

# **1.5 Schaltfunktionen und boolesche Ausdrücke**

**Zur Vorlesung Rechenanlagen**

**SS 2019**



# Schaltfunktionen

## Definition

Eine partielle Funktion  $f: B^n \rightarrow B^k$

heißt (allgemeine) Schaltfunktion

Wir haben schon gesehen, dass gewisse Mengen von Schaltfunktionen eine boolesche Algebra bilden:

$S_n := \mathbf{Abb}(B^n, B)$  bildet unter punktweiser Verknüpfung

$$f \vee g(p) := f(p) \vee g(p);$$

$$f \cdot g(p) := f(p) \cdot g(p); \bar{f}(p) := \overline{f(p)}$$

eine boolesche Algebra.

# Schaltfunktionen ff

Die Elemente  $f$  aus  $\mathcal{S}_{n,k} := \text{Abb}(\mathbf{B}^n, \mathbf{B}^k)$

kann man darstellen als

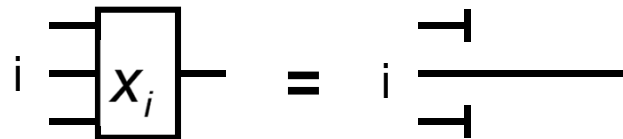
$$f(x) = (f_1(x), \dots, f_k(x)), \text{ mit } f_i \in \mathcal{S}_n$$

Also sollte unser Hauptinteresse zunächst einmal folgenden booleschen Algebren gelten:

$$\mathcal{S}_n \text{ sowie } \mathcal{S}_n^D := \{f: \mathbf{B}^n \rightarrow \mathbf{B} \mid D(f) = D\}$$

Betrachte die Schaltfunktionen  $x_i: \mathbf{B}^n \rightarrow \mathbf{B}$  mit  $x_i(p) = p_i$

Sie sind trivial zu realisieren, weil sie Projektionen auf die  $i$ -te Komponente sind, also einfach der  $i$ -te Eingang:



# Schaltfunktionen

## Satz

Jede Schaltfunktion aus  $\mathcal{S}_n$  kann mittels  $\{x_1, \dots, x_n\}$  und den Operatoren  $\cdot, \vee, \neg$  dargestellt werden.

## Beweis:

Zunächst eine Notation:

Für ein Element  $g$  einer booleschen Algebra und  $q$  aus  $\mathcal{B}$  sei:

$$g^q := \begin{cases} g & \text{falls } q = 1 \\ \bar{g} & \text{falls } q = 0 \end{cases}$$

Betrachte nun für  $p \in \mathcal{B}^n$  das Element  $x^p := x_1^{p_1} \cdot \dots \cdot x_n^{p_n}$

# Beweis: ff

Dann ist  $x^p(q) = x_1^{p_1} \cdot \dots \cdot x_n^{p_n}(q) = 1$

$$\Leftrightarrow \forall 1 \leq i \leq n: x_i^{p_i}(q) = 1$$

$$\Leftrightarrow \forall 1 \leq i \leq n: (x_i(q))^{p_i} = 1$$

$$\Leftrightarrow \forall 1 \leq i \leq n: (q_i)^{p_i} = 1$$

$$\Leftrightarrow \forall 1 \leq i \leq n: q_i = p_i \quad \boxed{\Leftrightarrow p = q}$$

Also liefert  $x_1^{p_1} \cdot \dots \cdot x_n^{p_n}$  genau den Minterm  $x^p$

Da aber  $At(\mathcal{S}_n) = \{x^p | p \in \mathbf{B}^n\}$

können wir jedes Element  $f$  darstellen als

$$f = \bigvee_{p \in ON(f)} x_1^{p_1} \cdot \dots \cdot x_n^{p_n}$$

# Disjunktive Normalform

## Definition

Wir nennen die Menge

$$ON(f) := \{p \mid f(p) = 1\} \quad (= \{p \mid f \cdot x^p = x^p\})$$

**ON-Set der Funktion  $f$** , und die Form

$$f = \bigvee_{p \in ON(f)} x_1^{p_1} \cdot \dots \cdot x_n^{p_n}$$

die **disjunktive Normalform von  $f$** .

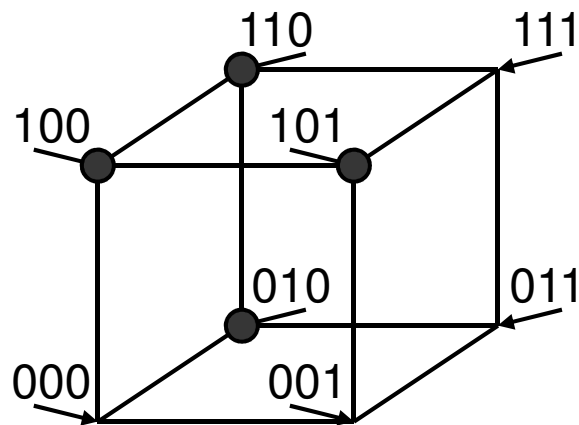
Jede totale Schaltfunktion in einem Ausgang läßt sich also eindeutig mit den Operationen and, or und not, sowie den Projektionen (Eingängen) darstellen.

# Anschaung:

Man kann die Vektoren  $p \in \mathbf{B}^n$  auch als Eckpunkte eines  $n$ -dimensionalen Einheitswürfels auffassen.

Eine Schaltfunktion kann man demnach assoziieren mit ihrem ON-Set, d.h. einer Teilmenge der Eckpunkte.

