Mehrstufige Logiksynthese unter Ausnutzung funktionaler Eigenschaften

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.–Ing.)
der Technischen Fakultät
der Universität des Saarlandes

von

Christoph Scholl

Saarbrücken
1996

Tag des Kolloquiums:	14. 04. 1997
Dekan:	Prof. Dr.–Ing. Alexander Koch
Gutachter:	Prof. Dr. rer. nat. Dr. hc. mult. Günter Hotz
	Prof. Dr. rer. nat. Bernd Becker

Inhaltsverzeichnis

Einleitung

In den letzten Jahren stieg die Anzahl der Anwendungsgebiete für integrierte Schaltungen ständig. Das Spektrum der Anwendungsgebiete reicht von Massenprodukten des täglichen Gebrauchs über industrielle Steuerungen bis zu Spezialanwendungen wie Höchstleistungsrechnern oder Schaltungen für die Raumfahrt.

Sowohl wachsende Integrationsdichte als auch Automatisierung des Entwurfs machten es erst möglich, für zahlreiche Aufgaben den Einsatz von mit universellen Mikroprozessoren ausgestatteten Leiterplatten zu ersetzen durch anwendungsspezifische Spezialschaltkreise (ASIC's = Application Specific Integrated Circuits). Durch geschickte anwendungsspezifische Integration von Schaltelementen auf einem oder mehreren Chips können so spezielle Funktionen eventuell mit weniger Komponenten realisiert werden. Der Einsatz entsprechender ASIC's führt dann zu einer Reihe von Vorteilen: geringere Systemkosten, geringeres Gewicht bzw. Volumen, größere Nachbausicherheit, durch Reduktion externer Verbindungen zwischen verschiedenen Bauteilen außerdem höhere Zuverlässigkeit, geringerer Leistungsverbrauch und höhere Geschwindigkeit des Gesamtsystems (durch Verringerung der zu treibenden Lasten). Aus diesen Gründen steigt der Anteil von ASIC's am Weltumsatz integrierter Schaltungen stetig an [ICE91] (siehe Abbildung 0.1).

Da anwendungsspezifische Schaltkreise meist nur in geringen Stückzahlen produziert werden, kommt es immer mehr zu einem Übergewicht der *Entwurfs*kosten gegenüber den

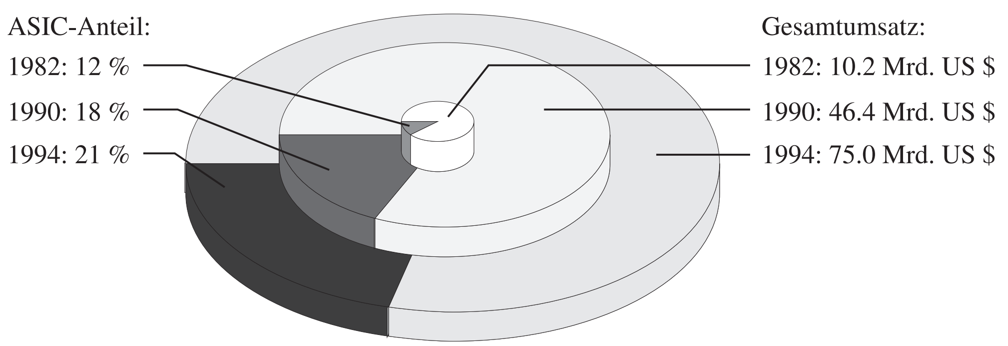


Abbildung 0.1: Anteil von ASIC's am Weltumsatz integrierter Schaltungen.

*Fertigungs*kosten eines Chips. Kurze Entwurfszeiten und damit niedrige Entwurfskosten können aber nur dann erreicht werden, wenn möglichst große Teile des Entwurfsprozesses automatisiert werden, d.h. rechnergestützt ablaufen. Sogar bei Standardschaltungen wie Mikroprozessoren besteht durch einen zunehmenden Kostendruck und die starke Abhängigkeit des Produkterfolgs vom Zeitpunkt des Markteintritts die Notwendigkeit eines schnellen und kostengünstigen Entwurfs. Auch aufgrund der wachsenden Integrationsdichte und der damit verbundenen Möglichkeit, eine wesentlich komplexere Funktionalität auf einem einzigen Chip zu realisieren, gewinnen klar strukturierte und automatisierte Entwurfsabläufe immer mehr an Gewicht.

Die grundsätzliche Aufgabe des Schaltungsentwurfs besteht darin, eine Schaltungs*spezifikation* in eine korrekte Schaltungs*implementierung* umzusetzen. Der Entwurfsablauf läßt sich grob in 5 Phasen gliedern, in den funktionalen Entwurf, die Logiksynthese, die Generierung eines Fertigungstests, die Layoutkonstruktion und die Validierung:

1. In der Phase des funktionalen Entwurfs wird die Spezifikation des Systemverhaltens definiert. Diese Spezifikation kann beispielsweise mit Hilfe einer Hardwarebeschreibungssprache (HDL) oder mit Hilfe graphischer Modelle erfolgen.

2. Im Rahmen der Logiksynthese wird ausgehend von der Spezifikation, die der funktionale Entwurf liefert, eine geeignete Implementierung generiert. Ergebnis der Logiksynthese ist eine Netzlistenbeschreibung über einer bestimmten Menge von Grundzellen.

3. Bei der Generierung eines Fertigungstests werden Testeingaben bestimmt, die dazu dienen, nach der Fertigung fehlerhaft produzierte Chips auszusortieren.

4. Bei der Layoutkonstruktion erfolgt die genaue Plazierung und Verdrahtung der Grundzellen.

5. In der Validierungsphase wird überprüft, ob die erhaltene Schaltungsimplementierung mit der Spezifikation übereinstimmt.

Der im Rahmen der vorliegenden Arbeit behandelte Themenkomplex ist im wesentlichen der 2. Phase der obigen Aufzählung zuzurechnen. Die Arbeit befaßt sich mit der automatischen Synthese kombinatorischer Teilschaltungen, d.h. mit dem Entwurf von Realisierungen zu Booleschen Schaltfunktionen $f : \{0,1\}^n \to \{0,1\}^m$. Zu gegebenen Schaltfunktionen sollen Schaltkreise gefunden werden, die diese Funktionen realisieren und möglichst geringe Kosten haben. Als Kostenmaße kommen hierbei z.B. die benötigte Schaltkreisfläche oder die Laufzeit der Schaltung in Frage. Beim Entwurf sequentieller Schaltungen, d.h. von Schaltungen, die speichernde Elemente enthalten, lassen sich nach Trennung der Gesamtschaltung in Kombinatorik und speichernde Elemente die rein kombinatorischen Komponenten mit Hilfe der hier vorgestellten Methoden generieren. Durch Minimierung der Gesamtfläche des entworfenen Chips lassen sich mehrere Ziele erreichen: Zum einen werden die Herstellungkosten eines einzelnen Chips gesenkt, da nun mehr Chips auf einem Wafer Platz finden. Zum anderen wird die Ausbeute bei der Fertigung erhöht, da die Defektrate pro Chip mit zunehmender Chipfläche drastisch ansteigt.

Abhängig von dem zu lösenden praktischen Problem sind bei der Logiksynthese die verschiedensten Schaltfunktionen zu realisieren. *Eine* Eigenschaft haben die in der Praxis zu realisierenden Schaltfunktionen jedoch zumeist gemeinsam: Sie unterscheiden sich in ihrem Aufbau wesentlich von Funktionen, deren Funktionstabellen zufällig ausgewürfelt wurden. Die in der Praxis auftretenden Funktionen sind in der Regel *nicht* durch (für große Eingangsvariablenzahlen *riesige*) Funktionstabellen gegeben, sondern entspringen einem praktischen Problem und genügen daher relativ einfachen und kurzen Bildungsgesetzen. Daher weisen solche Schaltfunktionen meistens gewisse Regelmäßigkeiten bzw. Struktureigenschaften auf. Bei der Logiksynthese (d.h. beim Entwurf von kombinatorischen Schaltungen zu diesen Funktionen) müssen solche Struktureigenschaften ausgenutzt werden, um zu guten Realisierungen zu kommen.

Mehrere Gründe sprechen für eine *automatische* Durchführung der Logiksynthese:

- Der Handentwurf guter Realisierungen ist häufig mühsam und zeitaufwendig.
  Der Aufwand zum Entwurf einer Schaltung läßt sich durch automatische Logiksynthese erheblich reduzieren.

- Es sollen auch Anwender, die keine Experten auf dem Gebiet der Schaltkreissynthese sind, in die Lage versetzt werden, Entwürfe durch eine einfache Spezifikation des gewünschten Ein–/Ausgabeverhaltens durchzuführen.

- Häufig enthalten die zu realisierenden Schaltfunktionen zwar Regelmäßigkeiten, aber diese Regelmäßigkeiten sind nicht auf den ersten Blick zu erkennen. Auf diese Weise wird es für den Menschen schwierig, eine Logiksynthese unter Ausnutzung vorhandener Struktureigenschaften durchzuführen. In Fällen, in denen die Struktur der zu realisierenden Booleschen Funktion schwer überschaubar ist, sind maschinelle Entwürfe Handentwürfen oftmals qualitativ überlegen.

- Handentwürfe komplexerer Boolescher Funktionen sind sehr fehleranfällig. Durch Verwendung eines (korrekten) Logiksynthesewerkzeuges kann dagegen garantiert werden, daß eine fehlerfreie Umsetzung einer Spezifikation in eine korrekte Implementierung erfolgt.

Häufig beschränkt man sich bei der automatischen Logiksynthese zur Realisierung von Schaltfunktionen auf zweistufige Lösungen. Dies hat im wesentlichen zwei Gründe: Einerseits lassen sich solche zweistufigen Lösungen durch programmierbare logische Felder (PLA's) leicht umsetzen, andererseits gibt es mittlerweile relativ ausgereifte heuristische Verfahren (z.B. ESPRESSO [BHM+84]), die in der Lage sind, kostengünstige zweistufige Realisierungen zu finden. Allerdings macht man die Beobachtung, daß in der Praxis sehr einfache Schaltfunktionen vorkommen, bei denen auch die beste zweistufige Realisierung *sehr groß* wird. Daher macht sich die Einschränkung des Suchraumes auf zweistufige Realisierungen oft negativ bemerkbar. Aus diesem Grund befaßt sich die vorliegende Arbeit mit *mehrstufigen* Realisierungen.

