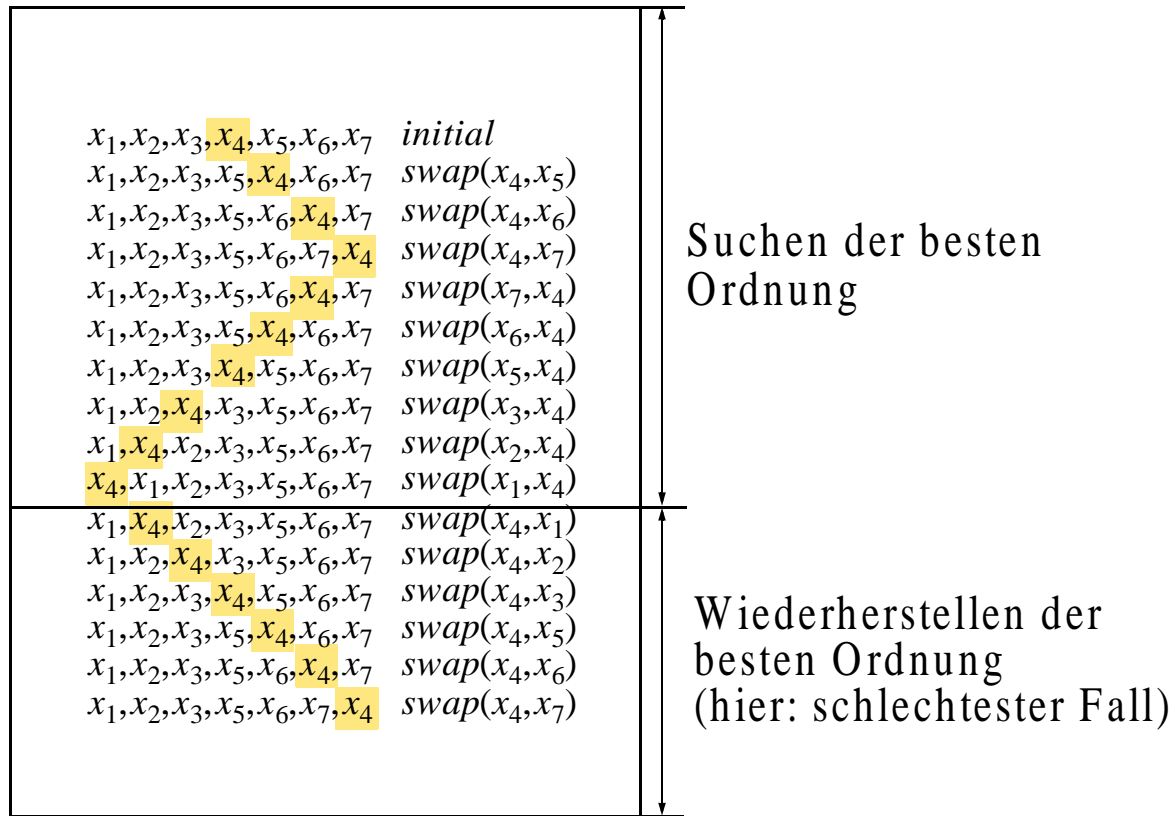
Bestimmung aus Schaltnetzstruktur vor dem ROBDD-Aufbau