Einen Minterm  $x^p$  kann man gleichsetzen mit einem Eckpunkt:

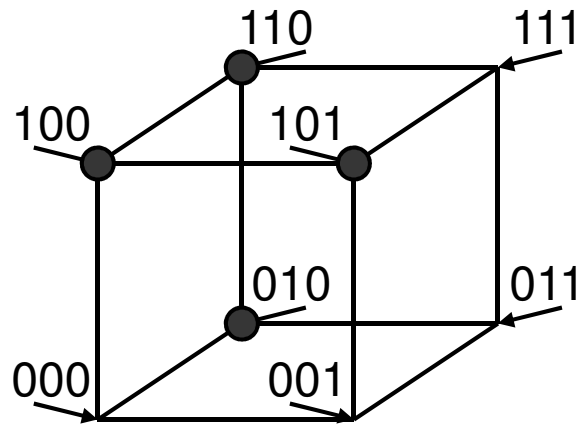


Geometrische Deutung

Funktionstafel

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

# Anschaung:



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

Wir lesen daraus:  $ON(f) = \{010, 100, 101, 110\}$

$$\text{Also: } f = x^{010} \vee x^{100} \vee x^{101} \vee x^{110}$$

$$= x_1^0 x_2^1 x_3^0 \vee x_1^1 x_2^0 x_3^0 \vee x_1^1 x_2^0 x_3^1 \vee x_1^1 x_2^1 x_3^0$$

$$= \overline{x_1} x_2 \overline{x_3} \vee x_1 \overline{x_2} \overline{x_3} \vee x_1 \overline{x_2} x_3 \vee x_1 x_2 \overline{x_3}$$



# Boolesche Ausdrücke

Wir benutzen schon seit Einführung der booleschen Algebra Ausdrücke. Da wir im Verlauf der Vorlesung noch viel Gebrauch davon machen werden, wollen wir sie nun auf eine solide, formale Basis stellen.

Ausdrücke sind zunächst bedeutungslos (Kreidehaufen an der Tafel oder Pixel im Display). Sie erhalten eine Bedeutung dadurch, dass man zunächst ihre Struktur, die Syntax, festlegt, und sie dann als Elemente einer Algebra interpretiert.

## Definition

Sei  $Y = \{y_1, \dots, y_n\}$  eine Menge von Namen. Dann nennen wir Ausdrücke der Form  $y_i, \bar{y}_i$  **Literale**.

# Boolesche Ausdrücke ff

## Definition

Die Menge der **booleschen Ausdrücke** über einer Variablenmenge  $Y = \{y_1, \dots, y_m\}$ ,  $BA(Y)$ , ist syntaktisch gegeben durch

$\langle \text{boole\_Ausdruck} \rangle$

$::= \langle \text{boole\_Ausdruck} \rangle \vee \langle \text{boole\_Term} \rangle \mid \langle \text{boole\_Term} \rangle$

$\langle \text{boole\_Term} \rangle$

$::= \langle \text{boole\_Term} \rangle \cdot \langle \text{boole\_Faktor} \rangle \mid \langle \text{boole\_Faktor} \rangle$

$\langle \text{boole\_Faktor} \rangle$

$::= (\langle \text{boole\_Ausdruck} \rangle) \mid \neg \langle \text{boole\_Faktor} \rangle$

$\mid 0 \mid 1 \mid y_1 \mid \dots \mid y_m$

# Boolesche Ausdrücke ff

## Bemerkungen:

Terme sind stets eindeutig zerlegbar in ein Produkt aus Term und Faktor oder sind selbst ein Faktor.

Ausdrücke sind stets eindeutig zerlegbar in eine Disjunktion aus Ausdruck und Term oder sind selbst ein Term

Faktoren sind Konstante, Literale, geklammerte Ausdrücke oder negierte Faktoren.

**Beispiele:** Sei  $Y = \{a, b, c\}$

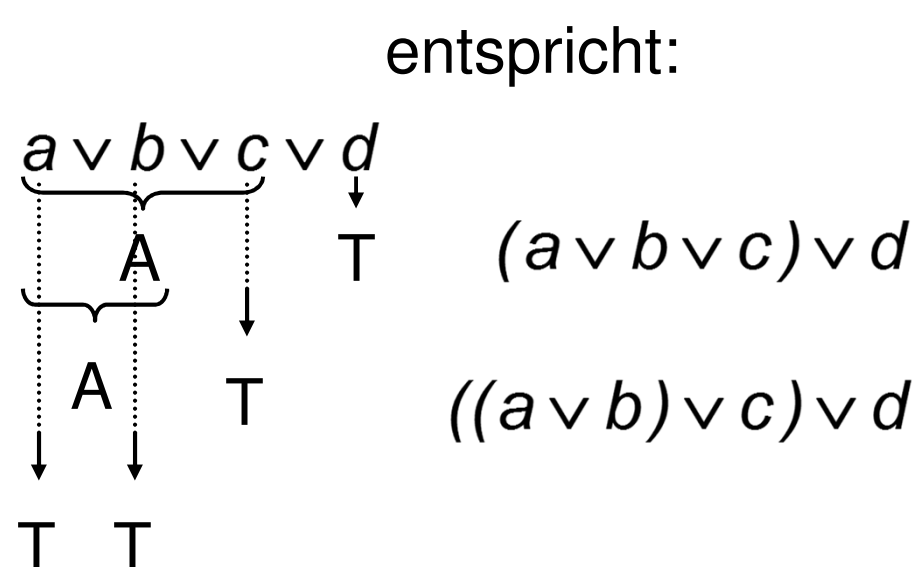
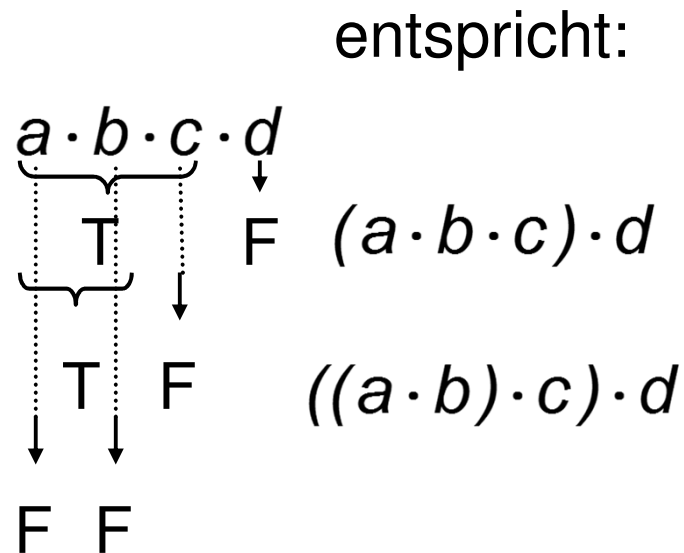
$a, a \vee b, a \cdot (\bar{b} \vee c)$ Ausdrücke	$a, a \cdot (\bar{b} \vee c)$ Terme	$a, \bar{b}, (\bar{b} \vee c)$ Faktoren
--	--	--