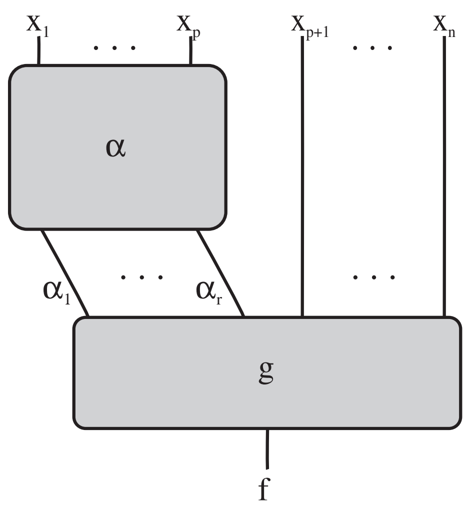
Bei den Werkzeugen zur (automatischen) mehrstufigen Logiksynthese sind im wesentlichen drei Richtungen zu unterscheiden: Verfahren, die auf der Anwendung lokaler Transformationsregeln basieren (z.B. [DJBT81, DBJT84]), Verfahren, die auf Methoden der zweistufigen Logiksynthese basieren und durch Ausfaktorisieren mehrstufige Realisierungen erzeugen (der zur Zeit bekannteste Vertreter dieses Ansatzes ist *misII* [BRS+87] bzw. *sis* [S+92]) und Verfahren, die Funktionen durch (rekursive) Zerlegungen realisieren.

Das im Rahmen dieser Arbeit entwickelte Verfahren beruht auf der Durchführung rekursiver Zerlegungen.

Logiksynthese unter Ausnutzung von Symmetrien und nichttrivialen Zerlegungen

Das hier entwickelte rekursive Zerlegungsverfahren nutzt im wesentlichen zwei wichtige Struktureigenschaften Boolescher Funktionen aus, nämlich Symmetrien (z.B. Invarianz einer Booleschen Funktion gegenüber der Vertauschung von Eingangsvariablen) und nichttriviale Zerlegbarkeit.

Die Durchführung von disjunkten Zerlegungen bei der Logiksynthese wurde schon in Arbeiten von Ashenhurst [Ash59], Curtis [Cur61] und Karp [Kar63] erstmals betrachtet. Eine disjunkte Zerlegung einer Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$ mit den Eingangsvariablen $x_1, \ldots, x_n$ ist eine Darstellung von folgender Form:



Die Eingangsvariablen $x_1, \ldots, x_n$ werden hierbei in zwei disjunkte Mengen $\{x_1, \ldots, x_p\}$ (die Menge der „gebundenen“ Variablen) und $\{x_{p+1}, \ldots, x_n\}$ (die Menge der „freien“ Variablen) aufgeteilt. Es treten „Zerlegungsfunktionen“ $\alpha_1, \ldots, \alpha_r$ auf. Die Funktion g wird als „Zusammensetzungsfunktion“ bezeichnet. Es werden Zerlegungen mit möglichst wenigen Zerlegungsfunktionen gesucht (auf jeden Fall sollte $r < p$ sein). Dies hat folgende Vorteile:

- Ist die Anzahl der Zerlegungsfunktionen gering, so hat man bei rekursiver Anwendung des Verfahrens weniger (Zerlegungs-)Funktionen zu realisieren. Es besteht die Hoffnung, daß die Gesamtkosten des resultierenden Schaltkreises somit möglichst gering bleiben.

- Je weniger Zerlegungsfunktionen bei einer Zerlegung auftreten, desto weniger Eingänge hat die Zusammensetzungsfunktion g. Wenn g weniger Eingänge hat, dann kann man hoffen, daß die Kosten zur Realisierung von g auch geringer sind.

- Aus der Netzliste, die die Logiksynthese liefert, wird das Layout eines Schaltkreises bestimmt, der die gesuchte Funktion realisiert. Häufig beobachtet man Schaltungen, bei denen die Fläche des Layouts nicht unbedingt durch die Anzahl der zu realisierenden Gatter bestimmt wird, sondern durch die Anzahl der zu realisierenden globalen Verbindungsleitungen. Eine Zerlegung mit wenig Zerlegungsfunktionen legt gerade eine Realisierung mit wenig globalen Verbindungsleitungen nahe.

Es folgt eine Auflistung wichtiger Problemstellungen und Lösungen, die die vorliegende Arbeit zu einer entscheidenden Weiterentwicklung der Ansätze von Ashenhurst, Curtis und Karp machen:

- Eine fundamentale Aufgabe der mehrstufigen Logiksynthese ist die Identifizierung von Teillogik, die in mehreren Schaltungsteilen mit Vorteil verwendet werden kann. Gerade auf der Tatsache, daß gleiche Teilschaltungen nur einmal realisiert, jedoch mehrfach verwendet werden, beruht häufig die Effizienz guter Realisierungen Boolescher Funktionen.

  Aus diesem Grund ist es von großer Bedeutung, die Identifizierung mehrfachverwendbarer Teillogik in das Zerlegungsverfahren zu integrieren.

  Im Rahmen dieser Arbeit gelingt dies durch die Ausnutzung von Freiheiten bei der Wahl von Zerlegungsfunktionen: Bei der Realisierung von Funktionen mit mehreren Ausgängen $f_1, \ldots, f_m$ werden *gemeinsame* Zerlegungsfunktionen für $f_1, \ldots, f_m$ berechnet. Man hat dabei das Problem zu lösen, die Zerlegungsfunktionen so zu berechnen, daß möglichst viele von ihnen in der Zerlegung möglichst vieler Ausgangsfunktionen f_i verwendet werden können.[1]

[1] Das Vorgehen ist nicht nur bei der Bearbeitung von Booleschen Funktionen mit *mehreren* Ausgängen von Bedeutung. Auch wenn eine Funktion ursprünglich nur einen Ausgang hat, so treten bei der rekursiven Anwendung des Zerlegungsverfahrens auf ihre Zerlegungsfunktionen im allgemeinen Funktionen mit mehreren Ausgängen auf, so daß die Verfahren zur Identifizierung mehrfachverwendbarer Teillogik auch hier Anwendung finden.

- Die in praktischen Problemen vorkommenden Booleschen Funktionen sind keine Funktionen mit „zufällig gewürfelter Funktionstabelle“, sondern zeichnen sich durch bestimmte Struktureigenschaften aus, so daß ihre Komplexität weit unter der Durchschnittskomplexität über alle möglichen Booleschen Funktionen mit der gleichen Anzahl von Eingängen liegt. Solche Struktureigenschaften sollen im Rahmen eines Logiksyntheseverfahrens nach Möglichkeit beibehalten werden.

  Die in dieser Arbeit vorgestellten Verfahren erreichen dieses Ziel durch eine Einschränkung des Suchraumes auf eine bestimmte Teilklasse von Zerlegungsfunktionen, die *strikten* Zerlegungsfunktionen. Auf diese Weise soll verhindert werden, daß in einer Zerlegung Zerlegungsfunktionen gewählt werden, die keine solchen Struktureigenschaften aufweisen und somit mit hoher Wahrscheinlichkeit eine hohe Komplexität besitzen. Für strikte Zerlegungsfunktionen läßt sich nämlich zeigen, daß sich Struktureigenschaften (wie z.B. Symmetrie in Paaren von Variablen oder Unabhängigkeit von Variablen) von der zerlegten Funktion auf die Zerlegungsfunktionen übertragen.

- Symmetrien Boolescher Funktionen spielen bei der Logiksynthese eine große Rolle. Symmetrische Funktionen sind im allgemeinen sehr viel leichter zu realisieren als allgemeine Funktionen (beispielsweise benötigt man für eine totalsymmetrische Funktion mit n Eingängen im worst case höchstens linear viele 2–Input–Gatter [Weg89], für allgemeine Funktionen jedoch $\theta(\frac{2^n}{n})$ [Sha49]).

  Es stellt sich die Frage, wie Symmetrien in Zusammenhang mit Zerlegungen ausgenutzt werden können.

  Dabei stellt sich heraus, daß es von Vorteil ist, die Menge der *gebundenen* Variablen ($\{x_1, \ldots, x_p\}$ im obigen Beispiel) so zu wählen, daß möglichst viele Symmetrien in dieser Menge auftreten. Symmetrien in der Menge der gebundenen Variablen können zu einer erheblichen Reduzierung der Anzahl der benötigten Zerlegungsfunktionen führen.
  Bei partiellen Funktionen kommt in Zusammenhang mit Symmetrien die zusätzliche Freiheit ins Spiel, daß verschiedene Fortsetzungen partieller Funktionen zu totalen Funktionen im allgemeinen auch verschiedene Symmetrieeigenschaften aufweisen.

- Bei praktischen Problemen treten häufig *partielle* Funktionen auf (d.h. Funktionen, deren don't care–Menge nicht leer ist), da die Schaltkreise, die diese partiellen Funktionen realisieren sollen, beispielsweise in ein größeres System eingebettet werden und man daher gewisse Eingabevektoren von vornherein ausschließen kann oder da im praktischen Problem bei einer Funktion mit mehreren Ausgängen für bestimmte Eingaben der Funktionswert nur an einem Teil der Ausgänge interessant ist. Darüber hinaus beobachtet man bei dem rekursiven Zerlegungsverfahren auch dann, wenn die ursprünglich zu realisierende Funktion total ist, auf höheren Rekursionsstufen *partielle* Funktionen.

  Speziell bei großen don't care–Mengen können sich hinsichtlich der Komplexität der resultierenden Booleschen Funktion je nach don't care–Belegung *enorme* Unterschiede ergeben.

Es besteht also ein dringender Bedarf nach Verfahren, die eine günstige Belegung von don't cares partieller Funktionen berechnen.

In dieser Arbeit gelingt dies hauptsächlich durch zwei Ansätze: Zum einen wurde ein Verfahren entwickelt, das bei einer vorgegebenen Variablenaufteilung don't cares so belegt, daß die Anzahl der Zerlegungsfunktionen im aktuellen Zerlegungsschritt minimiert wird. Zum anderen wurde ein Verfahren hergeleitet, das don't cares mit dem Ziel belegt, möglichst viele Symmetrieeigenschaften herzustellen. Dieses Verfahren hat im Gegensatz zur Minimierung der Anzahl von Zerlegungsfunktionen im aktuellen Zerlegungsschritt eine eher „globale" Auswirkung: Durch Beschränkung des Suchraumes auf *strikte* Zerlegungsfunktionen konnte erreicht werden, daß sich Symmetrien von der zerlegten Funktion auf die Zerlegungsfunktionen *übertragen*. Insofern wirkt diese Art von don't care–Belegung nicht nur auf den aktuellen Zerlegungsschritt.
Es konnte darüber hinaus gezeigt werden, daß die beiden angesprochenen Verfahren zur don't care–Belegung sogar insofern verträglich sind, als für das Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen unter Einhaltung bestimmter Nebenbedingungen garantiert werden kann, daß es keine Symmetrien zerstört, die durch das Verfahren zur Erzeugung von Symmetrien hergestellt wurden.

Schließlich kann man auch die Frage stellen, ob sich don't cares evtl. auch im Hinblick auf die Identifizierung mehrfachverwendbarer Teillogik belegen lassen.