dynamische Verfahren

Umordnung des ROBDDs bei Größenüberschreitung





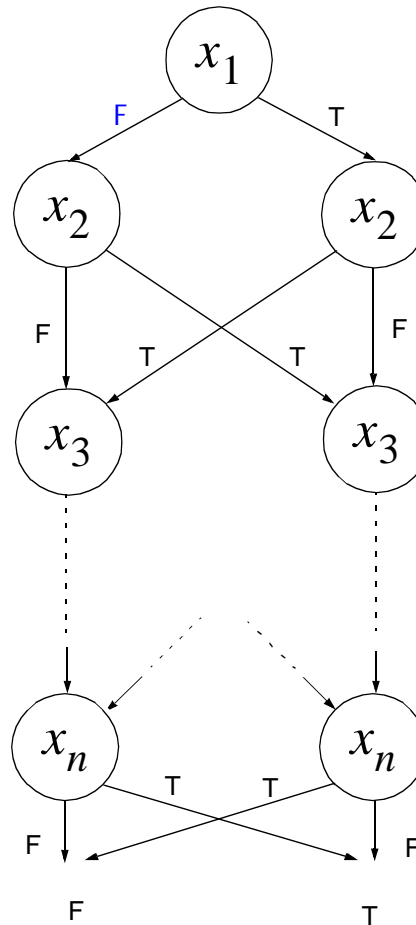


- *Bösartige Funktionen*
 - Alle Ordnungen führen zu nicht polynomieller Größe.
- *Gutartige Funktionen*
 - Alle Ordnungen führen zu polynomieller Größe.
- *Fast gutartige Funktionen*
 - Es gibt für jedes n mindestens eine Ordnung, die zu nicht polynomieller Größe führt, aber der Anteil der Ordnungen mit polynomieller Größe konvergiert gegen 1.
- *Unklare (ambigouse) Funktionen*
 - Es gibt Ordnungen, die zu polynomieller und zu exponentieller Größe führen, kein Anteil konvergiert jedoch gegen 1.
- *Fast böartige Funktionen*
 - Es gibt für jedes n mindestens eine Ordnung, die zu polynomieller Größe führt, aber der Anteil der Ordnungen mit nicht polynomieller Größe konvergiert gegen 1.

