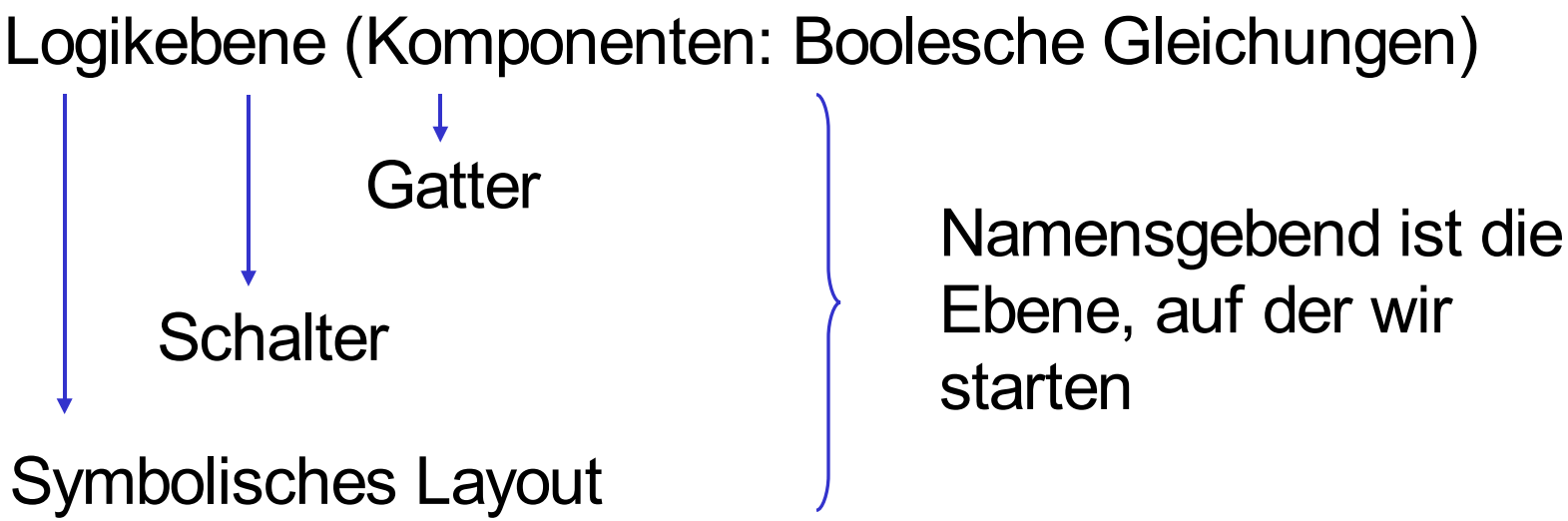
Logiksynthese

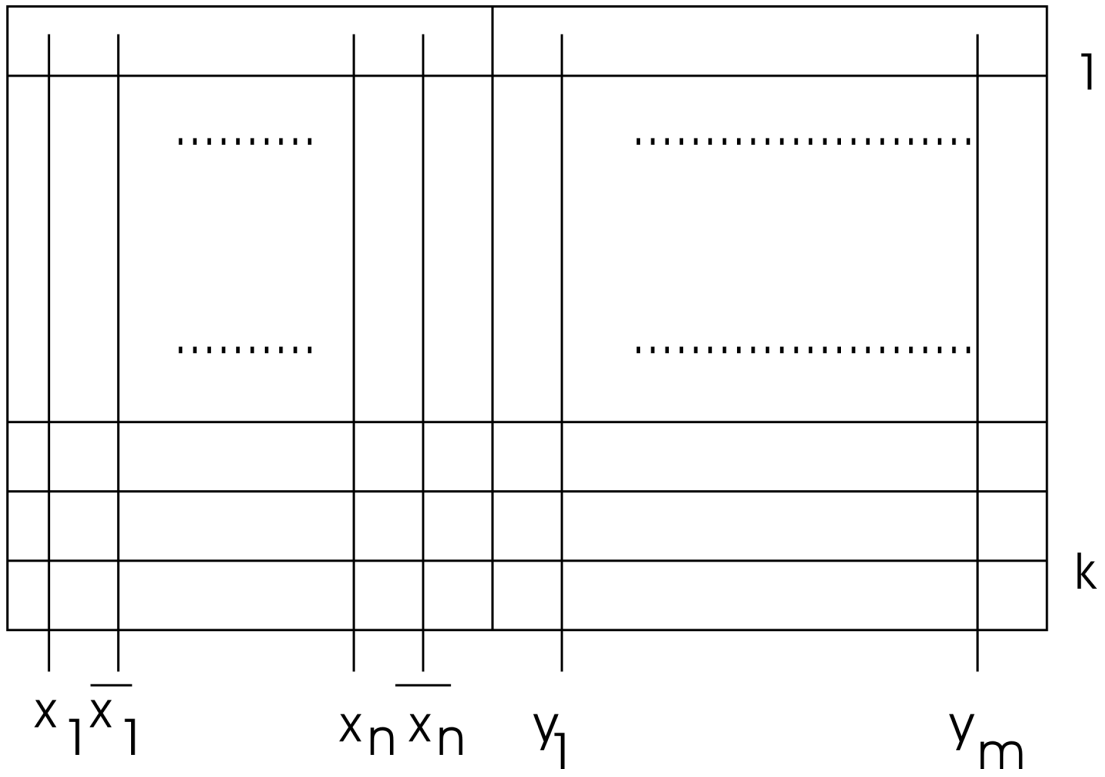
Peter Marwedel
TU Dortmund,
Informatik 12

Begriff der Logiksynthese



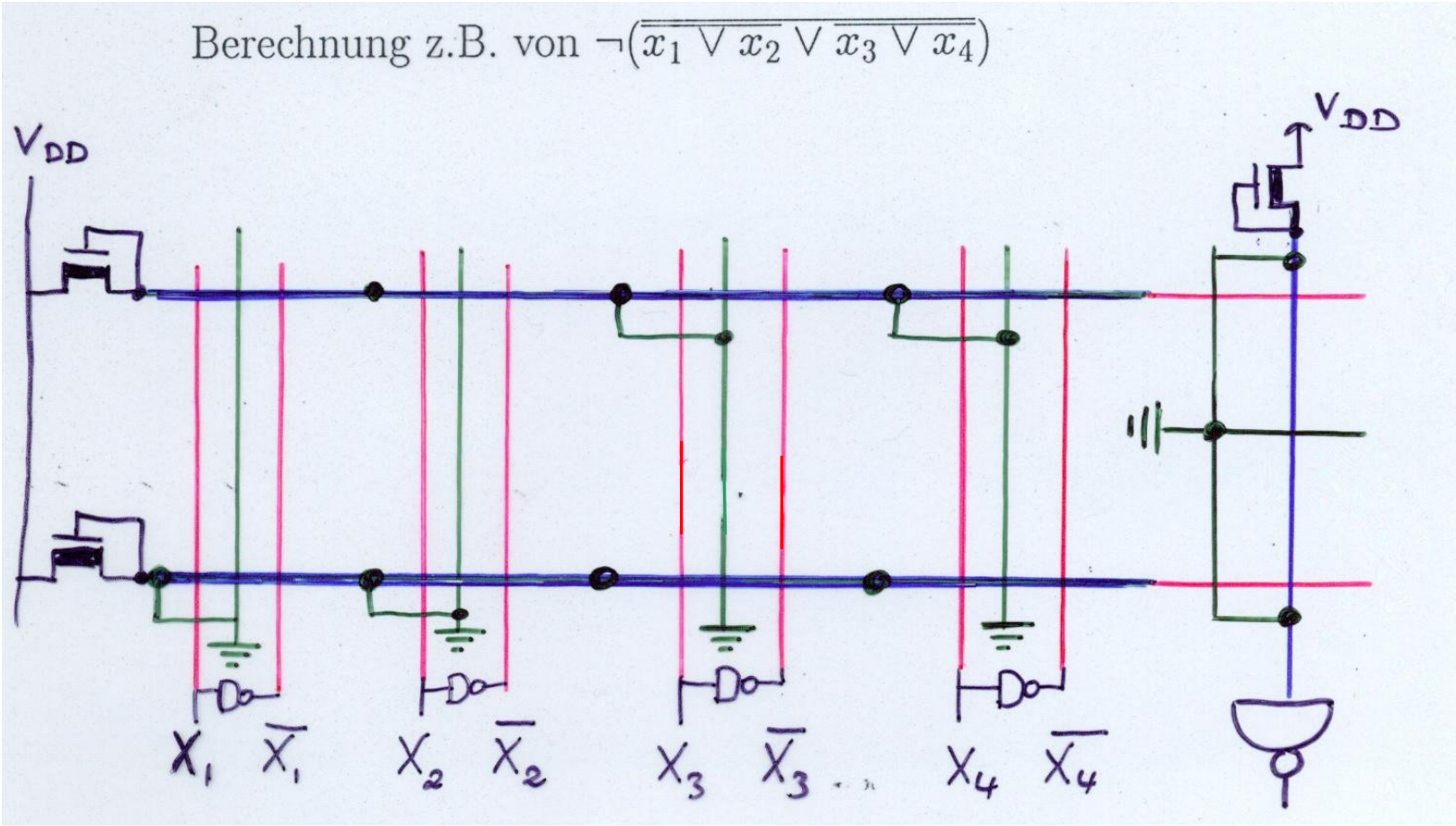
Klassische Minimierungstechniken für 2-stufige Logik