Auch diese Frage konnte positiv beantwortet werden: Es wurde ein Verfahren gefunden, das don't cares im Hinblick auf die Berechnung gemeinsamer Zerlegungsfunktionen mehrerer Ausgangsfunktionen belegt.

Durch Anwendung der Verfahren zur don't care–Belegung konnten sogar bei *totalen* Benchmarkfunktionen (aufgrund des Auftretens von don't cares auf höheren Rekursionsstufen des Verfahrens) die Kosten um bis zu 35 % gesenkt werden. Bei der Synthese eines partiellen 4–Bit–Multiplizierers (der viele Symmetrien enthält), konnten die Kosten sogar um 43 % verringert werden (vergleiche Kapitel 4).

Ein entscheidender Durchbruch bezüglich der praktischen Anwendbarkeit der Algorithmen auf größere Beispiele ist dadurch gelungen, daß sämtliche Algorithmen auf Grundlage von ROBDDs [Bry86] als Datenstruktur zur Repräsentation Boolescher Funktionen formuliert werden konnten. ROBDDs (**r**educed **o**rdered **b**inary **d**ecision **d**iagrams) haben die Eigenschaft, daß sie für viele praktisch relevante Boolesche Funktionen eine relativ kompakte Darstellung liefern. [2]

[2]Die tabellenbasierte Version des implementierten Verfahrens benötigte z.B. zum Generieren eines 8–Bit–Addierers über 18 Stunden, der 16–Bit–Addierer konnte aufgrund der Datenmenge zur Repräsentation als Tabelle (8 Gigabyte) durch die tabellenbasierte Version nicht mehr erzeugt werden. In der ROBDD–basierten Version läßt sich dagegen ein 32–Bit–Addierer noch in 3 min, ein 64–Bit–Addierer in ca. $\frac{1}{2}$ Stunde erzeugen.

Aufbau der Arbeit und Resultate

Das erste Kapitel definiert grundlegende Begriffe, die zum Verständnis der Arbeit notwendig sind. Es führt verschiedene Repräsentationsmöglichkeiten Boolescher Funktionen ein: Ω–Schaltkreise, Boolesche Ausdrücke und ROBDDs zur Darstellung totaler und partieller Boolescher Funktionen. Außerdem werden bekannte Resultate zur Komplexität Boolescher Funktionen angegeben, die deutlich machen, was man prinzipiell von Algorithmen zur Logiksynthese erwarten kann und was nicht.

Das zweite Kapitel beschäftigt sich mit Symmetrien Boolescher Funktionen. Mit Hinblick auf die Logiksynthese wird zu einer Booleschen Funktion f mit den Eingangsvariablen $x_1, \ldots, x_n$ eine möglichst kleine Partition $P = \{\lambda_1, \ldots, \lambda_l\}$ bestimmt, so daß für alle Variablenpaare (x_j, x_k) aus einer solchen Menge λ_i die Funktion invariant ist gegenüber Vertauschung von x_j und x_k. Dieses Problem erweist sich bei partiellen Funktionen im Vergleich zu totalen Funktionen als wesentlich schwieriger. Die auftretenden Probleme werden anhand von Beispielen demonstriert. Es wird erstmals eine Heuristik angegeben, die (in Anlehnung an eine Heuristik zur Färbung von Graphen) zu einer partiellen Funktion f eine Belegung der don't cares liefert, so daß die resultierende Funktion symmetrisch in einer möglichst kleinen Partition der Eingangsvariablen ist. Das in dieser Arbeit eingeführte Konzept der *starken Symmetrie* partieller Boolescher Funktionen erfüllt hierbei die Aufgabe, den speziell im Zusammenhang mit Symmetrien *partieller* Funktionen auftretenden Problemen zu begegnen. Diese Suche nach Erweiterungen partieller Boolescher Funktionen, die möglichst viele Symmetrieeigenschaften besitzen, ist deshalb von großer Bedeutung, weil symmetrische Funktionen im allgemeinen sehr viel leichter zu realisieren sind als allgemeine Funktionen. Die Resultate werden von Vertauschungssymmetrien verallgemeinert auf sogenannte Äquivalenzsymmetrien (d.h. Symmetrien, bei denen vor Vertauschung zweier Eingangsvariablen eine der beiden Variablen noch negiert wird).

Das dritte Kapitel befaßt sich mit der Realisierung Boolescher Funktionen durch (rekursive) Zerlegung. Zwei Möglichkeiten zur Ausnutzung von Freiheiten sind dabei wesentlich: Bei totalen Funktionen die Ausnutzung von Freiheiten bei der konkreten Auswahl von Zerlegungsfunktionen und bei partiellen Funktionen zusätzlich die Belegung von don't cares, um zu einer möglichst guten Realisierung zu kommen. Die Freiheit bei der Wahl von Zerlegungsfunktionen wird wiederum auf zwei verschiedene Arten ausgenutzt:

- Durch die Beschränkung des Suchraumes auf die sogenannten *strikten* Zerlegungsfunktionen wird erreicht, daß sich Struktureigenschaften von der zerlegten Funktion auf die Zerlegungsfunktionen übertragen.
- Hat man Funktionen mit mehreren Ausgängen $f_1, \ldots, f_m$ zu zerlegen, so werden solche (strikte) Zerlegungsfunktionen berechnet, die in der Zerlegung möglichst vieler Ausgangsfunktionen f_i mit Vorteil verwendet werden können.

Das Kapitel beginnt mit einem Abschnitt zur Zerlegung von Funktionen mit einem Ausgang, in dem hauptsächlich Grundlagen zur Zerlegung Boolescher Funktionen behandelt werden.

Danach folgt ein Abschnitt über Zerlegungen partieller Boolescher Funktionen. Es wird ein Verfahren angegeben, das bei vorgegebener Variablenaufteilung don't cares mit dem Ziel belegt, die Anzahl der in der Zerlegung benötigten Zerlegungsfunktionen zu minimieren. Daneben besteht auch die Möglichkeit, don't cares so zu belegen, daß die resultierende Funktion möglichst viele Symmetrieeigenschaften aufweist. In einem späteren Abschnitt über Symmetrieausnutzung wird die Verträglichkeit der beiden Verfahren gezeigt.

Da don't cares Freiheiten darstellen, deren Ausnutzung zu Realisierungen mit geringeren Kosten führen können, ist es im Rahmen des rekursiven Zerlegungsverfahrens erstrebenswert, für Zerlegungs- und Zusammensetzungsfunktionen möglichst große don't care–Mengen zu definieren. Aus diesem Grund wird im Abschnitt über partielle Funktionen noch gezeigt, wie man bei einer vorgegebenen Zerlegung maximale don't care–Mengen für Zerlegungs– und Zusammensetzungsfunktionen berechnet.

Im Abschnitt über Funktionen mit mehreren Ausgängen wird ein Algorithmus zur Identifizierung mehrfach verwendbarer Teillogik entwickelt, der auf der Berechnung gemeinsamer strikter Zerlegungsfunktionen mehrerer Ausgangsfunktionen basiert. Ein wesentlicher Punkt ist hierbei, daß der Algorithmus komplett auf der Basis von ROBDDs arbeitet. Kann die zu zerlegende Funktion kompakt durch ROBDDs dargestellt werden, so wird die Zerlegung durchgeführt, ohne daß man einen Umweg über Funktionstabellen o.ä. machen muß. Darüber hinaus wird in diesem Abschnitt ein Verfahren angegeben, das bei partiellen Funktionen don't cares im Hinblick auf die Berechnung gemeinsamer Zerlegungsfunktionen belegt. Auch für dieses Verfahren kann die Verträglichkeit mit dem Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen gezeigt werden.

Im Gegensatz zu den bisher betrachteten disjunkten Zerlegungen sind bei nichtdisjunkten Zerlegungen die Mengen der freien und gebundenen Variablen nicht disjunkt. Es wird gezeigt, daß man das in dieser Arbeit beschriebene Logiksyntheseverfahren ohne größere Probleme auf nichtdisjunkte Zerlegungen erweitern kann.

Das Kapitel endet mit einem Abschnitt zum Thema Testen. In diesem Abschnitt wird ein Verfahren entwickelt, das parallel zur rekursiven Zerlegung einer Funktion $f : \{0,1\}^n \rightarrow \{0,1\}^m$ einen vollständigen Test hinsichtlich des Stuck-at–Fehlermodells (oder hinsichtlich des Zellenfehlermodells) berechnet. Dabei spielt die Berechnung sogenannter Freiheitsrelationen Boolescher Funktionen eine entscheidende Rolle. Zu jedem (Einzel-)Stuck-at-Fehler im resultierenden Schaltkreis erhält man so eine Testeingabe oder die Information, daß der Fehler redundant (d.h. nicht testbar) ist. Erkannte redundante Fehler können während des rekursiven Zerlegungsverfahrens direkt entfernt werden. Es kann jedoch nicht garantiert werden, daß bei dem Verfahren keine redundanten Fehler auftreten. Grund dafür ist die Tatsache, daß beim Entfernen redundanter Fehler testbare Fehler zu redundanten Fehlern werden können. Dies wird anhand eines Beispiels demonstriert.

Kapitel 4 zeigt Ergebnisse des auf diesen Grundlagen implementierten Logiksyntheseverfahrens. Neben Benchmarkvergleichen wird die Wirkungsweise des Algorithmus auch anhand einiger Beispielentwürfe gezeigt.

Das Logiksynthesewerkzeug ist in das Entwurfssystem CADIC [BHK+87] integriert, das auf einem in [Hot65] entwickelten Kalkül basiert. Ein wichtiges Merkmal von CADIC ist die Beschreibung von Schaltungen durch rekursive, parametrisierte Gleichungen. Ein System von Gleichungen definiert nicht nur einen einzelnen Schaltkreis, sondern eine ganze Schaltkreisfamilie. Damit eignet sich CADIC in besonderer Weise zur kompakten, hierarchischen Beschreibung regulärer Schaltkreise. Anhand des Entwurfs eines fehlererkennenden Dividierers [TY88] wird demonstriert, daß die automatische Logiksynthese nicht unbedingt in Konkurrenz zu Handentwürfen regelmäßiger Schaltungen zu sehen ist, sondern auch in Verbindung mit solchen Entwürfen nutzbringend eingesetzt werden kann. Es erfolgte ein parametrisierter Entwurf für einen n–Bit–Dividierer (in einer redundanten Zahlendarstellung) mit dem Entwurfssystem CADIC. Der Dividierer ist regelmäßig unter Verwendung bestimmter Grundmodule aufgebaut. Diese Grundmodule, aus deren Spezifikation keine Regelmäßigkeiten abzulesen waren, wurden mit Hilfe des Logiksynthesewerkzeuges automatisch generiert.