Gutartige Funktion: „Paritätsfunktion“

2.2-26

$$f(x_1, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n$$

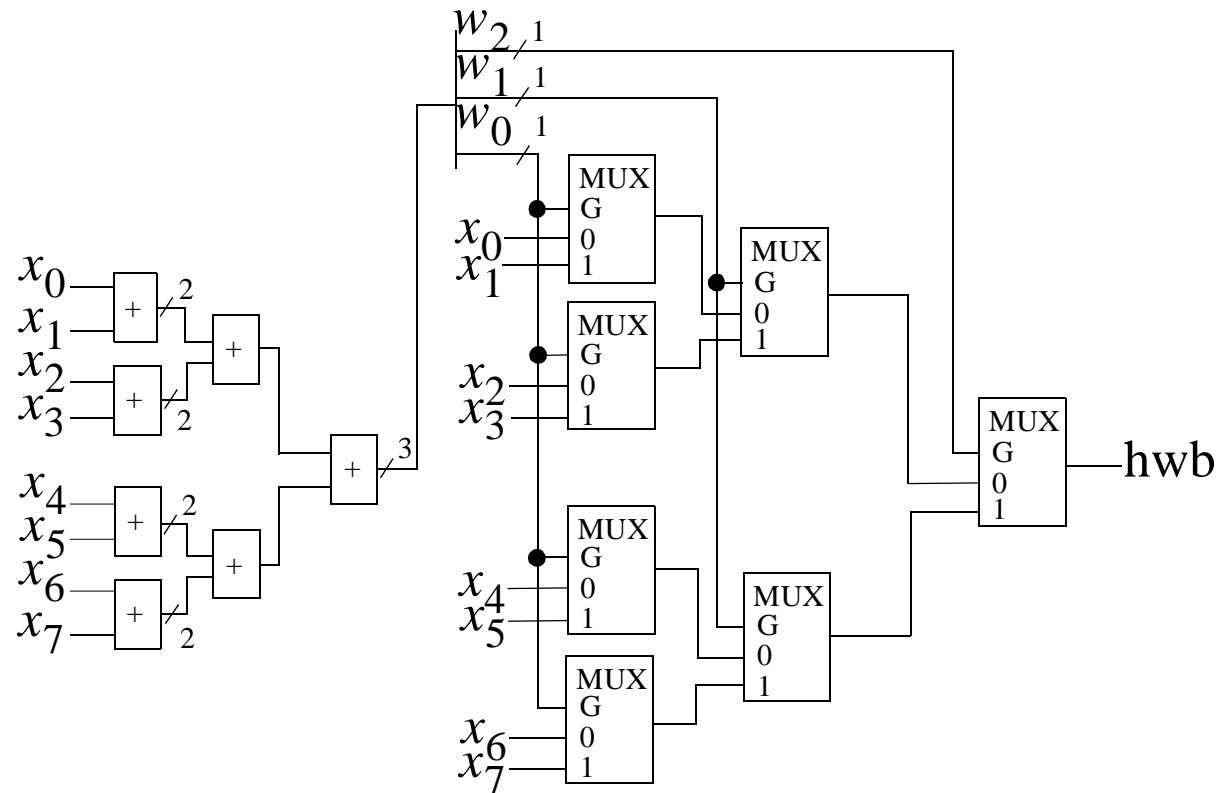


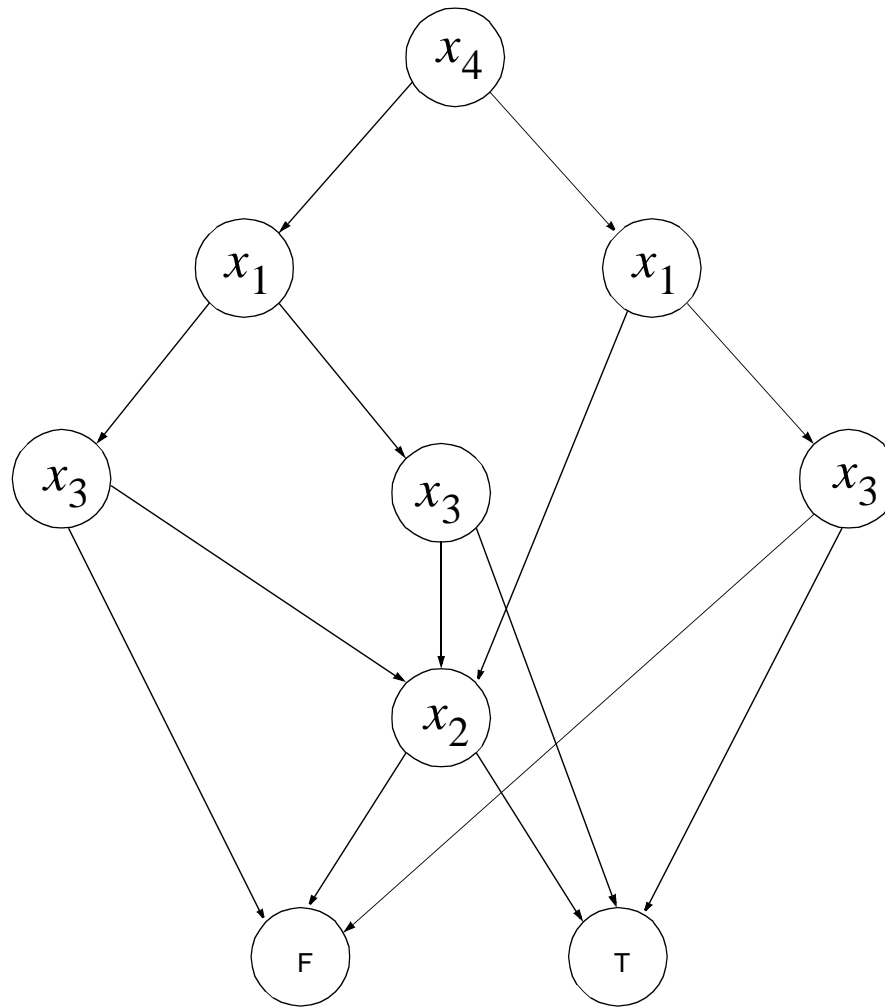
Bösartige Funktion: „hidden weighted bit“

2.2-27

hwb : ($\mathbb{B}^n \rightarrow \mathbb{B}$) sei definiert als $\text{hwb}(x_1, \dots, x_n) := x_{\text{sum}}$ mit $\text{sum} := x_1 + \dots + x_n$

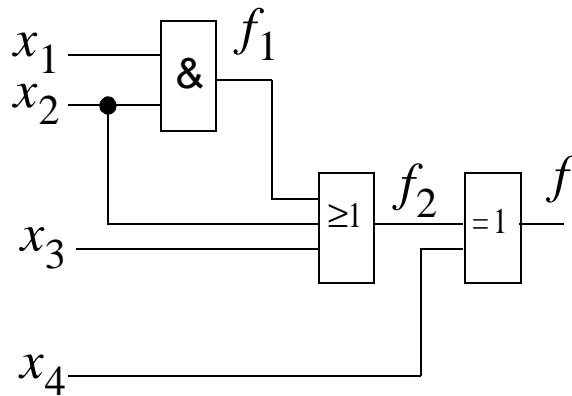
und $x_0 := 0$:





$n = 4$

Schaltnetz



$$f_1(x_1, x_2) = x_1 \wedge x_2$$

$$f_2(x_1, x_2, x_3) = f_1 \vee x_2 \vee x_3$$

$$= (x_1 \wedge x_2) \vee x_2 \vee x_3$$

$$f(x_1, x_2, x_3, x_4) = f_2 \oplus x_4$$

$$= ((x_1 \wedge x_2) \vee x_2 \vee x_3) \oplus x_4$$

Vorwärtserzeugung

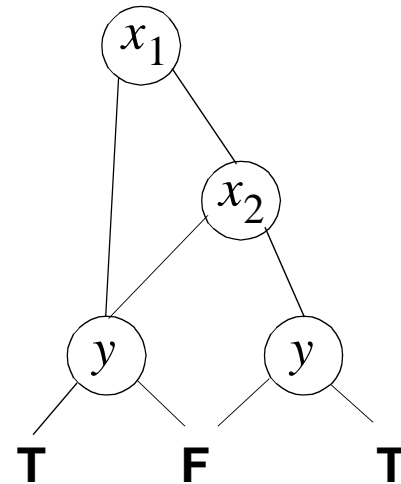
- von den Eingängen ausgehend, Boolesche Funktionen erzeugen und verknüpfen, bis der Ausgang erreicht ist