# Boolesche Ausdrücke ff

## Bemerkungen:

Durch obige Definition ist ferner eine Zerlegung vorgegeben, die implizit eine Klammerung von links nach rechts bedeutet:

## Beispiel:



# Spezielle Formen von Ausdrücken

Sei  $p = p_1 \cdot \dots \cdot p_n$  und jedes  $p_i$  sei ein Literal. Dann heißt  $p$  ein **Produkt** (Monom oder Cube).

Hat ein Ausdruck  $w$  die Form  $w = w_1 \vee \dots \vee w_n$  und jedes  $w_i$  ist ein Produkt, dann heißt  $w$  **disjunktive Form** (DF oder boolesches Polynom oder Summe von Produkten).

Dual dazu kann man auch betrachten:

Sei  $c = c_1 \vee \dots \vee c_n$  und jedes  $c_i$  sei ein Literal. Dann heißt  $c$  eine **Klausel**.

Hat ein Ausdruck  $w$  die Form  $w = (w_1) \cdot \dots \cdot (w_n)$  und jedes der  $w_i$  ist eine Klausel, dann heißt  $w$  **konjunktive Form** (CF oder Produkt von Summen).

# Syntaxbäume

Durch obige Definition ist eine eindeutige Zerlegung der Ausdrücke in Teilausdrücke vorgegeben. Diese Zerlegung kann man auch in einem Diagramm veranschaulichen, das wir induktiv nach der Länge eines Ausdruck definieren wollen:

Sei  $w$  ein boolescher Ausdruck, dann ist der **abstrakte Syntaxbaum** (Operatorbaum)  $T(w)$  zu  $w$  gegeben wie folgt:

1.  $|w| = 1$ : Dann ist  $w$  eine Konstante oder Variable. Wir setzen

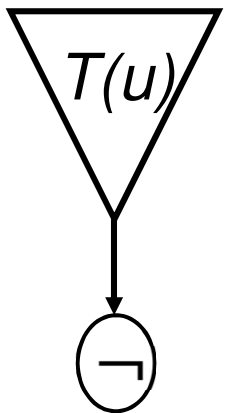
$$T(w) := \boxed{0} \quad \text{bzw.} \quad \boxed{1} \quad \text{bzw.} \quad \boxed{y_i}$$

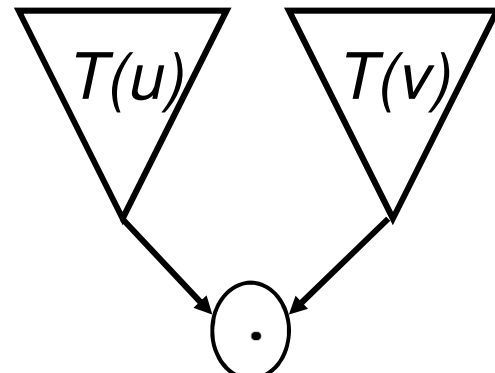
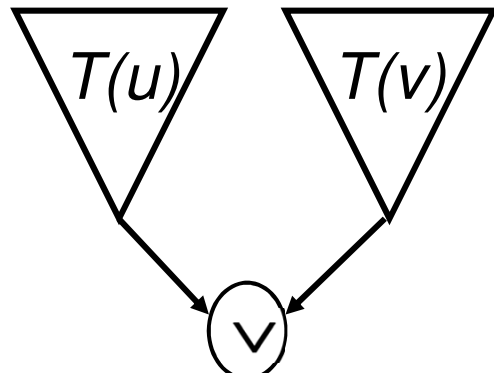
für  $T(w) = y_i \in Y$