Nützlich für PLAs (*programmable logic arrays*):
PLAs sind flächeneffiziente, 2-stufige Realisierung Boolescher Funktionen



Elektrotechnische Realisierung

Aus elektrotechnischen Gründen in beiden Stufen NOR-Gatter + Invertierung des Ergebnisses

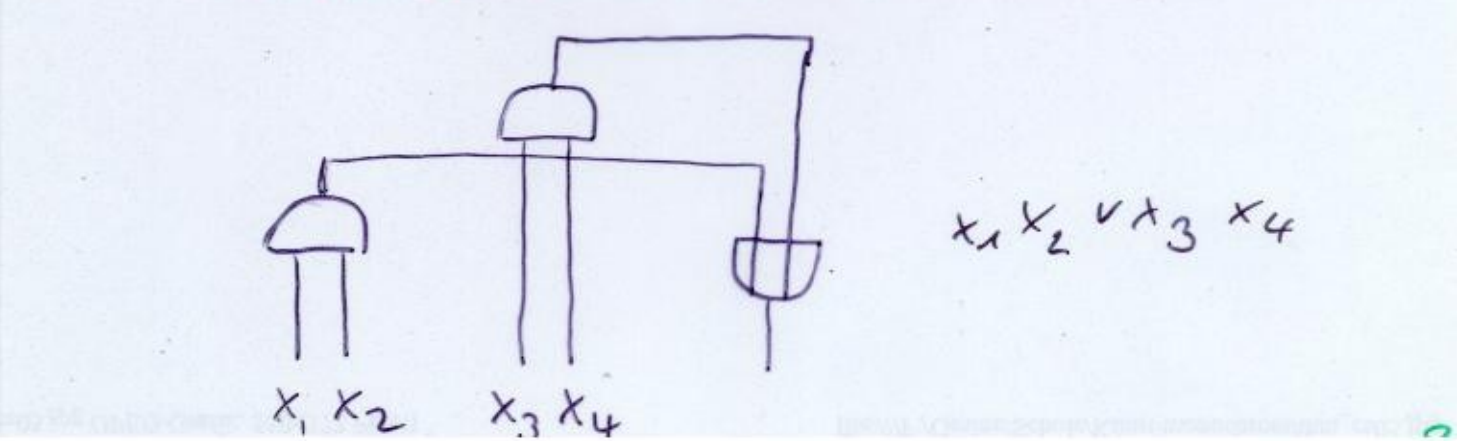


Vergleich mit AND und OR-PLanes

Unterscheidet sich nur durch komplementierte Eingangssignale

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

Denn: Falls es echte AND und OR-Planes wären, würde bei Anschluss an die nicht-invertierenden Eingänge die folgende Funktion berechnet:



Charakterisierung von PLAs (*Personality*)

Spezifische Eigenschaften von PLAs in Matrizen *and* und *or* festgelegt:

Def.: *and* : $[1..k] \times [1..n] \rightarrow [0..2]$ ist definiert durch:

- *and*(*i*,*j*) = 0, falls in Produktterm *i* Variable x_j invertiert* vorkommt
- *and*(*i*,*j*) = 1, falls in Produktterm *i* Variable x_j nicht-invertiert vorkommt,
- *and*(*i*,*j*) = 2, falls in Produktterm *i* die Variable x_j nicht vorkommt.

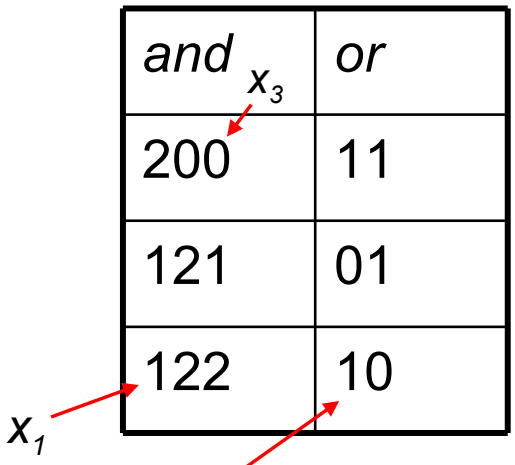
Def.: *or* : $[1..k] \times [1..m] \rightarrow [0..1]$ ist definiert durch:

- *or*(*i*,*j*) = 0, falls Produktterm *i* in Berechnung von y_j nicht eingeht,
- *or*(*i*,*j*) = 1, falls Produktterm *i* in Berechnung von y_j eingeht.

* Bei der MOS-Technik ist in diesem Fall die **nicht-invertierte** Spaltenleitung an einen Transistor angeschlossen.

Beispiel

Gegeben sei das PLA mit der Individualität



Erste Zeile: $\overline{x_2} \cdot \overline{x_3}$

Zweite Zeile: Term $x_1 x_3$

Dritte Zeile: Term x_1

$$y_1 = \overline{x_2} \cdot \overline{x_3} + x_1$$

$$y_2 = \overline{x_2} \cdot \overline{x_3} + x_1 \cdot x_3$$

Reduktion der Fläche von PLAs

Techniken der klassischen Booleschen Minimierung im Prinzip anwendbar.

Für praktisch relevante Problemgrößen u.a. aufgrund ihrer Laufzeiten nicht geeignet.

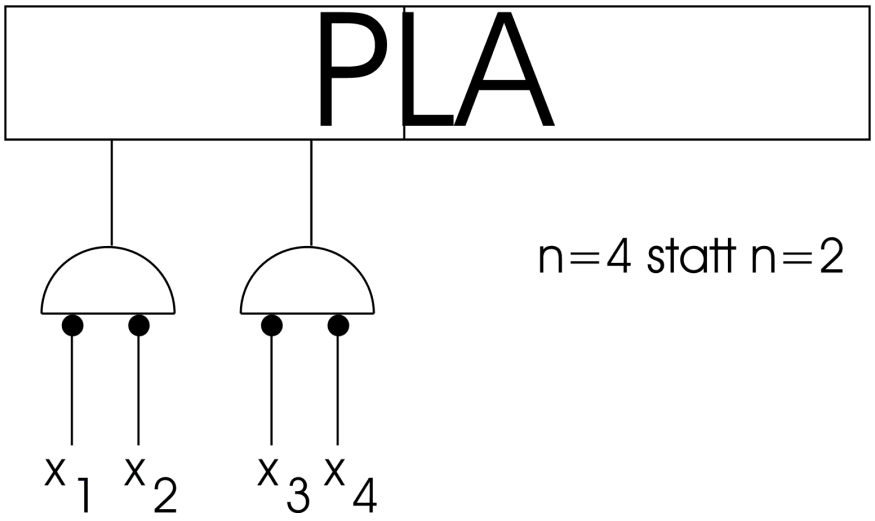
Ursache u.a. in der Generierung **aller** Primterme (exponentielles Wachstum mit der Anzahl der Eingangsvariablen).