Speziell anhand der Entwürfe für einen Addierer und einen partiellen Multiplizierer kann demonstriert werden, daß auch bei Problemstellungen, die schon intensiv unter Einsatz *menschlicher* Intelligenz bearbeitet wurden, mit *automatischer* Logiksynthese konkurrenzfähige Entwürfe erzielt werden können. Aus der Spezifikation für den Addierer wurde ein Schaltkreis generiert, der starke Ähnlichkeit mit dem Conditional–Sum–Addierer hat [Sla60]. Kleinere Unterschiede im Detail bewirken sogar, daß die Gatteranzahl der generierten Lösung bei gleicher Tiefe im Vergleich zum Conditional–Sum–Addierer sogar noch geringer wird. Es konnten Addierer bis zu einer Bitbreite von 64 (d.h. mit 128 Eingängen und 64 Ausgängen) generiert werden.[3] Auch für den partiellen Multiplizierer wurde ein interessanter Schaltkreis gefunden. Analysiert man den resultierenden Schaltkreis und verallgemeinert man das Prinzip auf variable Bitbreiten, so erhält man einen Multiplizierer, der (wie z.B. der Wallace–Tree–Multiplizierer [Wal64]) logarithmische Tiefe hat und $O(n^2)$ Gatter benötigt. Anhand dieser Beispiele erkennt man also, daß es sogar denkbar ist, Realisierungen, die von dem automatischen Logiksynthesewerkzeug erzeugt wurden, als Anregung für parametrisierte Entwürfe (d.h. Entwürfe mit variabler Bitbreite) zu nutzen.

Danksagungen

Mein besonderer Dank gilt Herrn Prof. Dr. Günter Hotz und Herrn Prof. Dr. Paul Molitor für die begleitende Förderung dieser Arbeit, für zahlreiche Anregungen und Ratschläge.

Mein herzlicher Dank gilt auch Herrn Privatdozent Dr. Uwe Sparmann für das sorgfältige Korrekturlesen der Arbeit und seine Bereitschaft zu zahlreichen hilfreichen Diskussionen. Seine Änderungsvorschläge haben an vielen Stellen zur Verbesserung der Lesbarkeit der endgültigen Version beigetragen.

Nicht zuletzt danke ich meinen Kollegen am Lehrstuhl von Prof. Dr. Hotz für das gute Arbeitsklima während der Zeit der Entstehung dieser Arbeit.

[3] Bei dem 32–Bit–Addierer in 3 min, bei dem 64–Bit–Addierer in 31 min.

Diese Aufzählung ist natürlich nicht vollständig und so sei hier auch ohne namentliche Nennung all jenen Dank gesagt, die mir während der letzten Jahre beruflich wie privat mit Rat und Tat zur Seite standen.

Kapitel 1

Grundlagen

Das erste Kapitel definiert grundlegende Begriffe, die zum Verständnis der Arbeit notwendig sind. In einem ersten Abschnitt werden verschiedene Repräsentationsmöglichkeiten Boolescher Funktionen eingeführt: Ω–Schaltkreise, Boolesche Ausdrücke und ROBDDs zur Darstellung totaler und partieller Boolescher Funktionen. In einem zweiten Abschnitt werden dann bekannte Resultate zur Komplexität Boolescher Funktionen angegeben, die deutlich machen, was Algorithmen zur Logiksynthese prinzipiell leisten können bzw. was man nicht von ihnen erwarten kann.

1.1 Grundlegende Definitionen

1.1.1 Boolesche Funktionen

Aufgabe der Logiksynthese ist es, zu einer vorgegebenen Booleschen Funktion einen Schaltkreis zu finden, der diese realisiert. In diesem Teilabschnitt finden sich Definitionen und Bezeichnungen, die im Zusammenhang mit Booleschen Funktionen eine Rolle spielen.

Definition 1.1 (Totale Boolesche Funktionen)
$B_{n,m} = \{f : \{0,1\}^n \to \{0,1\}^m\}$ *sei die Menge aller* **(totalen) Booleschen Funktionen** *bzw. Schaltfunktionen von* $\{0,1\}^n$ *nach* $\{0,1\}^m$.

Bezeichnung 1.1 $B_n := B_{n,1}$

Definition 1.2 (Partielle Boolesche Funktionen)
$BP_{n,m} = \{f : D \to \{0,1\}^m \mid D \subseteq \{0,1\}^n\}$ *sei die Menge aller* **partiellen Booleschen Funktionen** *bzw. Schaltfunktionen von* $\{0,1\}^n$ *nach* $\{0,1\}^m$. *D heißt Definitionsbereich von* $f : D \to \{0,1\}^m$ *und wird mit* $D(f)$ *bezeichnet.*

Bezeichnung 1.2 *Für $D \subseteq \{0,1\}^n$ sei $S(D) = \{f : D \to \{0,1\}\}$ die Menge aller* **partiellen Booleschen Funktionen** *von D nach $\{0,1\}$.*

Definition 1.3 (ON–Menge, OFF–Menge, DC–Menge)
Sei $f : D \to \{0,1\}$, $D \subseteq \{0,1\}^n$.
Dann heißt

- $DC(f) = \{0,1\}^n \setminus D$ *die* **don't care–Menge** *(kurz DC–Menge) von f,*
- $ON(f) = \{x \in D \mid f(x) = 1\}$ *die* **ON–Menge** *von f,*
- $OFF(f) = \{x \in D \mid f(x) = 0\}$ *die* **OFF–Menge** *von f.*

Um aus vorgegebenen Booleschen Funktionen leicht neue Funktionen definieren zu können, benutzt man die Funktionen aus B_1 und B_2 als Operationen auf den Funktionen aus $S(D)$. Sind $g, h \in S(D)$ und ist f eine Operation aus B_2 dann schreibt man im allgemeinen statt $f(g,h)$ in Infixschreibweise $g\ f\ h$. Es gilt also:

$$\forall x \in D\ :\ (g\ f\ h)(x) = f(g(x), h(x))$$

Ebenso für $f \in B_1$:

$$\forall x \in D\ :\ (f\ g)(x) = f(g(x))$$

Allgemein sind für Funktionen aus B_1 bzw. B_2 folgende Bezeichnungen üblich:

Bezeichnung 1.3

(a) *$not(x) = 1$ genau dann, wenn $x = 0$. not heißt Negation von x. Schreibweise: $\overline{x}$ oder $\neg x$.*

(b) *$and(x,y) = 1$ genau dann, wenn $x = y = 1$. and heißt Konjunktion von x und y. Schreibweise: $x \wedge y$, $x \cdot y$ oder xy.*

(c) *$nand(x,y) = 0$ genau dann, wenn $x = y = 1$.*

(d) *$or(x,y) = 0$ genau dann, wenn $x = y = 0$. or heißt Disjunktion von x und y. Schreibweise: $x \vee y$ oder $x + y$.*

(e) *$nor(x,y) = 1$ genau dann, wenn $x = y = 0$.*

(f) *$exor(x,y) = 1$ genau dann, wenn $x \neq y$. $exor$ heißt exclusive-or-Funktion von x und y. Schreibweise: $x \oplus y$.*

(g) *$equiv(x,y) = 1$ genau dann, wenn $x = y$. Schreibweise: $x \equiv y$.*

(h) *$\mathbf{0}(x,y) = 0\ \forall x, y \in \{0,1\}$. $\mathbf{0}$ ist die konstante Nullfunktion.*

(i) *$\mathbf{1}(x,y) = 1\ \forall x, y \in \{0,1\}$. $\mathbf{1}$ ist die konstante Einsfunktion.*

Bemerkung 1.1 *Sei $\pi_i^n \in B_n$ die i–te Projektion, d.h. $\pi_i^n(x_1, \ldots, x_n) = x_i$. Falls aus dem Zusammenhang hervorgeht, was gemeint ist, wird häufig π_i^n einfach als x_i bezeichnet.*

1.1.2 Repräsentationen Boolescher Funktionen

In diesem Abschnitt werden verschiedene Möglichkeiten zur Repräsentation Boolescher Funktionen angegeben. Die allgemeinste Darstellungsform ist dabei ein Schaltkreis zur Realisierung der Booleschen Funktion:

1.1.2.1 Ω–Schaltkreise

Im allgemeinen hat man zur Realisierung Boolescher Funktionen eine gewisse Auswahl an (einfachen) Booleschen Funktionen zur Verfügung, mit deren Hilfe komplexere Funktionen ausgedrückt werden können (vgl. Standardzellenentwurf). Die schon verfügbaren Funktionen sind in einer Zellenbibliothek zusammengefaßt:

Definition 1.4 (Zellenbibliothek) *Eine endliche Teilmenge $\Omega \subseteq \bigcup_{n \in \mathbf{N}} B_n$ heißt Zellenbibliothek.*

Im folgenden wird häufig als Zellenbibliothek $\Omega = B_2$, $\Omega = STD = \{0, 1, and, nand, or, nor, not\}$ oder $\Omega = B_2 \setminus \{exor, equiv\} =: R_2$ betrachtet.

Im allgemeinen bezeichnet man Realisierungen für Funktionen aus einer Zellenbibliothek als Gatter.

Die beiden nächsten Definitionen geben an, wie man mit Hilfe von Funktionen aus einer solchen Zellenbibliothek Ω komplexere Funktionen darstellt. Die Darstellung erfolgt durch einen sogenannten Ω–Schaltkreis, der direkt als Netzliste einer Schaltung interpretiert werden kann.

Definition 1.5 (Schaltkreis)
Sei Ω eine Zellenbibliothek. Sei $T = \Omega \cup \{EPAD, APAD\}$.
*Ein Ω–***Schaltkreis** *S mit n Eingängen und m Ausgängen ist ein 4–Tupel*

$$(G = (V, E), typ, pe, pa),$$

wobei gilt:

- *G ist ein gerichteter, azyklischer, knotenorientierter[1] Graph. V ist die Menge der Knoten bzw. Zellen, E die Menge der Kanten bzw. Verbindungsleitungen.*
  Jeder Kante ist eine Richtung zugeordnet. Quelle und Ziel von Kanten sind gegeben durch die Abbildungen $Q : E \to V$ und $Z : E \to V$.
  Der Eingangsgrad eines Knotens v ist definiert durch die Abbildung $indeg : V \to \mathbf{N}$, $indeg(v) = |\{e \in E \mid Z(e) = v\}|$.
  Entsprechend ist der Ausgangsgrad eines Knotens v definiert durch die Abbildung

[1]Ein Graph G heißt *knotenorientiert*, wenn es für jeden Knoten v des Graphen sowohl eine Numerierung der einlaufenden Kanten als auch eine Numerierung der auslaufenden Kanten gibt.