# Syntaxbäume ff

2. Sei für  $u$ ,  $|u| < |w|$   $T(u)$  schon definiert.

Dann setze  $T(w) := T(u)$ , falls  $w = (u)$

$T(w) :=$   falls  $w = \neg u$

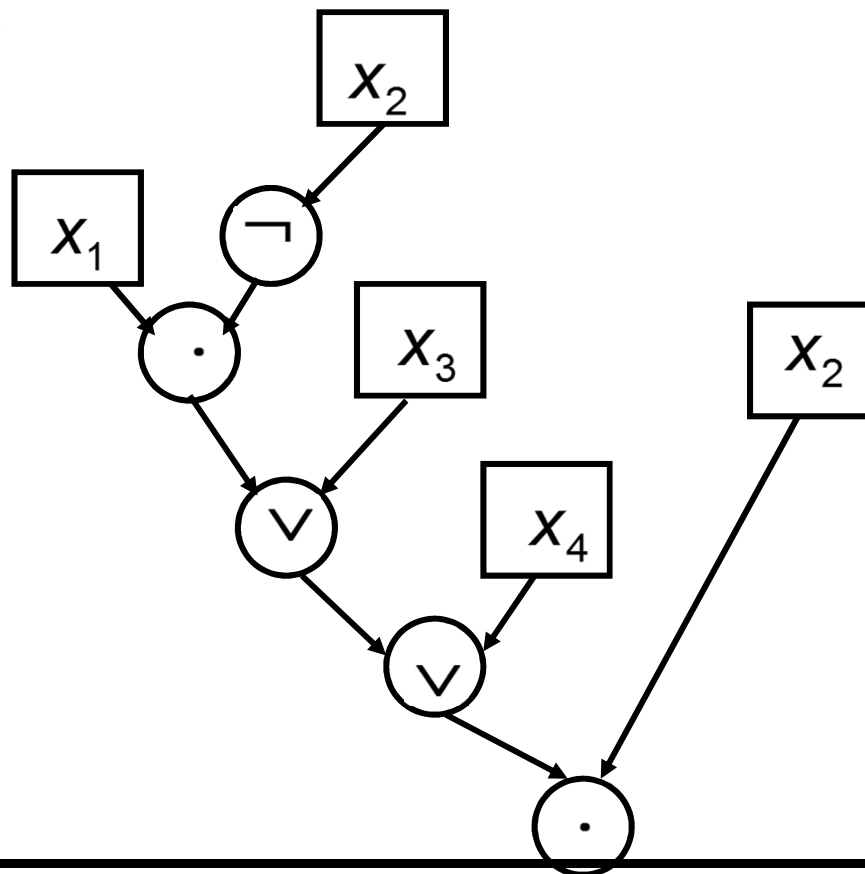
$T(w) :=$   falls  $w = u \cdot v$        falls  $w = u \vee v$

# Beispiel

Der abstrakte Syntaxbaum  $T(w)$  zu folgender Formel  $w$  ist demnach:

$$w = (x_1 \cdot \overline{x_2} \vee x_3 \vee x_4) \cdot x_2$$

$T(w) =$





# Interpretation boolescher Ausdrücke

## Definition

Die Fortsetzung  $I$  einer Zuordnung  $I': Y \mapsto M$  in eine boolesche Algebra  $M$  auf Ausdrücke  $w \in BA(Y)$  nachfolgender Regel

$$I(w) := \begin{cases} w & \text{falls } w \in \{0,1\} \\ I'(w) & \text{falls } w \in Y \\ \overline{I(u)} & \text{falls } w = \bar{u} \\ I(u) & \text{falls } w = (u) \\ I(u) \cdot I(v) & \text{falls } w = u \cdot v \\ I(u) \vee I(v) & \text{falls } w = u \vee v \end{cases}$$

heißt **Interpretation**.

# Interpretation ff

## Definition

Seien  $u, v$  boolesche Ausdrücke. Dann setzen wir

$$\begin{aligned}u &\equiv_I v: \Leftrightarrow I(u) = I(v) \\ u &\equiv v: \Leftrightarrow \forall I: I(u) = I(v)\end{aligned}$$

Beide Relationen sind Äquivalenzrelationen und zerlegen die Menge der Ausdrücke in Klassen. Dabei ist der erste Äquivalenzbegriff schwächer als der zweite, d.h.

$$[u]_{\equiv} \subseteq [u]_{\equiv_I}$$

$u \equiv v$  besagt, dass  $u$  und  $v$  ausschließlich mit den Rechenregeln der booleschen Algebra ineinander überführt werden können.

# Beispiel:

Betrachte folgende Interpretation von  $BA(\{a,b\})$  nach  $\mathcal{S}_2^D$   
mit  $D=\{00,01,10\}$  und  $I(a) := x_1, I(b) := x_2$

Dann gilt

$$\bar{a}b \vee a\bar{b} \equiv_I a \vee b$$

denn

$x_1$	$x_2$	$\bar{x}_1 x_2 \vee x_1 \bar{x}_2$	$x_1 \vee x_2$
0	0	0	0
0	1	1	1
1	0	1	1

=

Wählt man aber  $\mathcal{I}'$  nach  $\mathcal{S}_2$

$x_1$	$x_2$	$\bar{x}_1 x_2 \vee x_1 \bar{x}_2$	$x_1 \vee x_2$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

$\neq$

Also  $\bar{a}b \vee a\bar{b} \not\equiv_{\mathcal{I}'} a \vee b$

Anmerkung: Wir notiereren auch  $a \oplus b := \bar{a}b \vee a\bar{b}$

(Exor oder plus mod 2 oder Antivalenz )

Es ist  $a \equiv b = \overline{a \oplus b} = a \oplus b \oplus 1$  (Äquivalenz )

# Darstellung

## Definition

Sei  $I: BA(Y) \mapsto M$  eine Interpretation.

Für  $w \in BA(Y)$  sagen wir

**$w$  ist Darstellung von  $a$  :  $\Leftrightarrow I(w) = a$**

Die Menge der Darstellungen eines Elements  $a$  bei Interpretation  $I$  ist unendlich groß und gegeben durch

$$[w]_{\equiv_I} \text{ mit } I(w) = a$$

Es ist zum Beispiel:  $w \vee 0, w \vee 0 \vee 0, \dots \in [w]_{\equiv_I}$

aber auch  $w \cdot (u \vee v) \in [w]_{\equiv_I}$  für  $u \in [w]_{\equiv_I}$  und  $v$  beliebig.