Kodierung der Eingangsvariablen

Häufig nicht alle möglichen Werte der Eingabevariablen als Eingabe an ein PLA.

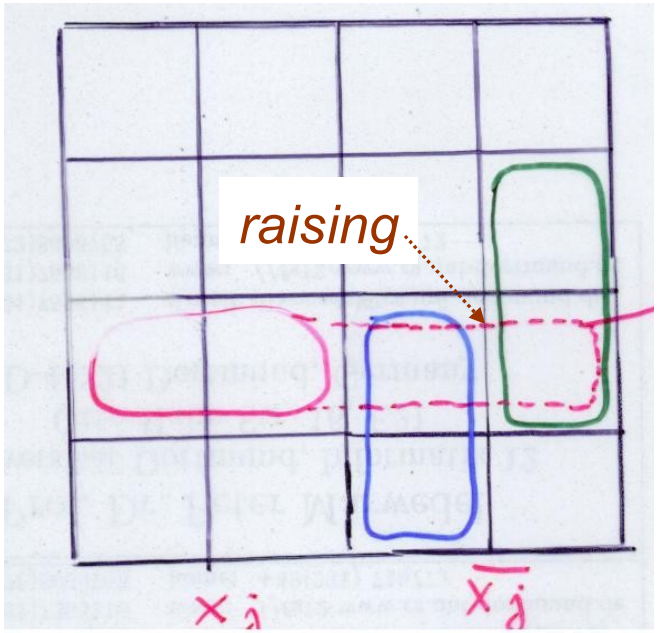
☞ Kodierung der Eingabevariablen mit kürzeren Bitmuster, um Spalten für Eingabevariablen einzusparen.

Einsatz bei Verfahren mit symbolischen Variablen.
Beispiel:



Streichen von Variablen in Termen

Schnelle Vereinfachungstechnik:
Suche nach Variablen, welche in einzelnen Termen redundant sind (=Übergang auf größere Blöcke von Einsen im KV-Diagramm).



Zu untersuchen, ob die Variable x_j in dem Term

$$x_j * f(x_1,(x_j),..,x_n)$$

überflüssig ist, der Term also durch

$$f(x_1,(x_j),..,x_n)$$

ersetzt werden kann.
Darin soll (x_j) bedeuten, dass f nicht x_j abhängt.

Zulässig, falls $\overline{x_j} * f(x_1,(x_j),..,x_n)$ durch die übrigen Terme bereits überdeckt wird. Ersetzung heißt **Term-Expansion** (engl. *raising*). Keine direkte Reduktion der Fläche. Ggf. entstehende redundante Zeilen.

Elimination redundanter Terme

Fläche ~ #(Zeilen), ☞ Fläche ~ #(Anzahl der Produktterme).
☞ **Elimination** redundanter PLA-Zeilen.
Zeile eines PLA kann entfernt werden, wenn alle logischen Einsen, die diese Zeile an den Ausgängen erzeugt, schon durch die übrigen Zeilen erzeugt werden (≙ Weglassen eines 1-Blockes, der durch andere Blöcke überdeckt wird). ☞

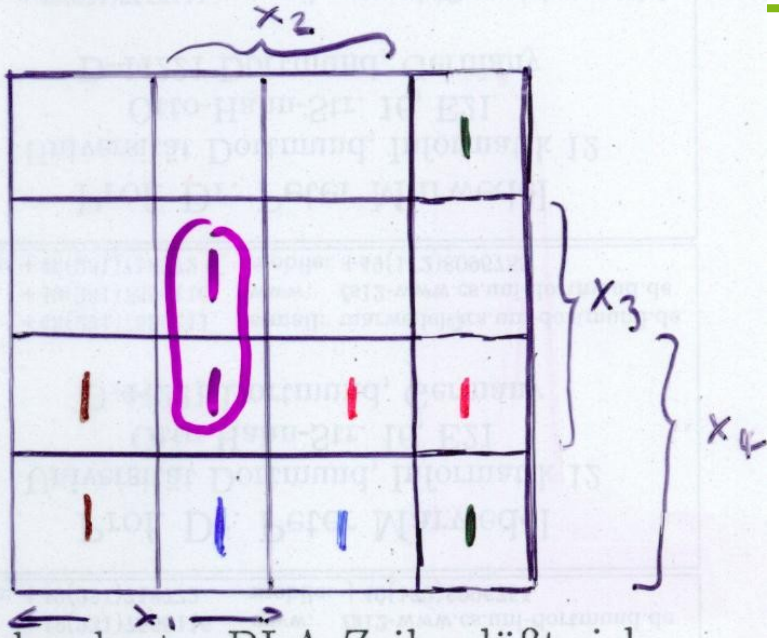
Lemma:
Zeile *row* ist redundant ⇔
$\forall\ c \in [1..m]$ mit $or(row,c)=1$ gilt:
$R := \{r \mid r \in [1..k], or(r,c)=1, r \neq row\}$ überdeckt die Zeile *row*

Beispiel

Beispiel:

and	*or*
2101	1
0002	1
0211	1
1112	1
1021	1

$x_2 \bar{x}_3 x_4$ ✓
$\bar{x}_1 \bar{x}_2 \bar{x}_3$ ✓
$\bar{x}_1 x_3 x_4$ ✓
$x_1 x_2 x_3$ ✓
$x_1 \bar{x}_2 x_4$ ✓



Durch "raising" und Elimination redundanter PLA-Zeilen läßt sich dieses zu einem PLA mit der folgenden Individualität vereinfachen:

and	*or*
2221	1
0002	1
1112	1

✓
✓

ALGORITHMUS SIEHE ULLMAN (GERINGERE KOMPLEXITÄT ALS KV)

7

ESPRESSO

Annahme: zu realisierende Funktion als Summe von Produkttermen gegeben.

Grober Ablauf des Verfahrens:

1. Expansion der Terme zu Primtermen
2. Extraktion essentieller Primterme,
   Elimination total redundanter Primterme
3. Behandlung der partiell-redundanten Terme
   a) Bestimmung einer minimalen Überdeckung
   b) Reduktion der Restterme
   c) erneute Expansion der Primterme,
      Entfernung von redundanten Primtermen