$outdeg : V \to \mathbf{N}$, $outdeg(v) = |\{e \in E \mid Q(e) = v\}|$.
Die Knotenorientierungen von G sind durch die partiellen injektiven Abbildungen $I, O : V \times \mathbf{N} \rightsquigarrow E$ *definiert. Falls* $1 \leq i \leq indeg(v)$, *dann ist* $I(v,i)$ *definiert,* $I(v,i) = e$ *mit* $Z(e) = v$ *und e heißt i–ter Input des Knotens v; falls* $1 \leq i \leq outdeg(v)$, *dann ist* $O(v,i)$ *definiert,* $O(v,i) = e$ *mit* $Q(e) = v$ *und e heißt i–ter Output von v.*

- $typ : V \to T$ *ist eine Abbildung, die jedem Knoten einen „Typ" (Funktion der Zellenbibliothek oder EPAD bzw. APAD) zuordnet.*
  Es muß gelten: $|typ^{-1}(EPAD)| = n$, $|typ^{-1}(APAD)| = m$.
  Falls $typ(v) = t$, *dann heißt t Typ von v, v ein t–Knoten.*
  Falls $typ(v) \in \Omega$, *dann gilt* $typ(v) \in B_{indeg(v)}$.
  Falls $typ(v) = EPAD$, *dann gilt* $indeg(v) = 0$,
  falls $typ(v) = APAD$, *dann gilt* $indeg(v) = 1$, $outdeg(v) = 0$.

- *pe, pa sind „Numerierungen" der EPAD– bzw. APAD–Knoten.*
  $pe : \{1, \ldots, n\} \to \{v \in V \mid typ(v) = EPAD\}$, $pa : \{1, \ldots, m\} \to \{v \in V \mid typ(v) = APAD\}$ *sind bijektive Abbildungen.*
  $pe(i)$ $(pa(i))$ *heißt der i. primäre Eingang (Ausgang).*

Die nun folgende Definition stellt den Zusammenhang her zwischen einem Ω–Schaltkreis und den Booleschen Funktionen, die durch diesen Schaltkreis realisiert werden.

Definition 1.6 (Durch Ω–Schaltkreis definierte Funktion)
Sei $S = (G, typ, pe, pa)$ *ein* Ω*–Schaltkreis mit n Eingängen und m Ausgängen.*
Sei $e \in E$ *eine Verbindungsleitung mit* $e = O(v, j)$.

- *Falls* $typ(v) = EPAD$ *und* $v = pe(i)$, *dann ist die durch Leitung e berechnete Funktion definiert durch*

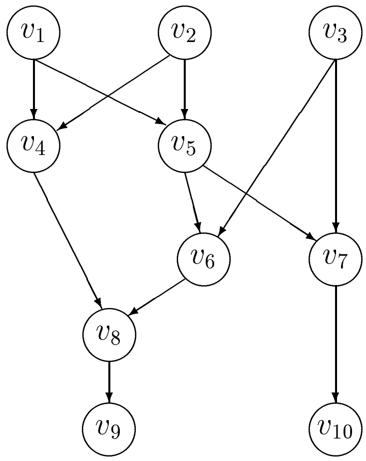
$$f_e : \{0,1\}^n \to \{0,1\}, \quad f_e(x_1, \ldots, x_n) = x_i.$$

- *Falls* $typ(v) = g \in \Omega$, $e_k = I(v,k)$ *für* $k = 1, \ldots, indeg(v)$, *dann ist die durch die Leitung e berechnete Funktion definiert durch*

$$f_e : \{0,1\}^n \to \{0,1\}, \quad f_e(\mathbf{x}) = g(f_{e_1}(\mathbf{x}), \ldots, f_{e_{indeg(v)}}(\mathbf{x}))$$

Seien nun für $1 \leq i \leq m$ $y_i = I(pa(i), 1)$, f_{y_i} *jeweils die durch* y_i *berechneten Funktionen. Dann ist die durch S realisierte Funktion* $f_S : \{0,1\}^n \to \{0,1\}^m$ *definiert durch*

$$f_S(\mathbf{x}) = (f_{y_1}(\mathbf{x}), \ldots, f_{y_m}(\mathbf{x})).$$



v	$typ(v)$
v_1	$EPAD$
v_2	$EPAD$
v_3	$EPAD$
v_4	and
v_5	$exor$
v_6	and
v_7	$exor$
v_8	or
v_9	$APAD$
v_{10}	$APAD$

i	$pe(i)$
1	v_1
2	v_2
3	v_3

i	$pa(i)$
1	v_9
2	v_{10}

Abbildung 1.1: Schaltkreis zu einem Volladdierer

Ein Schaltkreis S soll aber nicht nur als Realisierung einer einzigen totalen Funktion f_S gelten, sondern als Realisierung *aller* Funktionen, die man aus f_S durch Einschränkung des Definitionsbereichs erhält:

Definition 1.7 (*S* Realisierung von *g*) *Sei $f_S : \{0,1\}^n \to \{0,1\}^m$ die durch den Ω–Schaltkreis S realisierte Funktion.*
Dann heißt S Schaltkreis bzw. Realisierung für jede *Funktion $g : D \to \{0,1\}^m$, $D \subseteq \{0,1\}^n$, mit $f_S|_D = g$.*

Abbildung 1.1 illustriert die angegebenen Definitionen für einen Schaltkreis zu einem Volladdierer, d.h. zu der Funktion

$$fa : \{0,1\}^3 \to \{0,1\}^2,\ (x_1,x_2,x_3) \mapsto (c,s) \ \text{ mit } \ x_1+x_2+x_3 = s+2c.$$

Als Zellenbibliothek wurde $\Omega = \{\oplus,\ \cdot,\ \vee\}$ gewählt.

Üblicherweise wird man allerdings den Schaltkreis in einer etwas übersichtlicheren Form wie in Abbildung 1.2 graphisch darstellen. Es werden jetzt noch Kostenmaße für Ω–Schaltkreise eingeführt, die es ermöglichen sollen, die Kosten verschiedener Realisierungen der gleichen Funktion zu vergleichen. Dazu wird der Begriff der Zellenbibliothek noch um Zellenflächen und Zellenlaufzeiten erweitert.

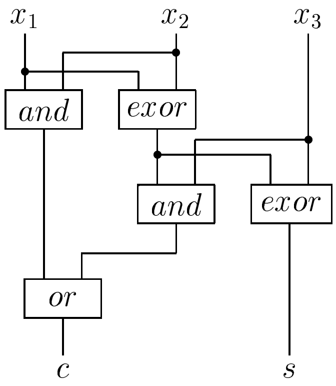


Abbildung 1.2: Schaltkreis zu einem Volladdierer (vereinfachte Darstellung)

Definition 1.8 (erweiterte Zellenbibliothek) *Eine* **erweiterte Zellenbibliothek** *ist ein Tripel* $(\Omega, A_\Omega, T_\Omega)$*, das aus einer Zellenbibliothek* Ω*, und zwei Funktionen* $A_\Omega : \Omega \to \mathbf{R}_0^+$ *und* $T_\Omega : \Omega \to \mathbf{R}_0^+$ *besteht. Hierbei ordnet* A_Ω *jeder Funktion aus* Ω *eine Gatterfläche zu,* T_Ω *ordnet jeder Funktion aus* Ω *eine Gatterlaufzeit zu.*

Definition 1.9 (Kostenmaße)

1. *Die* Ω–**Komplexität** *eines* Ω*–Schaltkreises* $S = ((V,E), typ, pe, pa)$ *ist*

$$C_\Omega(S) = |V \setminus \{v \in V \mid typ(v) \in \{EPAD, APAD\}\}|$$

(D. h. die Ω*–Komplexität wird bestimmt durch die Anzahl der Zellen des Schaltkreises, die keine* $EPAD$*– oder* $APAD$*–Zellen sind.)*
Die Ω*–Komplexität einer Booleschen Funktion* $f \in BP_{n,m}$ *mit Definitionsmenge* D *ist*

$$C_\Omega(f) = \min\{C_\Omega(S) \mid f_S|_D = f\}$$

Eine Realisierung S *von* f *mit* $C_\Omega(S) = C_\Omega(f)$ *heißt* Ω*–optimal.*

2. *Die* **Zellenfläche** *eines* Ω*–Schaltkreises* $S = ((V,E), typ, pe, pa)$ *ist definiert als*

$$A_\Omega(S) = \sum_{\substack{v \in V \\ typ(v) \notin \{EPAD, APAD\}}} A_\Omega(typ(v))$$

(D. h. die Zellenfläche ergibt sich als Summe der Gatterflächen sämtlicher Zellen des Schaltkreises.)
Die Zellenfläche einer Booleschen Funktion $f \in BP_{n,m}$ *mit Definitionsmenge* D *ist entsprechend*

$$A_\Omega(f) = \min\{A_\Omega(S) \mid f_S|_D = f\}$$

3. *Die* **Tiefe** *eines Ω–Schaltkreises $S = (G = (V, E), typ, pe, pa)$ ist $D_\Omega(S)$, die größte Anzahl von Zellen auf einem gerichteten Pfad in G (EPAD– und APAD–Zellen nicht mitgerechnet).*
   Die Tiefe einer Booleschen Funktion $f \in BP_{n,m}$ mit Definitionsmenge D ist entsprechend
   $$D_\Omega(f) = \min\{D_\Omega(S) \mid f_S|_D = f\}$$

4. *Die* **Zellenlaufzeit** *eines Ω–Schaltkreises $S = (G = (V, E), typ, pe, pa)$ $T_\Omega(S)$ ist definiert als die größte Summe der Gatterlaufzeiten auf einem gerichteten Pfad von einem EPAD–Knoten zu einem APAD–Knoten.*
   Die Zellenlaufzeit einer Booleschen Funktion $f \in BP_{n,m}$ mit Definitionsmenge D ist
   $$T_\Omega(f) = \min\{T_\Omega(S) \mid f_S|_D = f\}$$

Bemerkung 1.2 *Um die Laufzeit eines Schaltkreises S genauer zu modellieren, kann man auch den Verbindungsleitungen Laufzeiten zuordnen (z.B. auch in Abhängigkeit vom Ausgangsgrad der einzelnen Knoten) und für alle gerichteten Pfade von einem EPAD–Knoten zu einem APAD–Knoten zu den Gatterlaufzeiten der Zellen auf dem Pfad die Laufzeiten der Verbindungsleitungen auf dem Pfad addieren.*

Wenn die zugrundeliegende Zellenbibliothek aus dem Zusammenhang hervorgeht, wird teilweise auch $C(S)$ bzw. $C(f)$ statt $C_\Omega(S)$ bzw. $C_\Omega(f)$ geschrieben.

1.1.2.2 Boolesche Ausdrücke

Eine weitere Beschreibungsmöglichkeit Boolescher Funktionen aus B_n stellen geklammerte Boolesche Ausdrücke dar.