# Das Syntheseproblem

Typisch in der Informatik sind Konstruktionsprobleme, die sich immer dann stellen, wenn man keine eindeutige Darstellung für die zu konstruierenden (darzustellenden) Objekte hat. Allgemein lautet dies:

## Syntheseproblem

Gegeben:  $w \in X$

eine Interpretation  $I: X \mapsto M$

eine Kostenfunktion  $C: X \mapsto \mathbf{R}$

Gesucht:  $v \in X$  mit  $I(v) = I(w)$

und  $C(v) = \min\{C(u) | I(u) = I(w)\}$

# Das Syntheseproblem

Übertragen auf Ausdrücke bedeutet dies zu Beispiel:

Gegeben:     ein boolescher Ausdruck  $w$   
              eine Interpretation als Schaltfunktion  
              etwa die Länge als Kostenfunktion

Gesucht:     ein kürzester Ausdruck mit gleicher  
              Interpretation

**Für dieses Problem gibt es bis heute**  
(außer erschöpfenden Verfahren)  
**keine Lösungsmethoden!!**

# Bemerkungen

Es ist entscheidend, wie man Ausdrücke interpretiert. Offenbar gibt es boolesche Algebren in denen Rechenregeln gelten, die nicht aus den Axiomen ableitbar sind.

Eine boolesche Algebra, in der alle Beziehungen aus den Axiomen ableitbar sind heißt **frei**. Frei ist z.B. die Algebra der totalen Schaltfunktionen.

Meist interpretieren wir die Variablen  $x_i$  als Projektionen auf die  $i$ -te Komponente über der Menge der Schaltfunktionen, d.h.  $x_i(p) = p_i$ . In diesem Falle werden wir nichts weiter dazu sagen, und nennen diese Interpretation auch **Standardinterpretation**.

Zusätzlich benutzen wir aber oft weitere Namen bzw. Variablen, die andere Schaltfunktionen bezeichnen. Formal kann man deren Interpretation auch dadurch beschreiben, dass man sie durch einen Ausdruck ersetzt:

# Substitution

## Definition

Sei  $w \in BA(Y)$  und seien  $a_1, \dots, a_k \in BA(Y)$

Dann sei  $w(y_1 = a_1, \dots, y_k = a_k)$  der Ausdruck, der durch Ersetzung aller Vorkommen von  $y_i$  durch  $a_i$  entsteht.

Wir nennen diesen Vorgang auch **Substitution**.

Beispiel:

$$w = y_1 \cdot \overline{y_2} \vee y_3$$

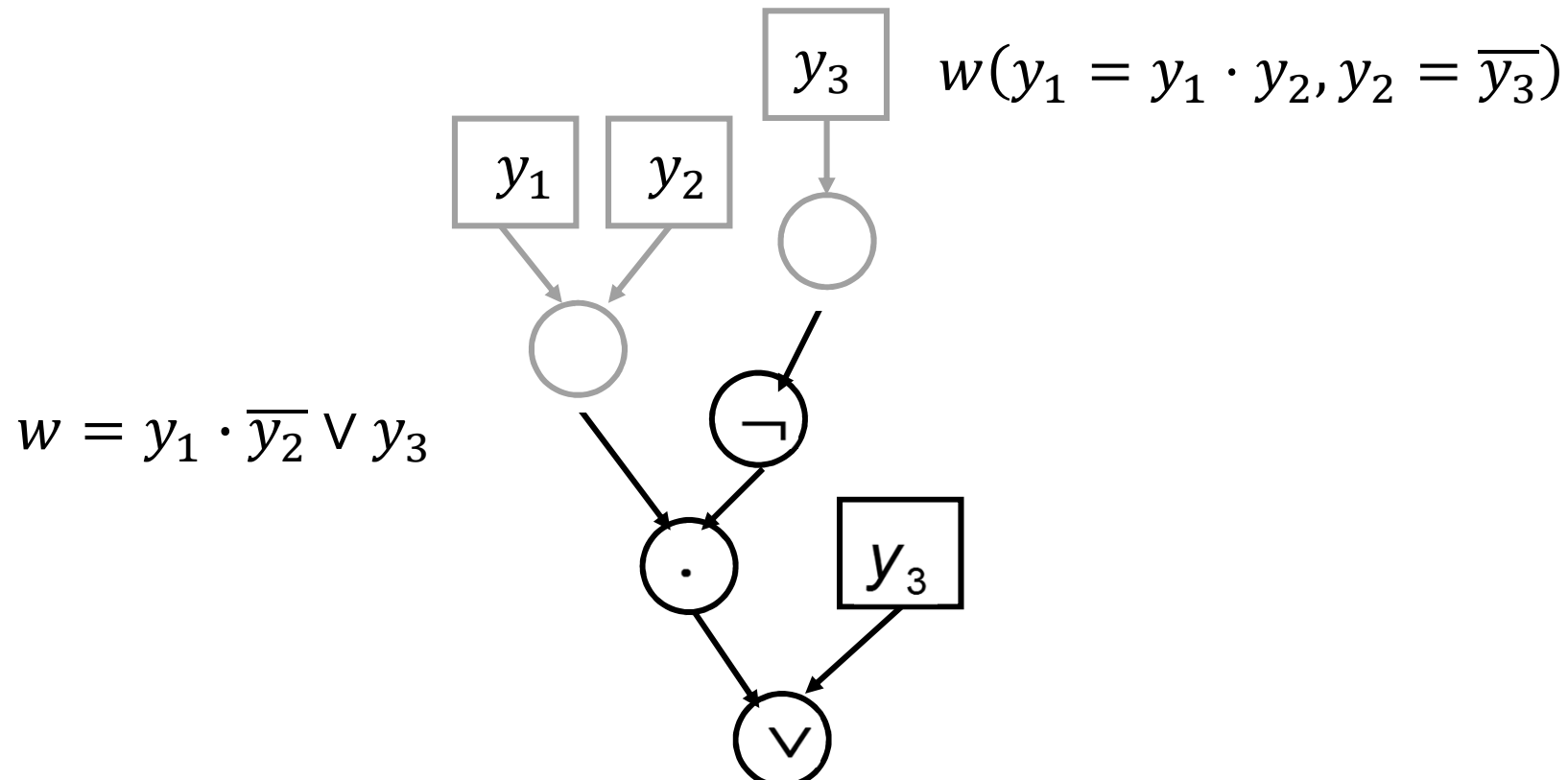
$$w(y_1 = y_1 \cdot y_2, y_2 = \overline{y_3}) = (y_1 \cdot y_2) \overline{(\overline{y_3})} \vee y_3$$