   bis keine redundanten Primterme mehr entfernt werden können

Unterschiede gegenüber dem klassischen Ansatz

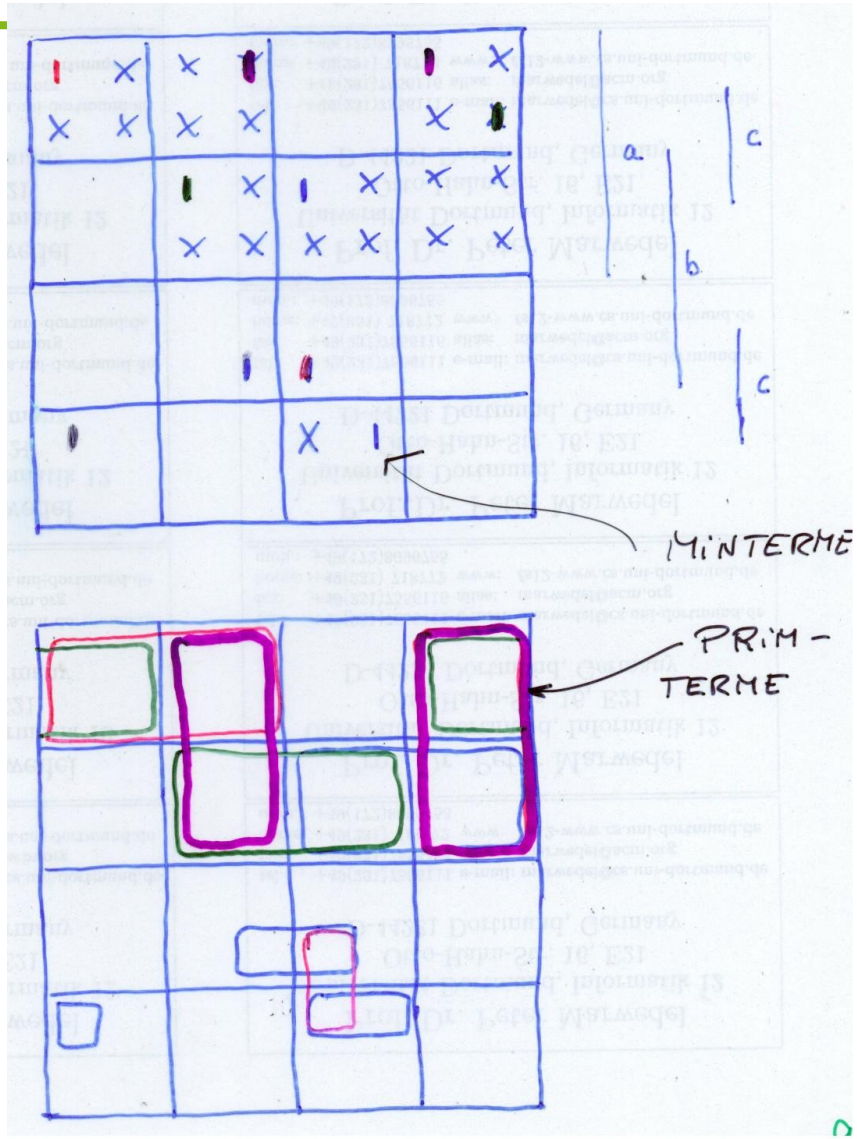
Zwei wesentliche Unterschiede gegenüber dem klassischen Ansatz:

- Verzicht auf die Erzeugung aller Primterme;
  nur Verallgemeinerung der gegebenen Terme zu Primtermen.
- Bestimmung kostengünstiger, aber nicht notwendig der optimalen Überdeckung.

☞ Einzelne Schritte des Verfahrens
(nur Darstellung mit KV-Tafeln;
die Algorithmen arbeiten auf Bitvektoren):

1. Expansion der Terme zu Primtermen

Man ersetzt nur jeden Term der Ausgangsüberdeckung durch einen aus diesem abgeleiteten Primterm.
Anzahl der Terme konstant.
Die Primterme überdecken nicht nur jene Werte der Eingangs-variablen, in denen die Funktion eine '1', liefern soll, sondern ggf. auch Argumentwerte, für die der Funktionswert redundant ist. Ziel: Primüberdeckung, deren Terme sich hochgradig überschneiden.



2. Extraktion essentieller Primterme

Die Terme teilt man nun in drei Klassen ein:

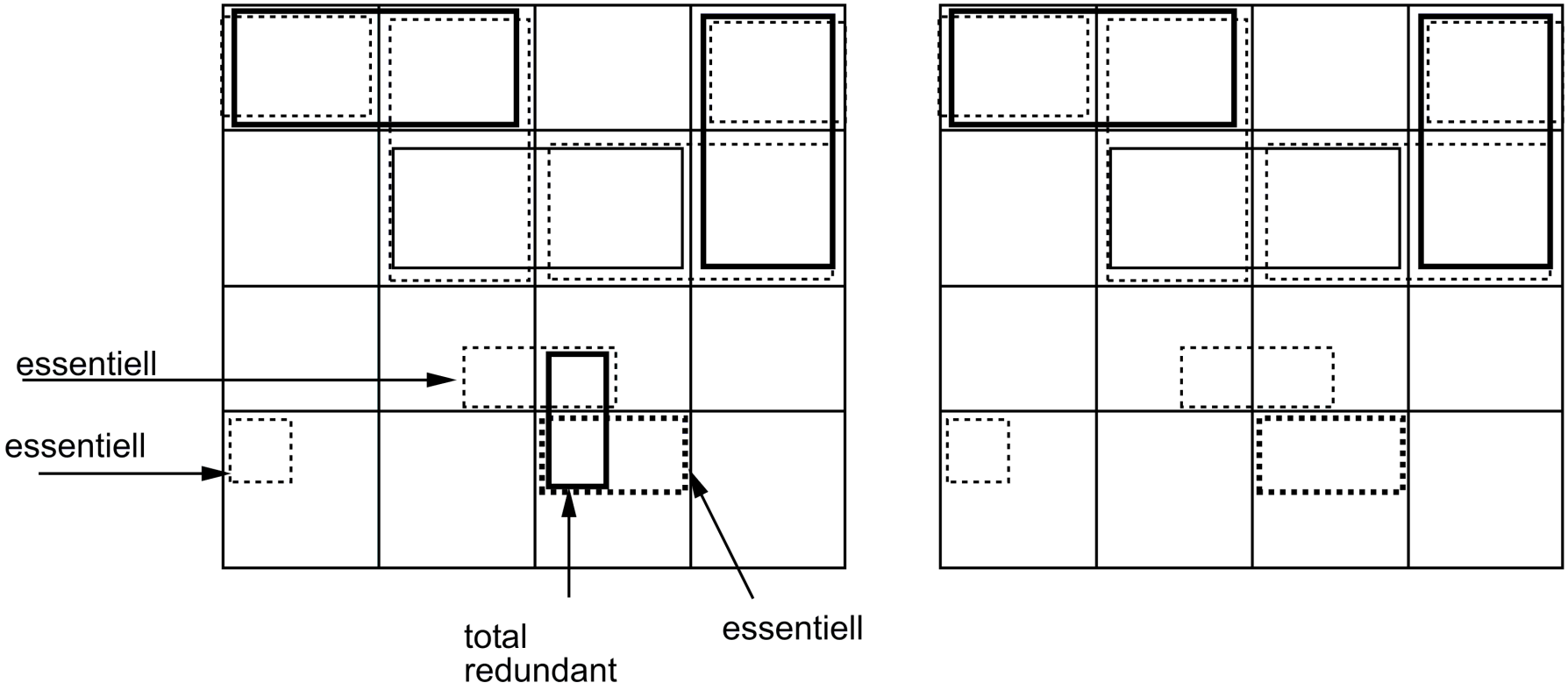
a) Ein Primterm, für den Argumentwerte existieren, an denen einzig dieser Term für eine Überdeckung sorgt, heißt **essentieller Primterm**.

b) Ein Primterm, welcher allein schon von den essentiellen Primtermen überdeckt wird, heißt **total redundant**.

c) Alle übrigen Terme heißen partiell redundant.

2. Total redundante Primterme

Total redundante Primterme können ohne Verlust an Allgemeinheit eliminiert werden:



3.a) Partiell redundante Terme

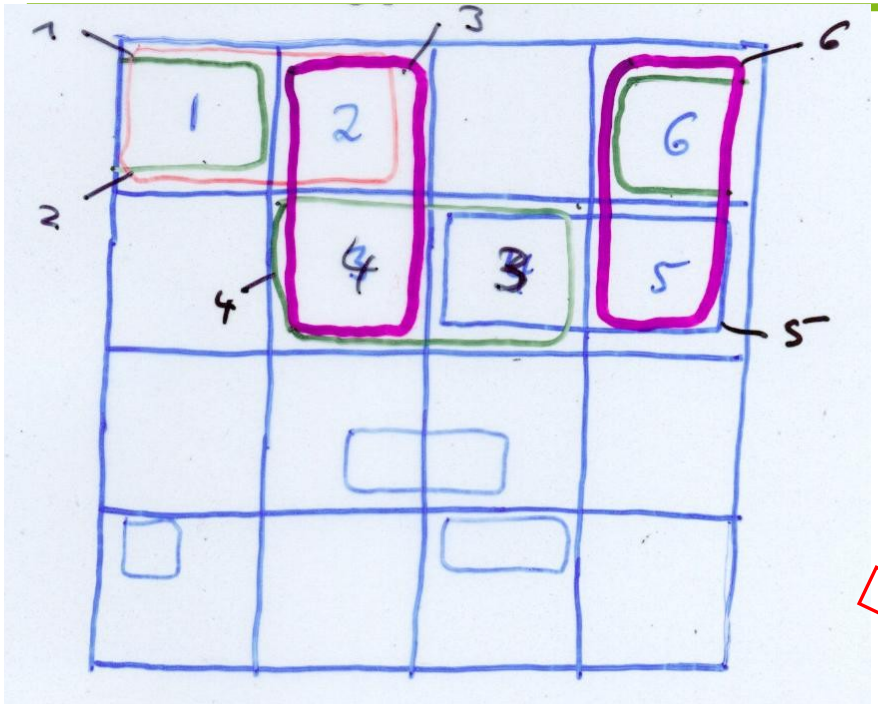
Partiell redundante Terme werden jeweils durch einen anderen Term überdeckt.