Rückwärtserzeugung

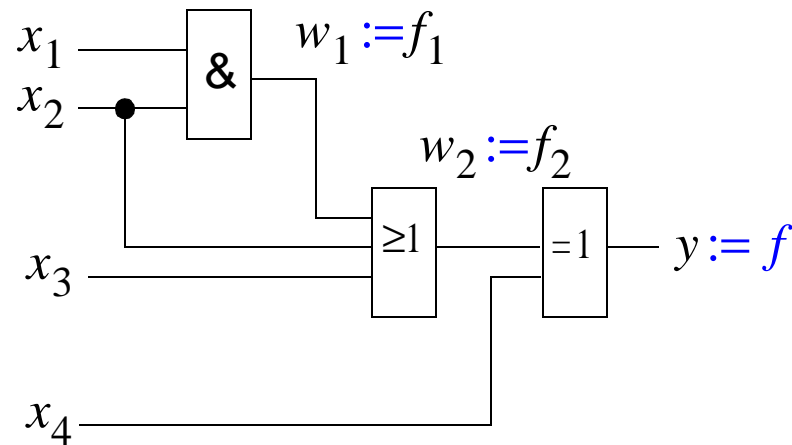
- bei dem Ausgangsgatter beginnen
 - zuerst für dieses eine Boolesche Funktion erzeugen
 - anschließend jeweils Funktionen für die unmittelbaren Vorgänger erzeugen und diese durch Komposition mit der bisher erzeugten Funktion verschmelzen
 - dies solange fortsetzen, bis die Eingänge erreicht sind

explizite Darstellung auch des Schaltnetzausgangs: $y \leftrightarrow f(x_1, \dots, x_n)$

	x_1	x_2	y	$y \leftrightarrow x_1 \wedge x_2$
J_0	F	F	F	T
J_1	F	F	T	F
J_2	F	T	F	T
J_3	F	T	T	F
J_4	T	F	F	T
J_5	T	F	T	F
J_6	T	T	F	F
J_7	T	T	T	T