# Substitution ff

Auf Syntaxbäumen ist die Substitution sehr anschaulich:

Man ersetze einfach die entsprechenden Blätter durch Syntaxbäume für die substituierten Ausdrücke:



# Substitution ff

## Bemerkungen:

- Eine Substitution aller Variablen mit  $0, 1$  entspricht einer Auswertung der dargestellten Funktion auf einem Punkt.
- Substitutionen lassen sich rekursiv auf Teilausdrücken vornehmen (  $uv(a=w) = u(a=w)v(a=w)$  ).
- Substitutionen sind reihenfolgenabhängig, wenn der substituierte Ausdruck Variablen enthält, die ebenfalls substituiert werden:

aber

$$\begin{aligned} & (y_1 \cdot \overline{y_2} \vee y_3)(y_1 = y_1 \cdot y_3)(y_3 = y_2) \\ &= ((y_1 y_3) \cdot \overline{y_2} \vee y_3)(y_3 = y_2) \\ &= ((y_1 y_3) \cdot \overline{y_2} \vee y_2) = y_2 \end{aligned}$$

$$\begin{aligned} & (y_1 \cdot \overline{y_2} \vee y_3)(y_3 = y_2)(y_1 = y_1 \cdot y_3) \\ &= (y_1 \cdot \overline{y_2} \vee y_2)(y_1 = y_1 \cdot y_3) \\ &= ((y_1 y_3) \cdot \overline{y_2} \vee y_2) \end{aligned}$$

# Kofaktoren

## Definition

Sei  $f \in \mathcal{S}_n$  eine Schaltfunktion. Dann heißen die Funktionen

$$f_{x_i} \text{ mit } f_{x_i}(x_1, \dots, x_n) := f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

$$f_{\bar{x}_i} \text{ mit } f_{\bar{x}_i}(x_1, \dots, x_n) := f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

**Kofaktoren** von  $f$  nach  $x_i$  bzw.  $\bar{x}_i$

Offenbar hängen die Kofaktoren einer Funktion nicht mehr vom Wert der  $i$ -ten Komponente ab, da diese grundsätzlich als  $1$  oder  $0$  betrachtet wird. Damit hat man eine Methode, ein Problem für eine Funktion, die von  $n$  Variablen abhängt, auf Funktionen, die nur noch von  $n-1$  Variablen abhängen, zurückzuführen.

# Kofaktoren

## Definition

Sei  $f \in \mathcal{S}_n$  eine Schaltfunktion. Wir sagen  
 $f$  ist unabhängig von  $x_i : \Leftrightarrow f_{x_i} = f_{\bar{x}_i}$

## Definition

Sei  $f \in \mathcal{S}_n$  eine Schaltfunktion. Wir nennen

$$\frac{\partial f}{\partial x_i} := f_{x_i} \oplus f_{\bar{x}_i}$$

die **boolesche Differenz** von  $f$  nach  $x_i$

Beobachtung:  $f$  ist offenbar unabhängig von  $x_i$

$$: \Leftrightarrow f_{x_i} = f_{\bar{x}_i} \Leftrightarrow (f_{x_i} \equiv f_{\bar{x}_i}) = 1 \Leftrightarrow \overline{(f_{x_i} \equiv f_{\bar{x}_i})} = 0 \Leftrightarrow f_{x_i} \oplus f_{\bar{x}_i} = 0 \Leftrightarrow \frac{\partial f}{\partial x_i} = 0$$

# Support

## Definition

Sei  $f \in \mathcal{S}_n$  eine Schaltfunktion. Wir nennen die Menge der Variablen, von denen  $f$  abhängt

$$\text{supp}(f) := \{x_i \mid f_{x_i} \neq f_{\bar{x}_i}\}$$

den **Support** von  $f$ .

## Beobachtungen:

Interpretiert man einen booleschen Ausdruck als Schaltfunktion, so kann diese nur von Variablen abhängen, die in dem Ausdruck vorkommen.

Darstellungen von Kofaktoren der dargestellten Funktion erhält man ferner ganz leicht durch Substitution der entsprechenden Variablen mit den entsprechenden Konstanten.

# Support auf Ausdrücken

## Definition

Sei  $w \in BA(X)$  ein boolescher Ausdruck. Wir nennen die Menge der Variablen, die in  $w$  vorkommen, den (syntaktischen) **Support** von  $w$ ,  $ssupp(w)$

## Lemma

Sei  $w \in BA(X)$  und sei  $I$  die Standardinterpretation. Dann gilt:

$$(a) \ I(w(x_i = 1)) = I(w)_{x_i}, \ I(w(x_i = 0)) = I(w)_{\bar{x}_i}$$

$$(b) \ ssupp(w) \supseteq supp(I(w))$$

# Support auf Ausdrücken ff

**Beweis:**

$$\begin{aligned} \text{(a): } I(w(x_i = 1))(x_1, \dots, x_n) &= I(w)(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \\ &= I(w)_{x_i}(x_1, \dots, x_n) \end{aligned}$$

analog folgt  $I(w(x_i = 0)) = I(w)_{\bar{x}_i}$

(b): Ist  $x_i \notin \text{ssupp}(w)$ , ist schon  $w(x_i = 1) = w = w(x_i = 0)$

also ist  $I(w)_{x_i} = I(w(x_i = 1)) = I(w) = I(w)_{\bar{x}_i}$

und damit  $x_i \notin \text{supp}(I(w))$

Vorsicht: Es gilt nicht  $\text{ssupp}(w) = \text{supp}(I(w))$

Denn:  $\text{ssupp}(x_1 \vee x_1 x_2) = \{x_1, x_2\}$

Aber:  $\text{supp}(I(x_1 \vee x_1 x_2)) = \text{supp}(I(x_1)) = \{x_1\}$

# Eigenschaften der Kofaktoroperation

## Lemma

Sei  $f \in \mathcal{S}_n$ . Dann gilt:

- (a) Für  $\varepsilon, \delta \in \{0, 1\}$  ist  $x_i^\varepsilon x_j^\delta = \begin{cases} 1 & i = j \text{ und } \delta = \varepsilon \\ 0 & i = j \text{ und } \delta \neq \varepsilon \\ x_i^\varepsilon & \text{sonst} \end{cases}$
- (b)  $(f_{x_i^\varepsilon})_{x_j^\delta} =: f_{x_i^\varepsilon x_j^\delta} = f_{x_j^\delta x_i^\varepsilon} := (f_{x_j^\delta})_{x_i^\varepsilon}$
- (c)  $(fg)_{x_i^\varepsilon} = f_{x_i^\varepsilon} g_{x_i^\varepsilon}; (f \vee g)_{x_i^\varepsilon} = f_{x_i^\varepsilon} \vee g_{x_i^\varepsilon}; \overline{f_{x_i^\varepsilon}} = \overline{f}_{x_i^\varepsilon};$

**Beweis:** Sei  $I$  die Standardinterpretation und  $w$  eine Darstellung von  $f$ .



# Eigenschaften ff

$$\begin{aligned}
 \text{(a): } x_i^\varepsilon x_j^\delta &= I(x_i^\varepsilon)_{x_j^\delta} = I\left(x_i^\varepsilon(x_j = \delta)\right) \\
 &= \begin{cases} I(\delta^\varepsilon) & i = j \\ I(x_i^\varepsilon) & \text{sonst} \end{cases} \\
 &\begin{cases} 1 & \text{falls } i = j \text{ und } \delta = \varepsilon \\ 0 & \text{falls } i = j \text{ und } \delta \neq \varepsilon \\ x_i^\varepsilon & \text{sonst} \end{cases} \quad \begin{pmatrix} 0^0 = \bar{0} = 1 = 1^1 \\ 1^0 = \bar{1} = 0 = 0^1 \end{pmatrix}
 \end{aligned}$$

$$\begin{aligned}
 \text{(b): } (f_{x_i^\varepsilon})_{x_j^\delta} &= (I(w)_{x_i^\varepsilon})_{x_j^\delta} = I(w(x_i = \varepsilon))_{x_j^\delta} \\
 &= I\left(w(x_i = \varepsilon)(x_j = \delta)\right) = I\left(w(x_i = \varepsilon, x_j = \delta)\right) \\
 &= I\left(w(x_j = \delta, x_i = \varepsilon)\right) = I(w(x_i = \delta))_{x_i^\varepsilon} \\
 &= (I(w)_{x_j^\delta})_{x_i^\varepsilon} = (f_{x_j^\delta})_{x_i^\varepsilon}
 \end{aligned}$$

# Eigenschaften ff

(c): Seien  $v, w$  Ausdrücke, mit  $I(v) = f, I(w) = g$ , dann ist

$$\begin{aligned}(f \cdot g)_{x_i^\varepsilon} &= (I(v) \cdot I(w))_{x_i^\varepsilon} = I((v \cdot w)(x_i = \varepsilon)) \\ &= I(v(x_i = \varepsilon) \cdot w(x_i = \varepsilon)) \\ &= I(v(x_i = \varepsilon)) \cdot I(w(x_i = \varepsilon)) = f_{x_i^\varepsilon} \cdot g_{x_i^\varepsilon}\end{aligned}$$

Analog die anderen Operationen.

## Satz

Jede Schaltfunktion  $f \in \mathcal{S}_n$  kann dargestellt werden durch die **Shannon Expansion**

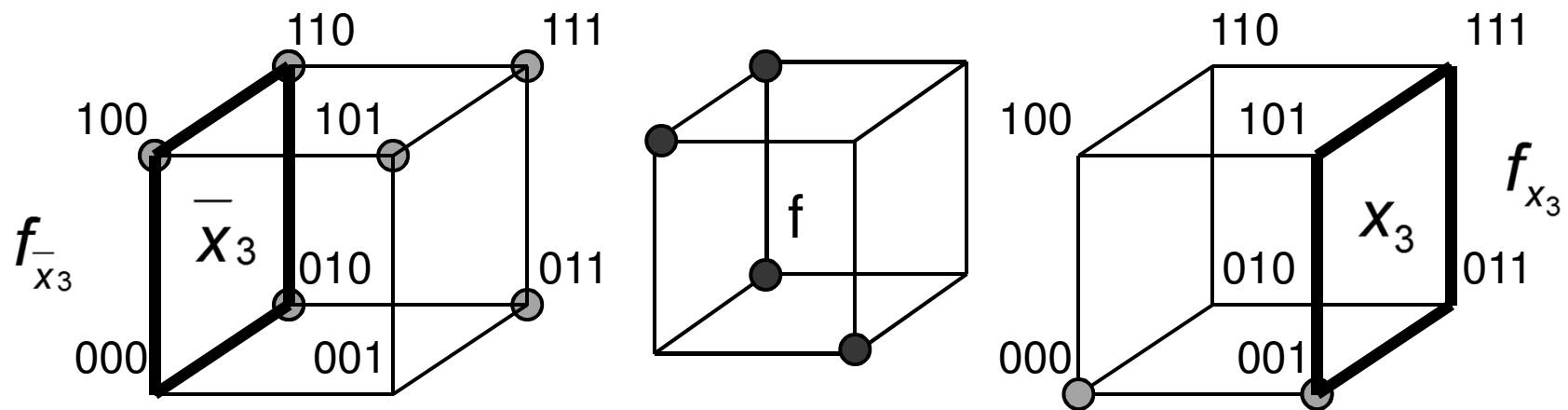
$$f = \bar{x}_i \cdot f_{\bar{x}_i} \vee x_i \cdot f_{x_i}$$