Einzelner partiell redundanter Term kann entfernt werden, ohne dass Überdeckungseigenschaft verloren geht.

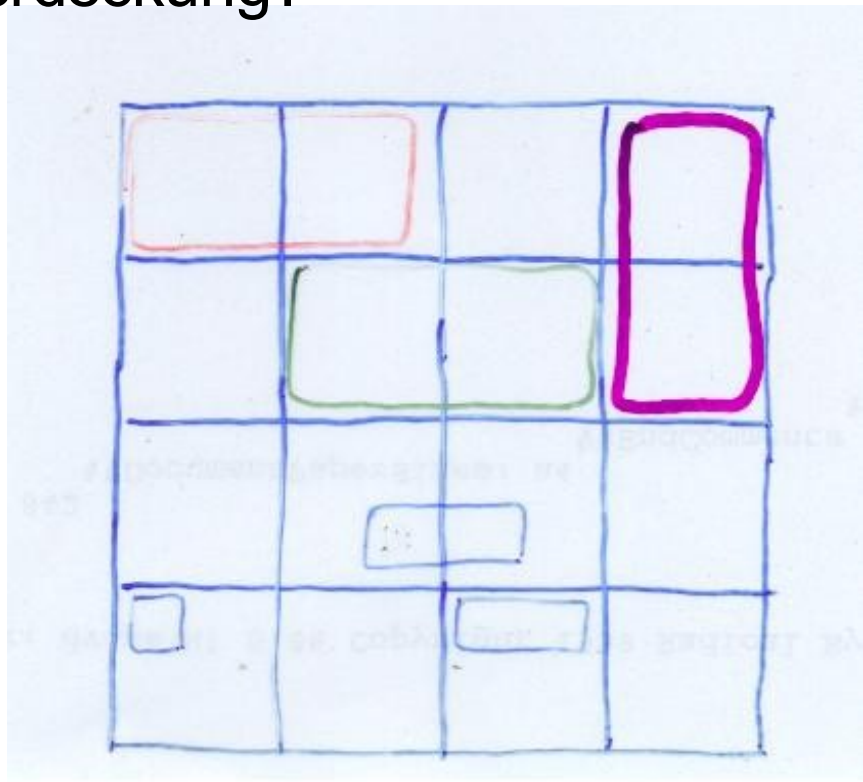
Für mehrere Terme gilt dieses nicht.

1. **Bestimmung einer Überdeckung**:
   Zur Bestimmung einer Auswahl von zu eliminierenden Termen, welche die Überdeckungseigenschaft erhält, benutzt ESPRESSO einen Algorithmus für das so genannte *minimum set covering problem*.

Bestimmung einer Überdeckung



$s_1=\{1,2\}$, $s_2=\{2,3\}$, $s_3=\{4,5\}$,
$s_4=\{3,4\}$, $s_5=\{5,6\}$, $s_6=\{6,1\}$
Überdeckung?



3.b-c) Iteration

a) Reduktion der Restterme
b) Erneute Expansion der Primterme, Entfernung von redundanten Primtermen: Nach der Reduktion liefert eine erneute Expansion in der Regel wieder andere Primterme, für die die bislang angeführte Prozedur wiederholt werden kann.

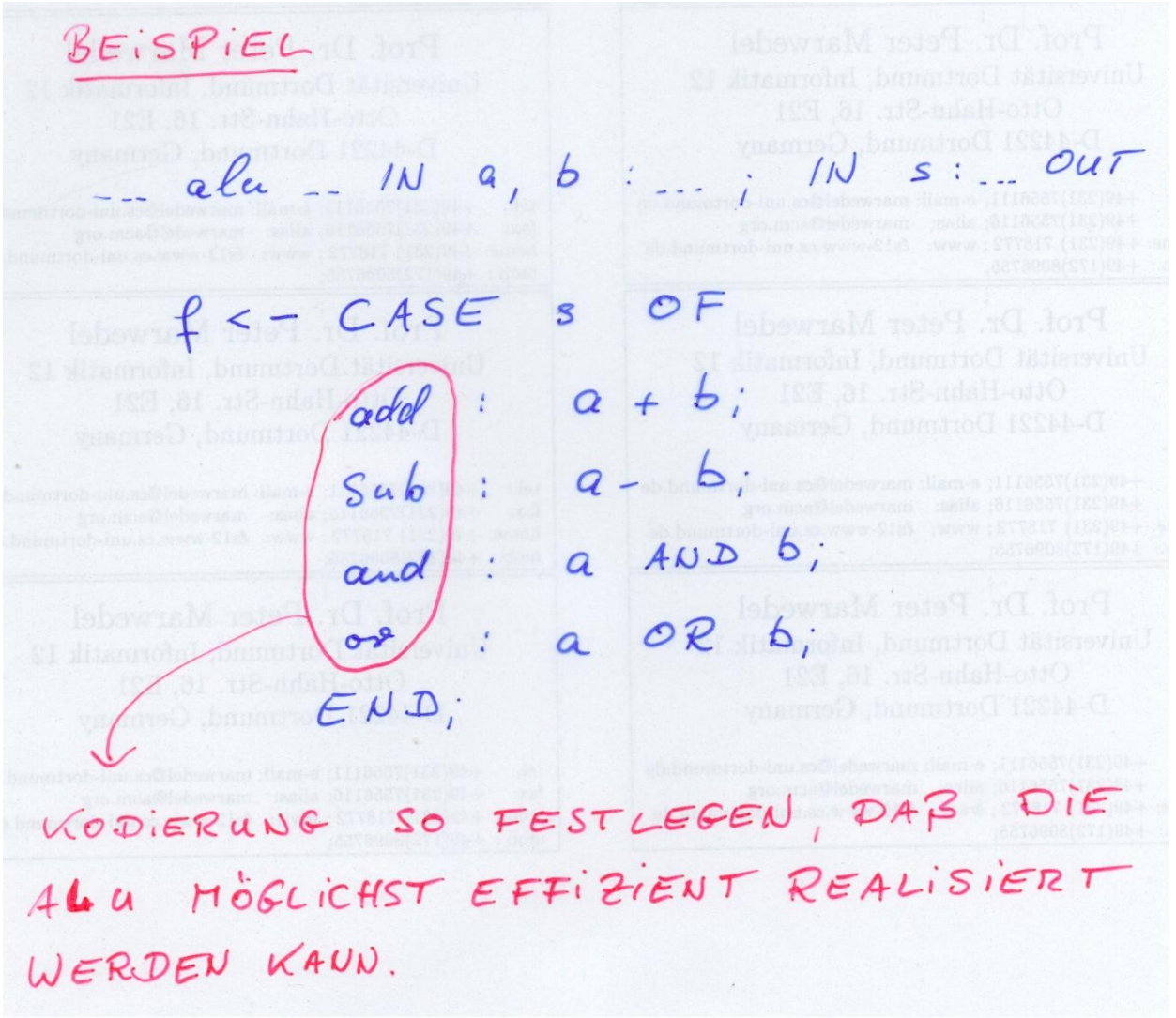
Keine Garantie global optimaler Auswahl; zur Verbesserung Ausführung der o.a. Schritte in einer Schleife:
GOTO a) bis keine redundanten Primterme mehr entfernt werden können.