Schaltnetz

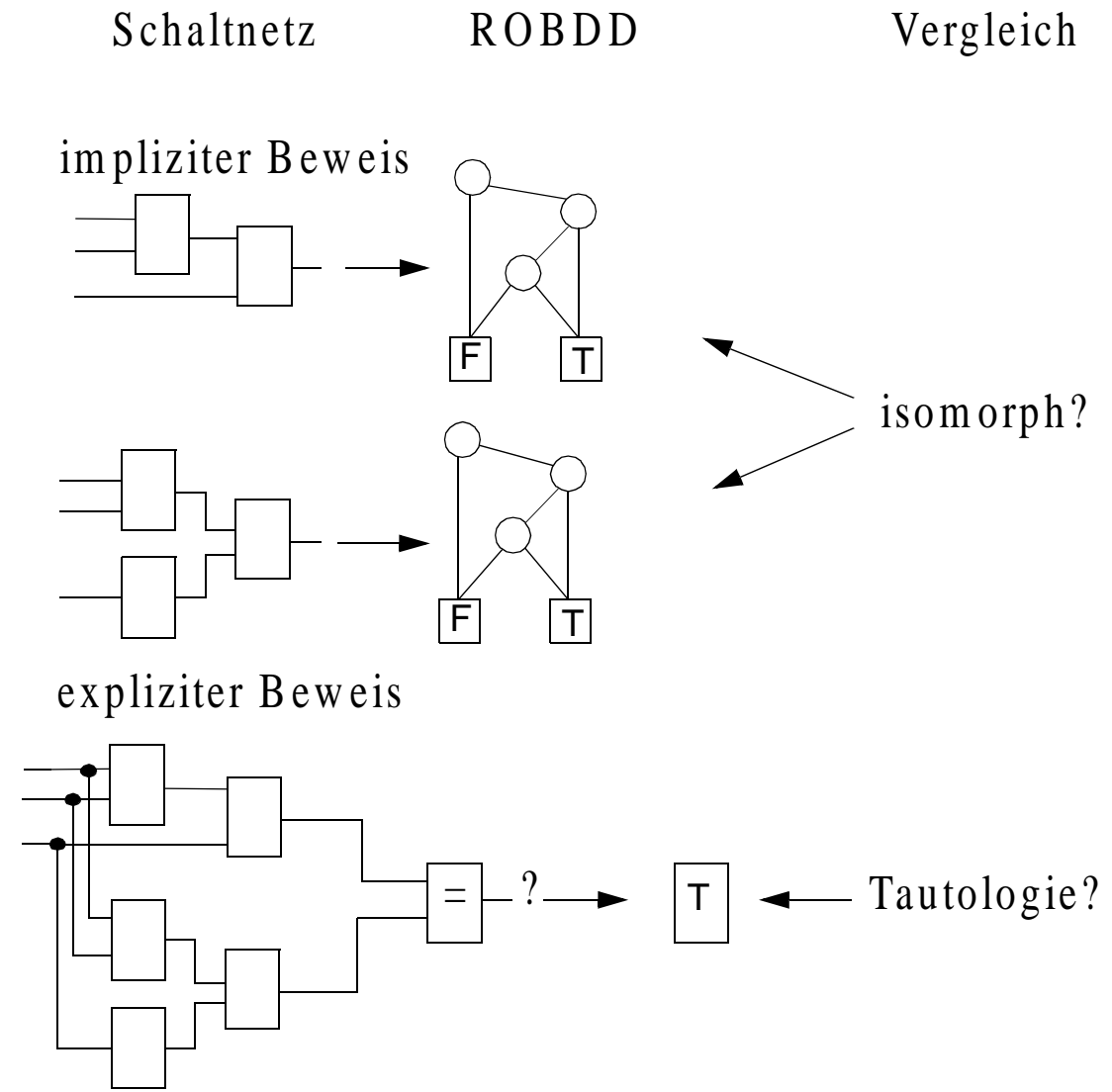


$$(w_1 \leftrightarrow (x_1 \wedge x_2)) \wedge$$

$$(w_2 \leftrightarrow (w_1 \vee x_2 \vee x_3)) \wedge$$

$$(y \leftrightarrow (w_2 \oplus x_4))$$

- ⇒ innere Signale werden explizit (als BDD-Variable) repräsentiert
- ⇒ es können mehrerer Ausgänge in einem BDD dargestellt werden



Impliziter Beweis

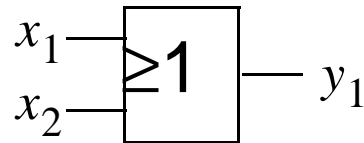
- direkter Vergleich zweier durch ROBDDs dargestellten Schaltnetze
 - funktionale Schaltnetzdarstellung (Folie 2.3-1)
 - ROBDDs haben Normalformeneigenschaft (Folie 2.2-13)
- ⇒ Nachweis der Isomorphie ausreichend

expliziter Beweis

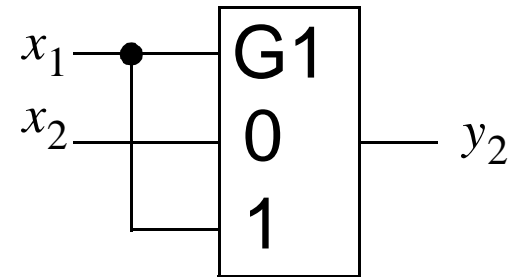
- relationale Schaltnetzdarstellung (Folie 2.3-2)
- Explizite Formulierung der Äquivalenz zweier Schaltnetze $\models I \rightarrow S$
- Gesamtimplementierung $I := I_1 \wedge I_2$
- Spezifikation der Ausgangsäquivalenz $S := (y_1 \leftrightarrow y_2)$

$$\models (I_1 \wedge I_2) \rightarrow (y_1 \leftrightarrow y_2)$$

Schaltnetz 1



Schaltnetz 2



Schaltnetz 1

$$y_1 \leftrightarrow (x_1 \vee x_2)$$

Multiplexer allgemein

$$y_2 \leftrightarrow (\neg s \wedge in_1) \vee (s \wedge in_2)$$

Schaltnetz 2

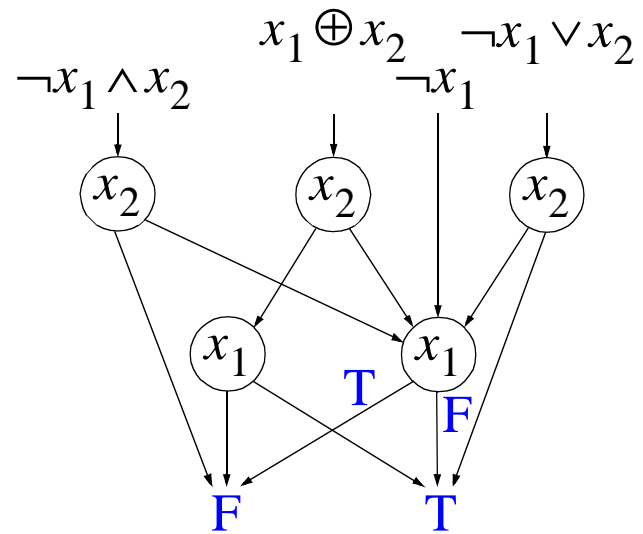
$$y_2 \leftrightarrow (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_1)$$

zu beweisendes Ziel

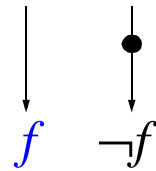
$$\models [(y_1 \leftrightarrow (x_1 \vee x_2)) \wedge (y_2 \leftrightarrow (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_1))] \rightarrow (y_1 \leftrightarrow y_2)$$