# Eigenschaften ff

**Beweis:**

$$\begin{aligned}
 (\bar{x}_i f_{\bar{x}_i} \vee x_i f_{x_i})(q_1, \dots, q_n) &= \bar{x}_i f_{\bar{x}_i}(q_1, \dots, q_n) \vee x_i f_{x_i}(q_1, \dots, q_n) \\
 &= \begin{cases} f_{\bar{x}_i}(q_1, \dots, q_n) \vee 0 & \text{falls } q_i = 0 \\ 0 \vee f_{x_i}(q_1, \dots, q_n) & \text{falls } q_i = 1 \end{cases} \\
 &= f(q_1, \dots, q_n)
 \end{aligned}$$

**Anschaung:**



# Boolesche Ausdrücke in WüHDL

In WüHDL (und auch VHDL) gibt es für die vordefinierten Typen BOOLEAN und BIT die binären Operatoren

AND, OR, NAND, NOR, XOR, XNOR

mit gleicher Präzedenz. D.h. es gibt nicht wie in unseren Formeln eine stärkere Bindung von AND gegenüber OR!

Lediglich der Operator NOT hat eine höhere Präzedenz.

Die Auswertung erfolgt von links nach rechts. Entscheidet der linke Operand des Operators den Wert des Ausdrucks, dann wird der rechte Operand nicht ausgewertet.

# Schaltfunktionen in WüHDL

Man kann nun in WüHDL auf verschiedene Art und Weise Schaltfunktionen spezifizieren, die von speziellen Werkzeugen (Synthesewerkzeuge) in Schaltungen umgesetzt werden können. Es hängt stark vom Synthesewerkzeug ab, welche Konstrukte es als Spezifikation einer Schaltfunktion interpretieren kann, und wie es diese Spezifikationen letztlich in Schaltungen umsetzt.

Da eine Behandlung dieses Themas den Umfang der Vorlesung sprengen würde, wollen wir uns hier zunächst auf ein spezielles Konstrukt beschränken, das wir auch noch als "boolesche Gleichungen" behandeln werden:

**concurrent signal assignments**

# Concurrent Signal Assignments

Man kann im Rumpf einer ARCHITECTURE statt innerhalb von Prozessen auch direkt Signalzuweisungen angeben. Zur Definition von Schaltfunktionen beschränken wir uns auf Signale vom Typ BIT.

Bedingungen:

- die zugewiesenen Signale sind entweder Ports der ENTITY vom Modus OUT oder lokale Signale der ARCHITECTURE
- zu jedem Signal gibt es höchstens eine Zuweisung
- auf der rechten Seite steht ein Ausdruck vom Typ des Signals ohne Angaben von Verzögerungen
- im Ausdruck auf der rechten Seite kommen nur lokale Signale oder Ports der ENTITY vom Modus IN vor
- lokale Signale dürfen nicht zyklisch voneinander abhängen

# Concurrent Signal Assignments ff

## Erläuterungen:

concurrent signal assignments treten in folgender Form auf:

```
BEGIN -- Architecture
```

```
    s <= (a AND b) OR NOT c;
```

```
    t <= (a XOR c) NAND b;
```

```
    PROCESS ...
```

```
END Architecture;
```

Im Grunde ist jede einzelne Signalzuweisung  $\text{sig} \leq \text{Ausdruck}(a, \dots, z)$ ; eine Kurznotation für

```
    PROCESS (a, ..., z)
```

```
        BEGIN sig <= Ausdruck(a, ..., z); END PROCESS;
```

Nun zu den Bedingungen:

# Concurrent Signal Assignments ff

Die zugewiesenen Signale sind Ports der ENTITY vom Modus OUT oder lokale Signale:

Andere Signale sind im Rumpf der Architecture nicht sichtbar bzw. schreibbar.

Zu jedem Signal gibt es höchstens eine Zuweisung:

Mit jedem Signal soll eine Schaltfunktion assoziiert werden können. Dies ist die Funktion, die man als Interpretation des Ausdrucks der rechten Seite erhält (Signaltyp = BIT).

Auf der rechten Seite steht ein Ausdruck vom Typ des Signals ohne Angaben von Verzögerungen:

Eine Schaltfunktion ist nicht zeitbezogen, deswegen sollten bei der Spezifikation die Verzögerungen = 0 sein. Ein Synthesewerkzeug erzeugt zu einer solchen synthetisierbaren Architecture dann einen Schaltkreis, der durchaus Verzögerungen hat. Manche Synthesewerkzeuge ignorieren auch einfach nur Verzögerungsangaben.



# Concurrent Signal Assignments ff

Im Ausdruck auf der rechten Seite kommen nur lokale Signale oder Ports der ENTITY vom Modus IN vor:

Andere Signale sind im Rumpf der Architecture nicht sichtbar oder lesbar.

Lokale Signale sollten nicht zyklisch voneinander abhängen:

Dies ist ein subtiler Punkt. Ein Signal  $s$  hängt direkt ab von  $s'$ , wenn  $s'$  auf der rechten Seite einer Zuweisung an  $s$  steht. Abhängigkeit ist der transitive Abschluss der Relation "hängt direkt ab von". Ist  $s$  von sich selbst abhängig, dann hat man eine zyklische Abhängigkeit. Man kann nun  $s$  keine Schaltfunktion mehr zuweisen, sondern eine solche Abhängigkeit produziert im Allgemeinen Gedächtnis.

Beispiel:

Es ist leicht, unser Basis R/S Latch durch Assignments mit zyklischer Abhängigkeit zu beschreiben

$qz \leq rz \text{ NAND } q;$

$q \leq sz \text{ NAND } qz;$