Gefundene Lösung hinsichtlich der Minimierung der Anzahl der Produkterme nicht notwendig minimal, da u.a. nicht alle Reihenfolgen der Termreduktion untersucht werden.

Erweiterung: ESPRESSO-MV

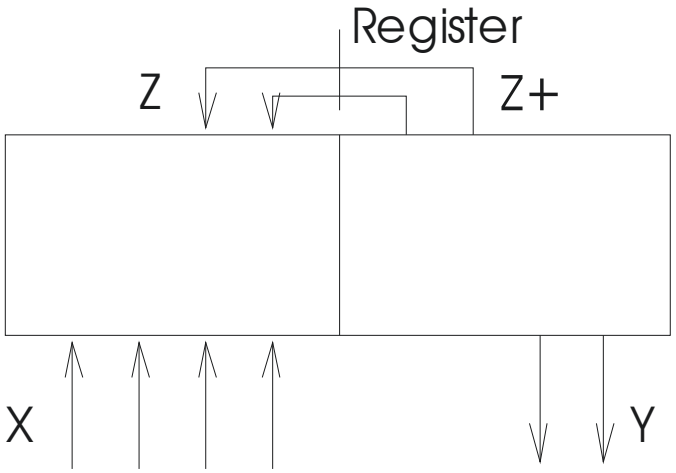
Statt Boolescher Argument-Variablen werden auch mehrwertige und symbolische Variable betrachtet.

Vorteil: Kodierung kann durch das Minimierungs-werkzeug fest gelegt werden.



Weitere Optimierungen für PLAs: Faltung der AND-Plane

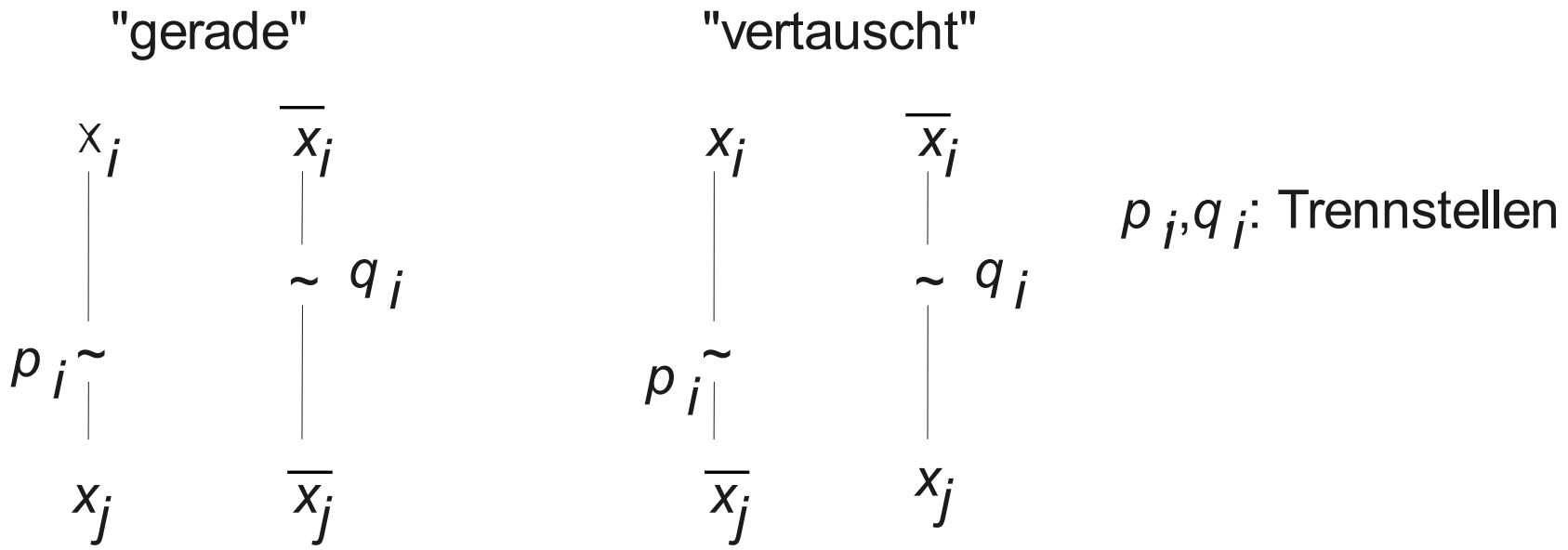
Viele Terme nur von Teil der Eingangsvariablen abhängig.
Man unterbricht vertikale Leitungen der AND-Plane und führt von oben und von unten zu.
Beispiel: Zustandsbits problemlos von oben zuzuführen:



☞ Flächenreduktion von PLAs durch Faltung (*(column) folding*).
Bei Vorgaben hinsichtlich der Zuführung: *constrained (column) folding*.
Einschränkung im Folgenden: invertierte und nicht-invertierte Eingangsvariable stets von derselben Seite zugeführt.

Polaritäten

Zwei Möglichkeiten: Spalten entweder für Signale gleicher oder gegensätzlicher Polarität



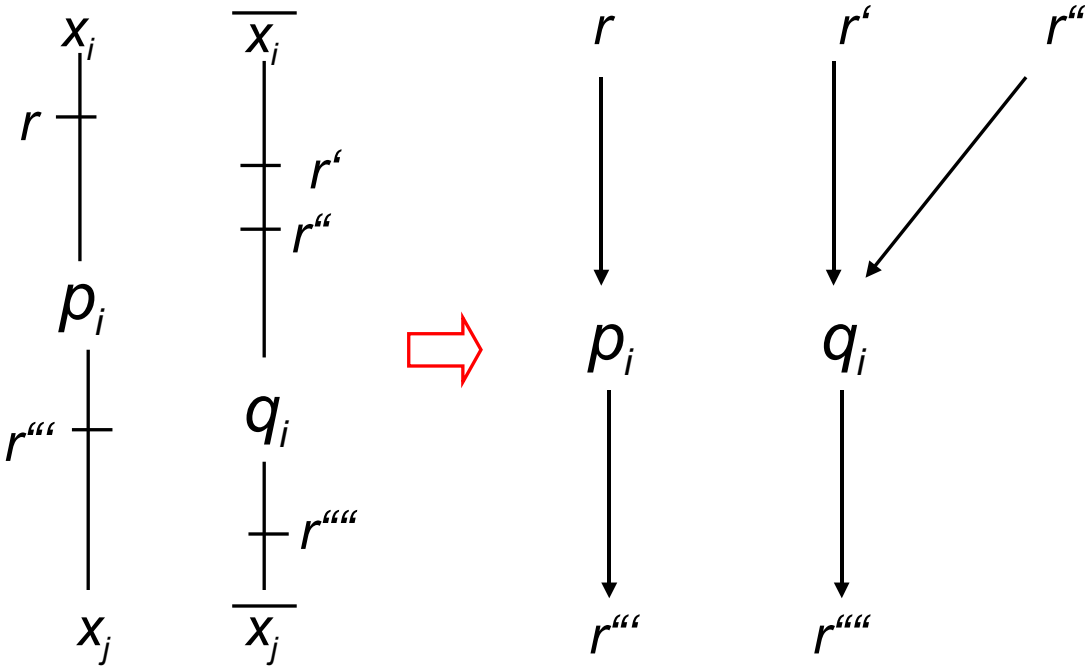
Charakterisierung der Trennstellen p_i,q_i durch einfache Indizierung: Index der oberen Variablen.

Zulässigkeit einer partiellen Lösung

Graph, mit dem die Zulässigkeit einer partiellen Lösung überprüft werden kann:

- Je einen Knoten für jeden Produktterm,
- je einen Knoten, der die Trennstelle zwischen den beiden vertikalen Leitungssegmenten symbolisiert,
- gerichtete Kante vom Knoten n_i zum Knoten n_j genau dann, wenn das durch n_i repräsentierte Objekt oberhalb von dem durch n_j repräsentierten Objekt anzuordnen ist.