Sie haben eine direkte Entsprechung zu Ω–Schaltkreisen über der Zellenbibliothek $\{0, 1, and, or, not\}$ und werden besonders zur Beschreibung zweistufiger Realisierungen verwendet.

Definition 1.10 (Boolesche Ausdrücke über Var_n) *Sei $Var_n = \{x_1, \ldots, x_n\}$ eine n–elementige Menge, die Menge der* **Variablen**, *$E = \{0, 1, (,), \cdot, \vee, \neg\} \cup Var_n$ und E^+ die freie Wörterhalbgruppe über E. Dann heißt $\mathcal{A}(Var_n)$, der Durchschnitt aller Teilmengen $L \subseteq E^+$ mit*

- $\{0, 1\} \cup Var_n \in L$,
- $w_1, \ldots, w_k \in L \Longrightarrow (w_1 \cdot \ldots \cdot w_k) \in L$ *und* $(w_1 \vee \ldots \vee w_k) \in L$,
- $w \in L \Longrightarrow (\neg w) \in L$,

die Menge der **Booleschen Ausdrücke über** Var_n.

Schreibweise: Für $(\neg w)$ schreiben wir auch $\overline{w}$.

Definition 1.11 (Durch Booleschen Ausdruck definierte Funktion)
Die durch einen Booleschen Ausdruck w über Var_n definierte Funktion $\Phi(w)$ ist definiert durch

- $\Phi(w) = \mathbf{0}$, *falls* $w = 0$,
- $\Phi(w) = \mathbf{1}$, *falls* $w = 1$,
- $\Phi(w) = \pi_i^n$, *falls* $w = x_i$,
- $\Phi(w) = \Phi(w_1) \cdot \ldots \cdot \Phi(w_k)$, *falls* $w = (w_1 \cdot \ldots \cdot w_k)$,
- $\Phi(w) = \Phi(w_1) \vee \ldots \vee \Phi(w_k)$, *falls* $w = (w_1 \vee \ldots \vee w_k)$,
- $\Phi(w) = not(\Phi(v))$, *falls* $w = (\neg v)$.

Zur Beschreibung zweistufiger Realisierungen durch Boolesche Ausdrücke werden noch folgende Definitionen benötigt:

Definition 1.12 (Boolesches Monom)
$m \in \mathcal{A}(\{x_1, \ldots x_n\})$ heißt **Boolesches Monom**, *falls*

$$m = x_{i_1}^{\epsilon_1} \cdot \ldots \cdot x_{i_j}^{\epsilon_j} \text{ mit } 1 \leq j \leq n, \text{ und } \epsilon_k \in \{0,1\} \text{ für } k = 1, \ldots, j,$$

$$\text{wobei } x_{i_k}^1 = x_{i_k} \text{ und } x_{i_k}^0 = \overline{x_{i_k}} \text{ gilt.}$$

Unter der **Variablenmenge** *von m versteht man die Menge $V(m) = \{x_{i_1}, \ldots, x_{i_j}\}$. Das* **Monom** *heißt* **vollständig**, *wenn $V(m) = \{x_1, \ldots, x_n\}$ gilt.*

Definition 1.13 (Boolesches Polynom)
$p \in \mathcal{A}(\{x_1, \ldots x_n\})$ heißt **Boolesches Polynom**, *falls gilt:*

- $p = \mathbf{0}$ *oder*
- $p = \mathbf{1}$ *oder*
- $p = (m_1 \vee \ldots \vee m_k)$, *wobei* $\forall i \in \{1, \ldots, k\}$ m_i *ein Monom ist.*

Die Definition der Kosten eines Booleschen Polynoms p ist motiviert durch die R_2–Komplexität des R_2-Schaltkreises, der p in natürlicher Weise zugeordnet werden kann:

Definition 1.14 (Kosten eines Booleschen Polynoms)
Die Kosten $C(p)$ eines Booleschen Polynoms p betragen

- $C(p) = 0$, *falls* $p = 0$ *oder* $p = 1$,
- $C(p) = k-1+\sum_{i=1}^{k} C(m_i)$, *falls* $p = (m_1 \vee \ldots \vee m_k)$, *wobei die Kosten eines Monoms* $m = x_{i_1}^{\epsilon_1} \cdot \ldots \cdot x_{i_j}^{\epsilon_j}$ $C(m) = j-1$ *betragen.*

Bei zweistufiger Logiksynthese wird zu einer gegebenen Funktion f aus $S(D)$, $D \subseteq \{0,1\}^n$ ein Polynom $p \in \mathcal{A}(Var_n)$ mit möglichst geringen Kosten (geringer Komplexität) gesucht, so daß $\Phi(p)(x) = f(x)\ \forall x \in D$ gilt.

Definition 1.15 (Minimalpolynom)
Sei f eine Boolesche Funktion aus $S(D)$, $D \subseteq \{0,1\}^n$. Ein Polynom $p \in \mathcal{A}(Var_n)$ heißt **Minimalpolynom** *von f, falls*

- $\forall x \in D \quad \Phi(p)(x) = f(x)$ *und*
- $C(p) \leq C(p')$ *für alle Polynome* p' *mit* $\Phi(p')(x) = f(x) \quad \forall x \in D$

Zweistufige Lösungen (Boolesche Polynome) werden recht häufig zur Realisierung Boolescher Funktionen gewählt. Zum einen kann man Boolesche Polynome leicht durch programmierbare logische Felder (PLA's) realisieren, zum anderen gibt es mittlerweile relativ ausgereifte heuristische Verfahren (z.B. ESPRESSO [BHM+84]), die in der Lage sind, zu einer Booleschen Funktion Minimalpolynome bzw. Polynome, deren Kosten die eines Minimalpolynoms nicht wesentlich übersteigen, zu berechnen.

Trotzdem ist es nicht ratsam, sich auf zweistufige Realisierungen zu beschränken, da in vielen Fällen Minimalpolynomrealisierungen von Schaltfunktionen *wesentlich* teurer sind als andere (mehrstufige) Realisierungen. Extrembeispiele sind z.B. gerade und ungerade Paritätsfunktion (*equiv* und *exor* mit n Eingängen) oder der Binäraddierer, die mit linearer C_{R_2}-Komplexität (linear in der Anzahl der Eingänge) als mehrstufige Schaltkreise realisierbar sind, deren Minimalpolynome aber exponentielle Größe haben(für den Addierer siehe [Weg89], Kapitel 2.11).

1.1.2.3 Geordnete binäre Entscheidungsgraphen bzw. OBDDs

Eine weitere Möglichkeit zur Repräsentation Boolescher Funktionen stellen geordnete binäre Entscheidungsgraphen bzw. OBDDs (ordered binary decision diagrams) und reduzierte geordnete binäre Entscheidungsgraphen bzw. ROBDDs (reduced ordered binary decision diagrams) dar [Bry86].

OBDDs bzw. speziell ROBDDs haben die Eigenschaft, daß sie für viele praktisch relevante Boolesche Funktionen eine relativ kompakte Darstellung liefern. (Beispielsweise können Paritätsfunktion und Binäraddierer, die im vorigen Abschnitt schon als Extrembeispiele mit exponentiell großen Minimalpolynomen erwähnt wurden, als ROBDDs mit linearer Größe dargestellt werden [Bry86].)

ROBDDs sind eine kanonische Darstellung Boolescher Funktionen, so daß Probleme wie z.B. der Test auf Erfüllbarkeit oder Tautologie oder allgemein der Test auf Gleichheit Boolescher Funktionen einfach werden, falls die entsprechenden Booleschen Funktionen als ROBDDs dargestellt sind.

Darüber hinaus haben ROBDDs zwei weitere Vorteile: Zum einen kann die Größe der ROBDDs durch Anwendung von Negation und binären Booleschen Verknüpfungen (*and*, *or*, *exor* usw.) nicht „explodieren". Während Boolesche Polynome allein durch Negation exponentiell anwachsen können, läßt die Negation die Größe eines ROBDDs unverändert und bei binären Booleschen Verknüpfungen kann die Größe des Resultats höchstens dem Produkt der Größen der eingehenden ROBDDs entsprechen. Zum anderen lassen sich diese Operationen sehr effizient durchführen; sie sind polynomiell in der Größe der eingehenden ROBDDs.

Binäre Entscheidungsgraphen wurden schon von Lee [Lee59] und Akers [Ake78] benutzt. Sie beruhen auf der Darstellung einer Booleschen Funktion durch ihre Shannon–Expansion [Sha38]

$$f(x_1,\ldots,x_n) = \overline{x_i}\cdot f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n) + x_i\cdot f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n).$$

Allerdings machte es erst die von Bryant [Bry86] eingeführte Beschränkung auf eine bestimmte Teilklasse, die *geordneten* binären Entscheidungsgraphen, möglich, effiziente Operationen auf diesen Repräsentationen Boolescher Funktionen durchzuführen.

Geordnete binäre Entscheidungsgraphen (OBDDs) sind wie folgt definiert:

Definition 1.16 (Geordneter binärer Entscheidungsgraph (OBDD))
Sei $X = \{x_1,\ldots,x_n\}$ eine Menge von Variablen, $index : \{1,\ldots,n\} \to \{1,\ldots,n\}$ eine Permutation auf $\{1,\ldots,n\}$. Durch index wird eine lineare Ordnung $<_{index}$ auf X induziert: Es gilt $x_{index(i)} <_{index} x_{index(j)}$ genau dann, wenn $i < j$.
$<_{index}$ wird als „Variablenordnung" bezeichnet.
Ein **geordneter binärer Entscheidungsgraph** (OBDD) *F über der Menge X mit Variablenordnung $<_{index}$ ist ein Paar (G,m) aus einem kantenorientierten und knotenorientierten Graphen $G = (V,E)$ und einer Markierungsfunktion $m : V \to (X \cup \{0,1\})$.*
G hat genau eine Quelle $q \in V$. Jeder Knoten $v \in V$, der keine Senke ist, hat genau 2 Söhne: den 0–Sohn v^0 und den 1–Sohn v^1.
Die Markierungsfunktion m ordnet jedem Knoten $v \in V$, der keine Senke ist, eine Markierung $m(v) \in X$ zu. Weiterhin gilt für alle Senken s $m(s) = 0$ oder $m(s) = 1$.
Für jeden Knoten $v \in V$, für den ein Sohn v^ϵ ($\epsilon \in \{0,1\}$) keine Senke ist, gilt: $m(v) <_{index} m(v^\epsilon)$.

Der *geordnete* binäre Entscheidungsgraph ist so definiert, daß auf jedem Pfad von der Quelle zu einer der Senken jede Variable aus X höchstens einmal als Markierung auftreten kann und zwar in der durch die Variablenordnung auf X vorgegebenen Reihenfolge $x_{index(1)}$ bis $x_{index(n)}$.