SBDD

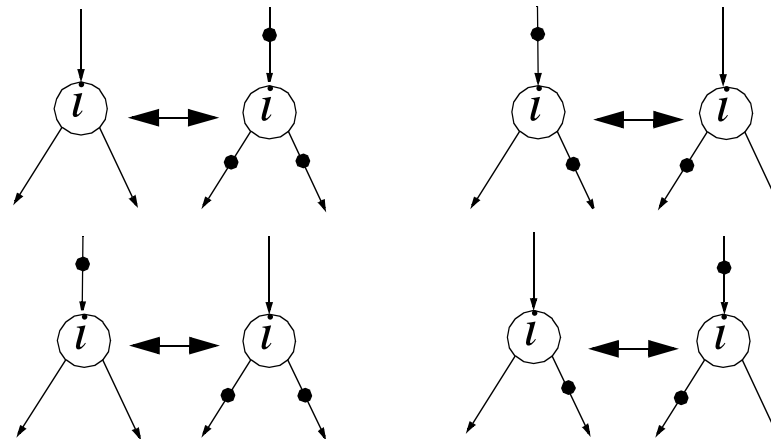
Darstellung mehrerer Boolescher Funktionen in einem Graph



Darstellung einer Booleschen Funktion und ihres Komplements



Kantentransformation zur Erhaltung der Normalformeigenschaft



Darstellungsziel

Reed-Muller Form: $f(x_1, x_2) := x_1 \oplus x_2 \oplus (x_1 \wedge x_2)$

Boolesche Differenz

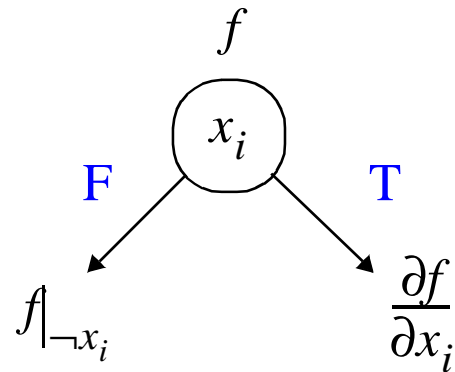
$$\frac{\partial f}{\partial x_i} := f|_{\neg x_i} \oplus f|_{x_i}$$

Davio-Entwicklung (vergleiche Folie 2.2-7)

positiv Davio-Entwicklung: $f(x_1, \dots, x_i, \dots, x_n) = f|_{\neg x_i} \oplus \left(x_i \wedge \frac{\partial f}{\partial x_i} \right)$

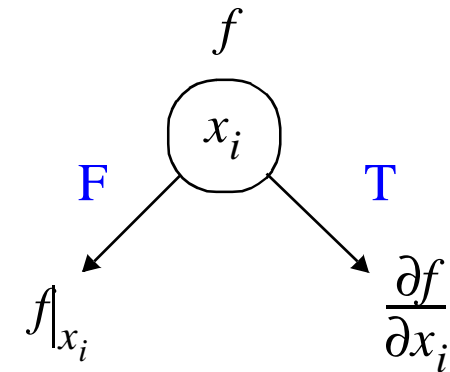
negative Davio-Entwicklung $f(x_1, \dots, x_i, \dots, x_n) = f|_{x_i} \oplus \left(\neg x_i \wedge \frac{\partial f}{\partial x_i} \right)$

positive Davio-Entwicklung
(FDD-Knoten)