Beispiel



Graph zyklenfrei → partielle Ordnung; Anordnung kann in einer dazu kompatiblen totalen Ordnung erfolgen (topologisches Sortieren). Test auf Zyklenfreiheit bildet Kern des Greedy-artigen Algorithmus.

Algorithmus, der Variable gruppiert

- Initialisierung mit Knoten pro Produktterm.
- Untersuchung von Paaren von Eingangsvariablen:
- Jede Variable: oben/unten und gerade/vertauscht zuführbar.
- Zurückweisung von (x_i und x_j) im geraden Fall, falls ein Produktterm existiert, der sowohl x_i wie auch x_j oder beide negierten Variablen benötigt.
- Entsprechend für den vertauschten Fall.

Zulässigkeit eines Paares

Zulässigkeit eines Paares von x_i und x_j:
temporäre Erweiterung um p_i, q_i, Erzeugung folgender Kanten für „oben und gerade“:

- Für alle Produktterme r, zu deren Berechnung x_i benötigt wird, eine Kante von r nach p_i.
- Für alle Produktterme r, zu deren Berechnung x_j benötigt wird, eine Kante von p_i nach r.
- Für alle Produktterme r, zu deren Berechnung $\neg x_i$ benötigt wird, eine Kante von r nach q_i.
- Für alle Produktterme r, zu deren Berechnung $\neg x_j$ benötigt wird, eine Kante von q_i nach r.

Generierter temporärer Graph zyklenfrei?

Algorithmus

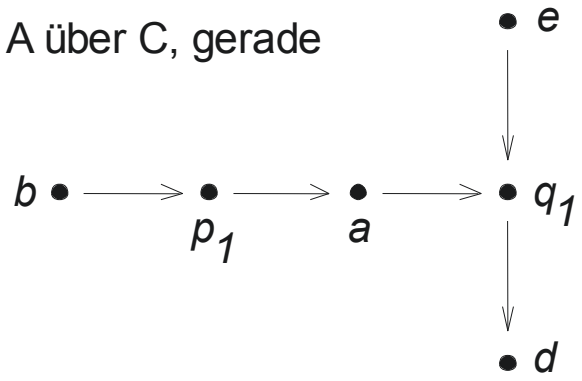
Graph G:=(V:= Produktterme; E:={}); $Used$:= {};
for all $i,j \in [1..n]$-$Used$ **do**
  for i oben, i unten **do**
    for gerade, vertauscht **do**
      if ex. keine konfliktbehaftete Zeile **then begin**
        V':=$V \cup \{p_i, q_i\}$; E':=E;
        Für i oben und gerade:
          $\forall$ r mit *and* (r,i)=1: E':= E' $\cup$ (r,p_i);
          $\forall$ r mit *and* (r,j)=1: E':= E' $\cup$ (p_i,r);
          $\forall$ r mit *and* (r,i)=0: E':= E' $\cup$ (r,q_i);
          $\forall$ r mit *and* (r,j)=0: E':= E' $\cup$ (q_i,r);
        Entsprechend für i unten oder vertauscht.
        if G'=(V',E') ist zyklenfrei **then**
          begin G:=G'; E:=E'; $Used$:=$Used \cup \{i,j\}$ **end**; **end**;

Beispiel (nach Ullman [Ull84])

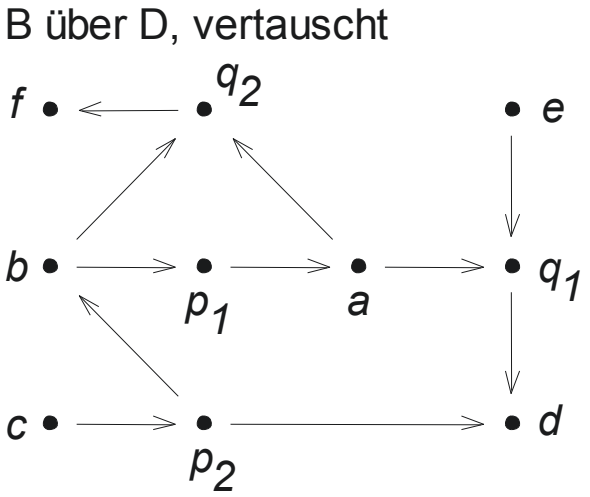
and	A	B	C	D	E
a	0	0	1	2	2
b	1	0	2	0	2
c	2	1	2	2	0
d	2	2	0	0	1
e	0	2	2	2	2
f	2	2	2	1	1

A-E: Eingangsvariable;
a-f: Produktterme.

Wegen Term a keine gerade Paarung von A und B.
Wegen Term b keine vertauschte Paarung.



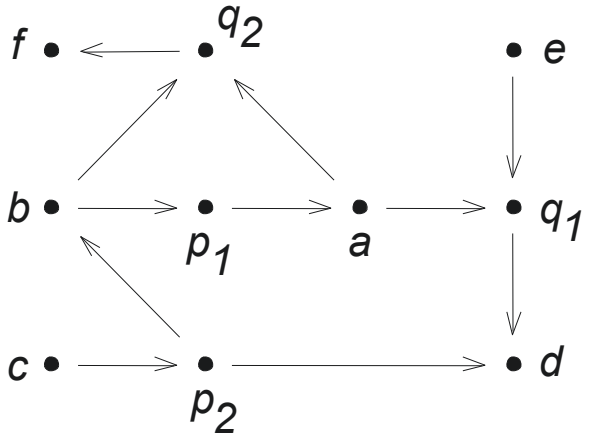
Wegen Term b keine gerade Paarung von B und D.

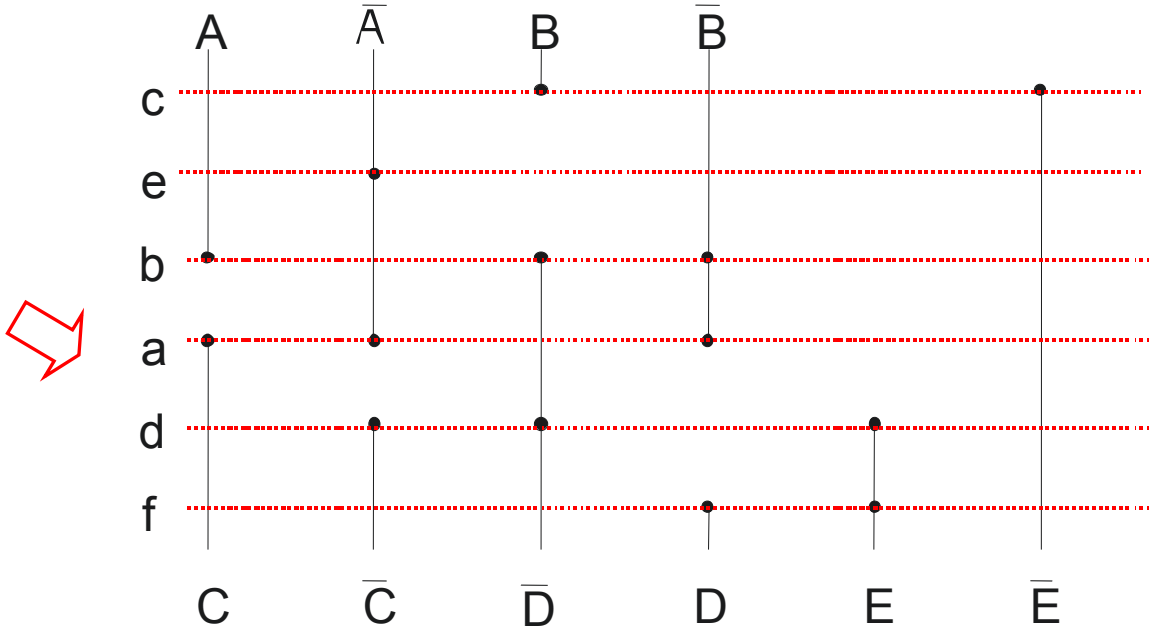


Keine weitere Paarungen.