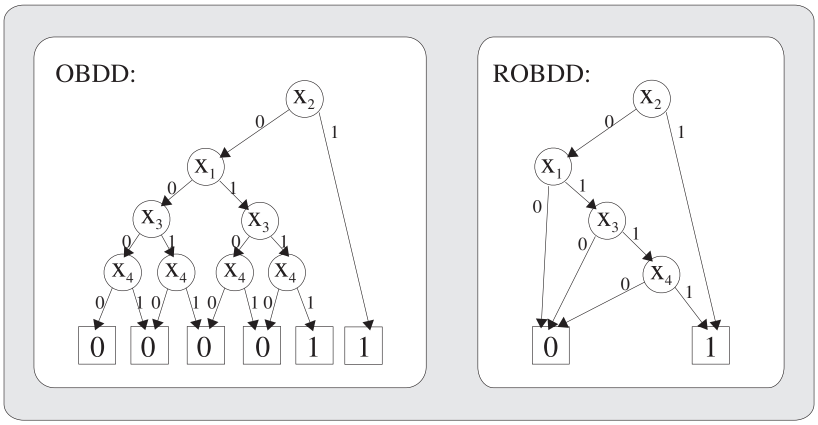


Abbildung 1.3: 2 OBDDs für $f(x_1, \dots, x_4) = x_2 + x_1x_3x_4$ mit der Variablenordnung $x_2 <_{index} x_1 <_{index} x_3 <_{index} x_4$.

Ein OBDD über $X = \{x_1, \dots, x_n\}$ (mit Variablenordnung $<_{index}$) definiert eine Boolesche Funktion $f \in B_n$:

Definition 1.17 (Durch OBDD definierte Funktion)
Sei $F = (G, m)$ ein geordneter binärer Entscheidungsgraph über der Menge $X = \{x_1, \dots, x_n\}$ von Variablen (mit Variablenordnung $<_{index}$). Sei $v \in V$ ein Knoten von G.

- *Sei $m(v) = 0$. Dann ist die durch v definierte Funktion f_v die Konstante* **0**.
  $f_v(x_1, \dots, x_n) = 0 \ \forall (x_1, \dots, x_n) \in \{0,1\}^n$.

- *Sei $m(v) = 1$. Dann ist die durch v definierte Funktion f_v die Konstante* **1**.
  $f_v(x_1, \dots, x_n) = 1 \ \forall (x_1, \dots, x_n) \in \{0,1\}^n$.

- *Sei $m(v) = x_i$. Sei v^0 der 0–Sohn von v, v^1 der 1–Sohn. Sei $f_{v^0} \in B_n$ die durch v^0 definierte Funktion, $f_{v^1} \in B_n$ die durch v^1 definierte Funktion.*

  Dann ist die durch v definierte Funktion $f_v \in B_n$ mit

  $$f_v(x_1, \dots, x_n) = \overline{x_i} \cdot f_{v^0}(x_1, \dots, x_n) + x_i \cdot f_{v^1}(x_1, \dots, x_n) \ \forall (x_1, \dots, x_n) \in \{0,1\}^n.$$

Sei $q \in V$ die einzige Quelle von G. Die durch $F = (G, m)$ definierte Funktion $f_F \in B_n$ ist dann $f_F = f_q$.

Abbildung 1.3 zeigt zwei OBDDs für die Funktion $f(x_1, \dots, x_4) = x_2 + x_1x_3x_4$ mit der Variablenordnung $x_2 <_{index} x_1 <_{index} x_3 <_{index} x_4$.

Bezeichnung 1.4 *Sei $F = (G = (V, E), m)$ ein geordneter binärer Entscheidungsgraph über der Variablenmenge $X = \{x_1, \ldots, x_n\}$ mit Variablenordnung $<_{index}$. Sei $\epsilon_{index} = (\epsilon_{index(1)}, \ldots, \epsilon_{index(k)}) \in \{0,1\}^k$, $k \leq n$. Dann ist der durch ϵ_{index} erreichbare Knoten von G durch folgenden Algorithmus definiert:*

1. *Beginne bei der Quelle q von G. $v := q$. $i := 1$.*
2. *Solange $i \leq k$: Falls $m(v) = x_{index(i)}$, gehe über zum $\epsilon_{index(i)}$–Sohn von v, d.h. $v := v^{\epsilon_{index(i)}}$.*
   Erhöhe i um 1.

Der durch ϵ_{index} erreichbare Knoten ist dann der Knoten v, an dem man zum Schluß des Verfahrens angekommen ist.

Bemerkung 1.3 *Ist $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$, dann ist der durch $(\epsilon_{index(1)}, \ldots, \epsilon_{index(n)})$ erreichbare Knoten eine Senke s von G. Es gilt:*

$$f_F(\epsilon_1, \ldots, \epsilon_n) = m(s).$$

Man kann also f_F an der Stelle ϵ auswerten, indem man bei der Wurzel von G beginnt und einen Pfad bis zu einer Senke gemäß den Belegungen in ϵ verfolgt.

Bezeichnung 1.5 *Ist $f \in B_n$, so wird die Funktion $f_{x_{i_1}^{\epsilon_{i_1}} \ldots x_{i_k}^{\epsilon_{i_k}}}$ mit*

$$f_{x_{i_1}^{\epsilon_{i_1}} \ldots x_{i_k}^{\epsilon_{i_k}}}(x_1, \ldots, x_n) = f(y_1, \ldots, y_n),$$

$y_j = \epsilon_j$, falls $j \in \{i_1, \ldots, i_k\}$, $y_j = x_j$, falls $j \notin \{i_1, \ldots, i_k\}$ $(\forall (x_1, \ldots, x_n) \in \{0,1\}^n)$, als **Kofaktor** *von f hinsichtlich $x_{i_1}^{\epsilon_{i_1}} \ldots x_{i_k}^{\epsilon_{i_k}}$ bezeichnet.*

Bemerkung 1.4 *Sei v der durch $\epsilon_{index} = (\epsilon_{index(1)}, \ldots, \epsilon_{index(k)})$ mit $k \leq n$ erreichbare Knoten im* OBDD *$F = (G, m)$. Dann läßt sich der Untergraph von G, der alle Knoten umfaßt, die von v aus erreicht werden können, zusammen mit den zugehörigen Markierungen als* OBDD *interpretieren. Man sieht leicht, daß die durch diesen* OBDD *definierte Funktion gerade der Kofaktor $f_{x_{index(1)}^{\epsilon_{index(1)}} \ldots x_{index(k)}^{\epsilon_{index(k)}}}$ ist.*

Man kommt nun von einem OBDD zu einem *reduzierten* geordneten binären Entscheidungsgraphen (ROBDD), indem man durch Löschen redundanter Knoten und mehrfach vorkommender Teilgraphen im OBDD den OBDD verkleinert, ohne die durch ihn definierte Funktion zu verändern.

Definition 1.18 (Reduzierter geordneter Entscheidungsgraph (ROBDD))
Ein OBDD *$F = (G = (V, E), m)$ heißt genau dann* **reduziert**,

- *wenn es keinen Knoten gibt, dessen* 0*–Sohn und dessen* 1*–Sohn identisch sind und*
- *wenn es keine zwei Knoten* v_1 *und* v_2 *in* V *gibt, so daß der Untergraph aller Knoten, die von* v_1 *aus erreicht werden können und der Untergraph aller Knoten, die von* v_2 *aus erreicht werden können, isomorph sind.*
  Zwei OBDD*s* $F_1 = (G_1 = (V_1, E_1), m_1)$ *und* $F_2 = (G_2 = (V_2, E_2), m_2)$ *heißen hierbei isomorph, wenn sie sowohl in Struktur als auch Beschriftung übereinstimmen, d.h. wenn es eine bijektive Abbildung* $\sigma : V_1 \to V_2$ *gibt, so daß* $\forall v_1 \in V_1, v_2 \in V_2$ *mit* $\sigma(v_1) = v_2$ *gilt:* $m_1(v_1) = m_2(v_2)$ *und falls* v_1 *und* v_2 *keine Senken sind:* $\sigma({v_1}^0) = {v_2}^0$ *und* $\sigma({v_1}^1) = {v_2}^1$.

Lemma 1.1 (Bryant '86) *Aus einem* OBDD $F = (G = (V, E), m)$ *läßt sich in Zeit* $O(|V| \cdot \log(|V|))$ *der zugehörige* ROBDD $F' = (G', m')$ *bestimmen, der dieselbe Funktion definiert.*

Für ROBDDs kann man nun beweisen, daß sie (bei vorgegebener Variablenordnung) eine kanonische Darstellung Boolescher Funktionen sind.

Satz 1.1 (Bryant '86) *Für jede Boolesche Funktion* $f \in B_n$ *gibt es (bis auf Isomorphie) einen eindeutigen* ROBDD*, der* f *definiert (und jeder andere* OBDD*, der* f *definiert, hat mehr Knoten).*

Die folgende Bemerkung ergibt sich leicht aus Bemerkung 1.4 und Satz 1.1.

Bemerkung 1.5 *Ist* $F = (G, m)$ *ein reduzierter geordneter Entscheidungsgraph über der Variablenmenge* $\{x_1, \ldots, x_n\}$ *mit Variablenordnung* $<_{index}$*, der eine Funktion* f *definiert und sind 2 Kofaktoren*

$$f_{x_{index(1)}^{\epsilon_{index(1)}} \ldots x_{index(k)}^{\epsilon_{index(k)}}} \text{ und } f_{x_{index(1)}^{\delta_{index(1)}} \ldots x_{index(k)}^{\delta_{index(k)}}}$$

gleich, so sind die durch $(\epsilon_{index(1)}, \ldots, \epsilon_{index(k)})$ *bzw.* $(\delta_{index(1)}, \ldots, \delta_{index(k)})$ *erreichbaren Knoten von* G *identisch.*

Diese Tatsache wird später in Kapitel 3 bei der Herleitung von Zerlegungen Boolescher Funktionen aus ROBDD–Repräsentationen ausgenutzt.

Aussagen zur Komplexität einiger Grundoperationen auf ROBDDs sind in Tabelle 1.1 zu finden (vergleiche [Bry86]).

Eingabe	Ausgabe	Laufzeit
ROBDD $F = ((V,E),m)$ zu $f \in B_n$	ROBDD zu $\overline{f}$	$O(\|V\|)$
ROBDD $F_1 = ((V_1,E_1),m_1)$ zu f_1, ROBDD $F_2 = ((V_2,E_2),m_2)$ zu f_2	ROBDD zu $f_1 < op > f_2$, $op \in \{and, nand, or, exor, \ldots\}$	$O(\|V_1\| \cdot \|V_2\|)$
ROBDD $F = ((V,E),m)$ zu $f \in B_n$	ROBDD zu Kofaktor $f_{x_i^{\epsilon_i}}$, $\epsilon_i \in \{0,1\}$	$O(\|V\| \log(\|V\|))$
ROBDD $F = ((V,E),m)$ zu $f \in B_n$	beliebiges Element aus $ON(f)$	$O(n)$
ROBDD $F = ((V,E),m)$ zu $f \in B_n$	$ON(f)$	$O(n \cdot \|ON(f)\|)$
ROBDD $F = ((V,E),m)$ zu $f \in B_n$	$\|ON(f)\|$	$O(\|V\|)$

Tabelle 1.1: Komplexität einiger Grundoperationen auf ROBDDs

Variablenordnungen ROBDDs stellen eine kanonische Repräsentationsform Boolescher Funktionen dar, falls die zugehörige Variablenordnung vorgegeben ist. Dies bedeutet aber nicht, daß sich die ROBDDs bei Änderung der Variablenordnung nicht ändern.