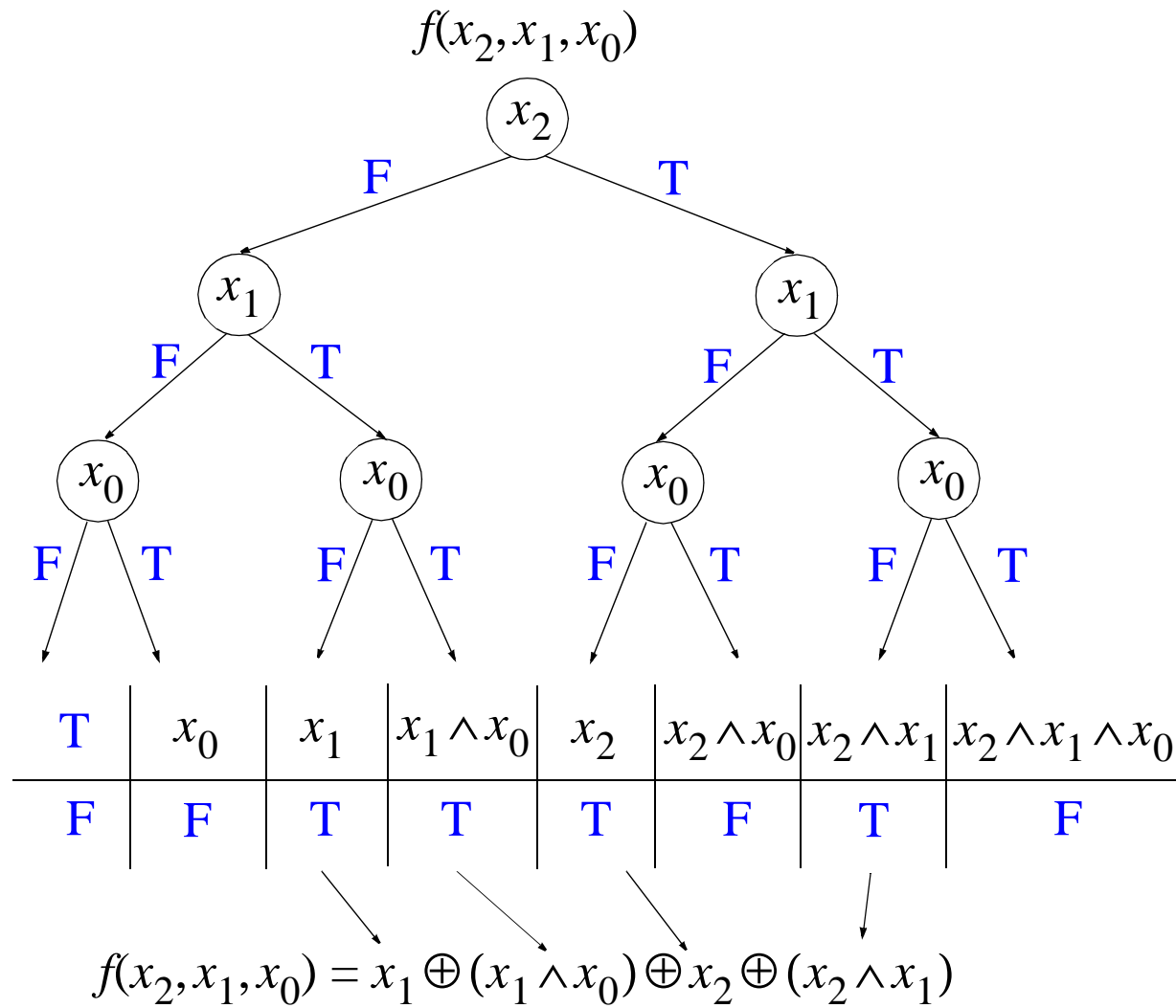


$$f := f|_{\neg x_i} \oplus \left(x_i \wedge \frac{\partial f}{\partial x_i} \right)$$

negative Davio-Entwicklung



$$f := f|_{x_i} \oplus \left(\neg x_i \wedge \frac{\partial f}{\partial x_i} \right)$$



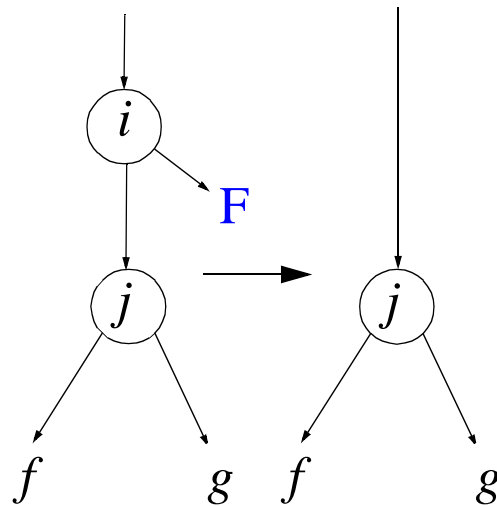
$$f := \underbrace{f|_{\neg x_i}}_{\text{linker Subgraph}} \oplus \underbrace{\left(x_i \wedge \frac{\partial f}{\partial x_i}\right)}_{\text{rechter Subgraph}}$$