Lösung

B über D, vertauscht

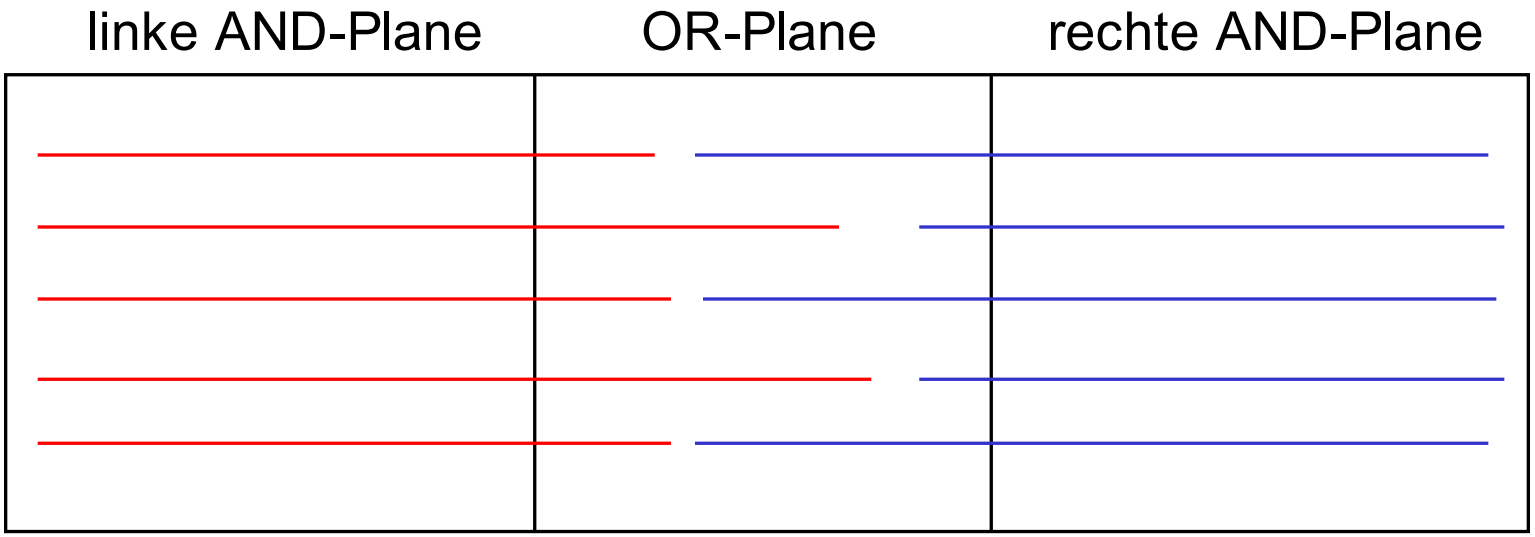




Keine Überprüfung aller Lösungen, daher kein optimaler Algorithmus. ☞"*Branch-and-Bound*"-Verfahren.

Faltung der OR-Plane

Neben der AND-Plane kann auch die OR-Plane gefaltet werden (*row folding*):



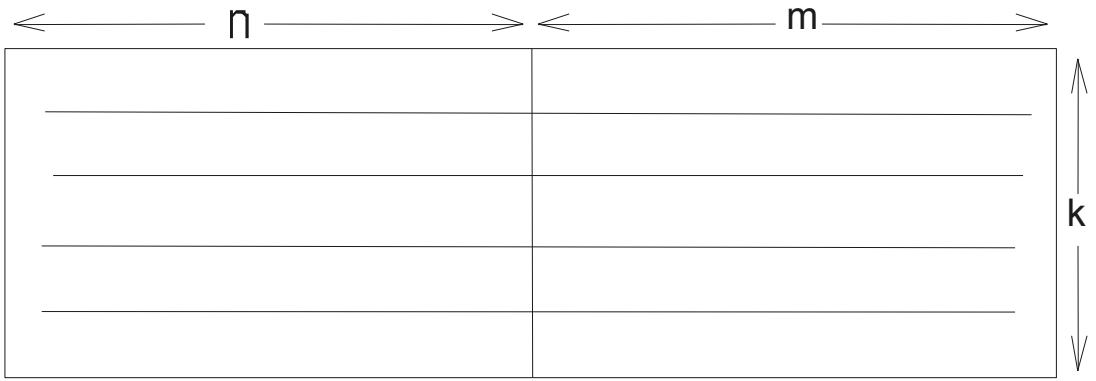
Für diese Faltung können prinzipiell die gleichen Algorithmen wie bei der Faltung der AND-Plane verwendet werden, jedoch entfällt die Behandlung von negierten Variablen.

Partitionierung von PLAs

Neben der Faltung kann auch die Partitionierung eines PLAs in ≥ 2 kleinere PLAs einen Flächengewinn bringen. Neben direkter Flächenreduktion vielfach Flächenreduktion, da kleinere PLAs leichter in ein Gesamtlayout zu integrieren sind als ein großes.

Die Fläche eines PLAs mit *k* Zeilen, *n* Eingangs- und *m* Ausgangsvariablen ist proportional zu

$$S = k * (n+m);$$

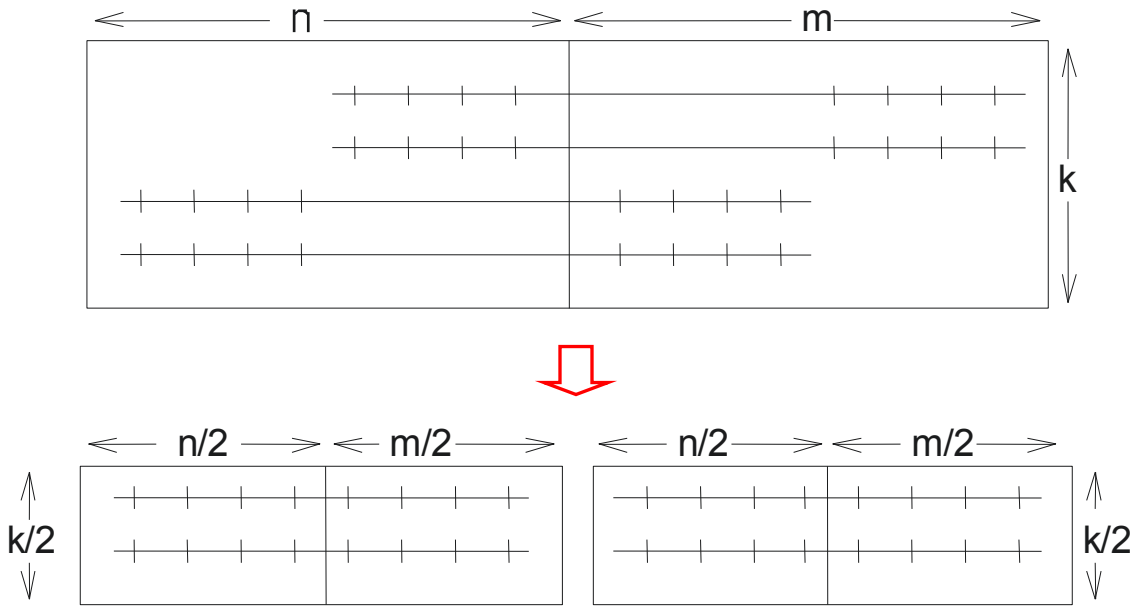


Partitionierung von PLAs (2)

Nach Partitionierung in 2 Teile ist die Fläche proportional zu

$$S' = \Sigma_i \, k_i * (n_i + m_i).$$

Falls $\forall i \in [1..2]$: $m_i = m/2$, $n_i = n/2$, $k_i = k/2$,
so wird Flächenreduktion um die Hälfte ($S' = S/2$) erreicht.



Zusammenfassung

Minimierung 2-stufiger Logik

- PLAs
- Einfache Minimierungstechniken
- ESPRESSO
- Faltung von PLAs
  - *column folding*
  - Partitionierung und *row folding*