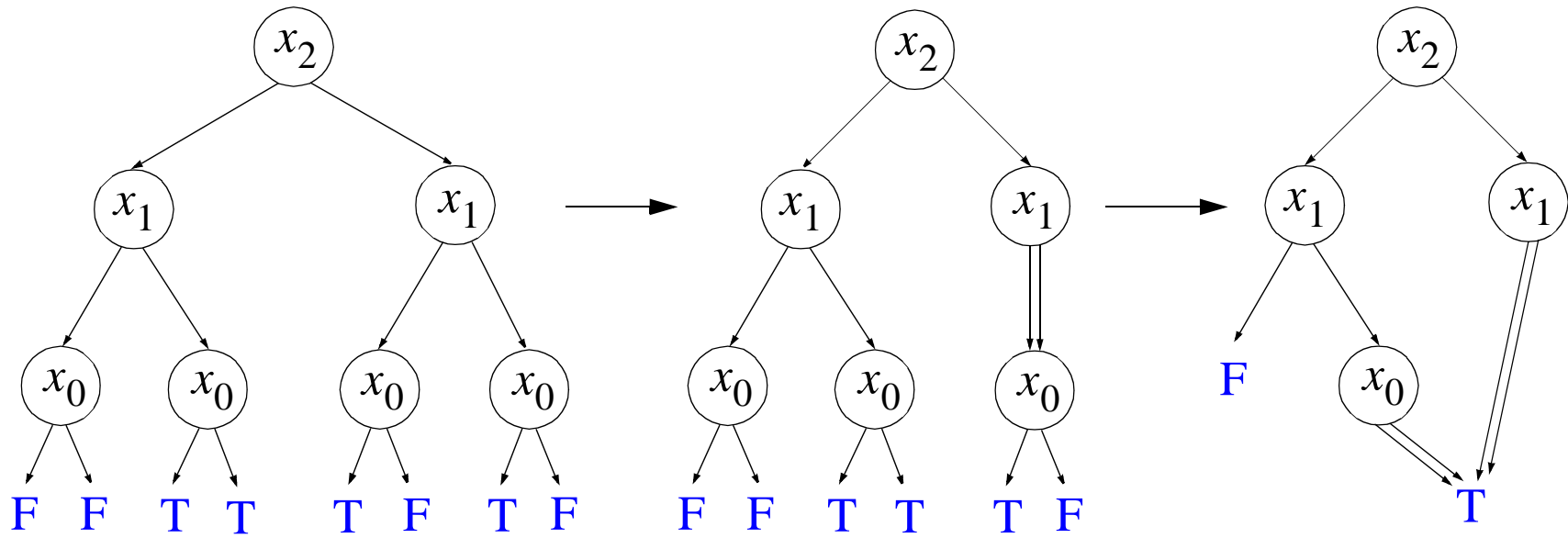
angenommen: rechter Subgraph = F

$$\Rightarrow f = f|_{\neg x_i} \oplus (x_i \wedge F)$$

$$\Rightarrow f = f|_{\neg x_i} \oplus F$$

$$\Rightarrow f = f|_{\neg x_i}$$





Definition

Es seien A und B Mengen für die gilt $B \subseteq A$. Die *charakteristische Funktion* χ_B ist definiert als

$$\chi_B(x) = \begin{cases} \mathbf{T}, & \text{falls } x \in B \\ \mathbf{F}, & \text{falls } x \notin B \end{cases}$$

Menge und Teilmengen aus \mathbb{B}^n :

$$\chi : \mathbb{B}^n \rightarrow \mathbb{B}$$

Es gilt

$$\chi_{B_1 \cap B_2} = \chi_{B_1} \wedge \chi_{B_2}$$

$$\chi_{B_1 \cup B_2} = \chi_{B_1} \vee \chi_{B_2}$$

$$\chi_{A \setminus B_1} = \neg \chi_{B_1}$$

dabei seien A eine gegebene Grundmenge sowie B_1 und B_2 Teilmengen von A

$$I \models S$$

⇒ I gibt die Belegungen an, unter denen S gelten muß (Folie 2.2-5)

⇒ Überprüfung aller Belegungen, ob jede dazu führt, daß S wahr wird:

$$I_1, I_2 \models y_1 \leftrightarrow y_2$$

Beispiel von Folie 2.4-2

Zulässige Belegungen ($J(x_1), J(x_2), J(y_1), J(y_2)$):

$$\{(F, F, F, F), (F, T, T, T), (T, F, T, T), (T, T, T, T)\}.$$

Offensichtlich gilt

$$\{(F, F, F, F), (F, T, T, T), (T, F, T, T), (T, T, T, T)\} \models y_1 \leftrightarrow y_2$$