Tatsächlich hängt die Größe der ROBDDs für viele Boolesche Funktionen sehr stark von der Variablenordnung ab. Man kann durch Änderung der Variablenordnung sogar von ROBDDs linearer Größe zu ROBDDs exponentieller Größe kommen bzw. umgekehrt. Ein Beispiel für eine solche Funktionsklasse wurde von Bryant in [Bry86] angegeben: es handelt sich um die Funktionsklasse aller $g_n \in B_{2n}$ mit $g_n(x_1, \ldots, x_{2n}) = x_1 \cdot x_2 + \ldots + x_{2n-1} \cdot x_{2n}$. Der ROBDD mit Variablenordnung $<_{index}$ mit $x_1 <_{index} x_2 <_{index} \ldots <_{index} x_{2n}$ hat die Größe $2n+2$, während der ROBDD mit Variablenordnung $<_{index'}$ mit $x_1 <_{index'} x_3 <_{index'} \ldots <_{index'} x_{2n-1} <_{index'} x_2 <_{index'} x_4 <_{index'} \ldots <_{index'} x_{2n}$ 2^{n+1} Knoten benötigt. In Abbildung 1.4 sind die ROBDDs für g_3 mit den Variablenordnungen $<_{index}$ und $<_{index'}$ angegeben.

Diese Tatsache stellt den Anwender von ROBDDs vor das Problem, zu gegebenen Booleschen Funktionen geeignete Variablenordnungen zu finden, unter denen sich die Funktionen möglichst kompakt als ROBDDs darstellen lassen. Häufig läßt sich aus dem Verständnis für die Struktur einer gegebenen Booleschen Funktion „von Hand" relativ leicht eine gute Variablenordnung finden. Es ist allerdings wünschenswert, daß eine solche Ordnung vom System automatisch gefunden wird, ohne daß der Anwender mit dieser Aufgabe belastet wird. Da sich das Problem, zu einer Booleschen Funktion eine Variablenordnung zu finden, die die ROBDD–Größe minimiert, als NP–vollständig erwiesen hat [BSM94], wird man im allgemeinen — zumindest für Boolesche Funktionen mit einer größeren Anzahl von Eingängen — Heuristiken zur Bestimmung einer Variablenordnung verwenden.

Die Heuristiken zur Bestimmung geeigneter Variablenordnungen lassen sich grob in zwei Klassen einteilen:

1. Heuristiken, die in der Lage sind, vor Aufbau eines ROBDD eine „Anfangsordnung" zu bestimmen. Diese Heuristiken leiten aus einer gegebenen Spezifikation einer Booleschen Funktion durch Analyse der Struktur dieser Spezifikation eine Variablenordnung her (z.B. [MWBS88, FFK88, FMK91, FOH93]).

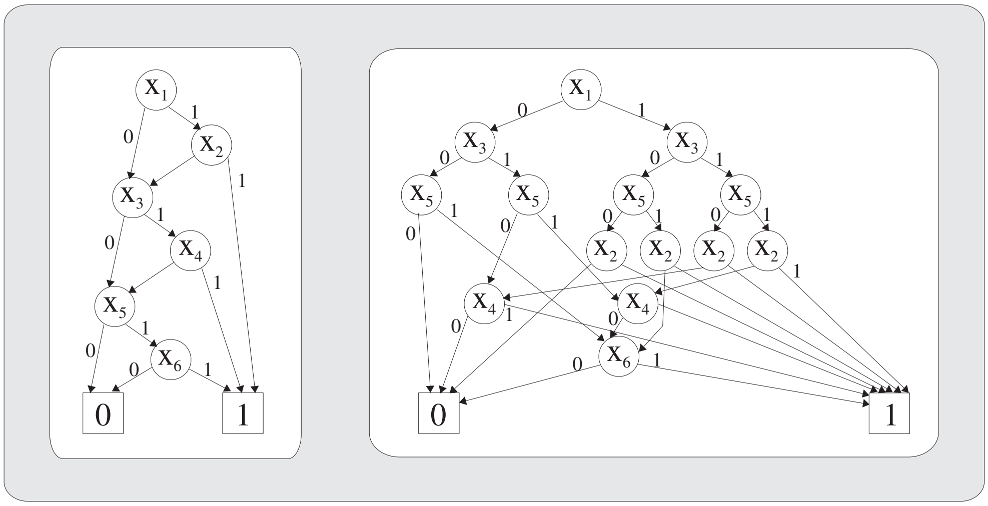


Abbildung 1.4: ROBDDs für $x_1 \cdot x_2 + x_3 \cdot x_4 + x_5 \cdot x_6$. Beim linken ROBDD gilt $x_1 <_{index} \ldots <_{index} x_6$, beim rechten $x_1 <_{index'} x_3 <_{index'} x_5 <_{index'} x_2 <_{index'} x_4 <_{index'} x_6$.

2. Heuristiken, die schon gegebene Variablenordnungen anhand schon aufgebauter ROBDDs verbessern (z.B. [ISY91, Rud93, FYBS93, BLM95, DBG95]). Diese Heuristiken haben den Vorteil, daß sie zusätzlich auch *dynamisch* eingesetzt werden können, d.h. nicht erst *nach* Aufbau des gewünschten ROBDD, sondern schon *während* des Aufbaus: Falls bei Aufbau eines ROBDD durch Boolesche Operationen aus „kleineren“ ROBDDs ein Zwischenergebnis zu groß wird, kann schon zu diesem Zeitpunkt die Variablenordnungsheuristik eingesetzt werden, um die Größe der schon berechneten Zwischenergebnisse zu minimieren und dann mit der gewonnenen neuen Variablenordnung weiterzuarbeiten.
   Die momentan wohl am häufigsten eingesetzte Variablenordnungsheuristik aus dieser Klasse ist der *sifting–Algorithmus* von Rudell [Rud93]. Er beruht stark auf der Tatsache, daß die Vertauschung zweier in der Variablenordnung $<_{index}$ benachbarter Variablen x_i und x_j eine „lokale“ Operation ist, d.h. bei Vertauschung eines solchen Variablenpaares in der Ordnung ändern sich nur die Knoten, die mit x_i und x_j beschriftet sind, alle anderen Knoten im ROBDD bleiben gleich. Weiterhin läßt sich eine solche Vertauschung effizient durchführen. Der sifting–Algorithmus beginnt mit einer Variablen x_i aus der Variablenmenge $\{x_1, \ldots, x_n\}$ und schiebt sie durch Variablenvertauschungen an alle n möglichen Positionen in der Ordnung. Danach wird die Variable an die Position in der Ordnung geschoben, an der die resultierende ROBDD–Größe am geringsten war. Danach wird das Verfahren mit der nächsten Variablen fortgesetzt, bis nach $O(n^2)$ Variablenvertauschungen die Position aller Variablen bestimmt ist.
   Obwohl im allgemeinen die Ergebnisse des sifting–Algorithmus sehr gut sind, wurde durch Panda/Somenzi [PS95] in experimentellen Ergebnissen ein Nachteil des

sifting–Verfahrens festgestellt: Häufig gibt es Gruppen von Variablen, die bei guten Variablenordnungen stets benachbart sind. Variablen aus solchen Gruppen tendieren dazu, durch den sifting–Algorithmus in ihrer Position nicht verändert zu werden. Angenommen es gibt eine starke „Anziehung“ zwischen zwei benachbarten Variablen x_i und x_j. Dann tendiert x_i bei der Verschiebung dazu, wieder in seine Position in der Nachbarschaft von x_j zurückzukehren und ebenso x_j bei seiner Verschiebung in der Variablenordnung. Als Resultat ist der sifting–Algorithmus nicht in der Lage, die *relativen* Positionen solcher Gruppen „zusammengehöriger“ Variablen zu optimieren. Ein Beispiel für solche „Anziehungskräfte“ zwischen Variablen sind *symmetrische* Variablen. (Eine Boolesche Funktion heißt symmetrisch in einem Paar x_i und x_j von Variablen, falls die Funktion sich bei Vertauschung von x_i und x_j nicht ändert, vgl. Kapitel 2.) Aus diesem Grund wurde von Möller/Molitor/Drechsler [MMD94] und Panda/Somenzi/Plessier [PSP94] „symmetrisches sifting“ eingeführt, bei dem nicht einzelne Variablen in der Variablenordnung verschoben werden, sondern *Gruppen* von Variablen, in denen die Boolesche Funktion symmetrisch ist. Hierbei wird auch die relative Position dieser Gruppen zueinander optimiert.

In Kapitel 2 wird ein Verfahren vorgestellt, das die Idee, symmetrische Variablen zusammenzugruppieren, ausnutzt, um möglichst kleine ROBDD–Repräsentationen *partieller* Funktionen zu erzeugen.

Repräsentation partieller Boolescher Funktionen durch ROBDDs Zum Abschluß dieses Abschnitts soll noch kurz auf Repräsentationsmöglichkeiten partieller Boolescher Funktionen durch ROBDDs eingegangen werden.

Eine naheliegende Idee besteht darin, im ROBDD außer den durch 0 und 1 beschrifteten Senken eine zusätzliche Senke einzuführen, die mit „dc“ beschriftet ist. Zu dieser Senke führen dann alle Pfade im ROBDD, die Belegungen aus der don't care–Menge entsprechen. Der resultierende reduzierte und geordnete Entscheidungsgraph wird im folgenden als DCBDD bezeichnet. Der erste Entscheidungsgraph aus Abbildung 1.5 zeigt eine solche Darstellung für die Funktion $f \in S(D)$ mit $D = \{(0,0,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)\}$,

$$f(\epsilon) = \begin{cases} 1, & falls\ \epsilon = (1,1,1) \\ 0, & falls\ \epsilon \in D \setminus \{(1,1,1)\}. \end{cases}$$

Da für jede partielle Boolesche Funktion f $DC(f) = \{0,1\}^n \setminus (ON(f) \cup OFF(f))$ gilt, genügt es, zur Darstellung einer solchen Funktion, zwei der drei Mengen $ON(f)$, $OFF(f)$ und $DC(f)$ darzustellen. Die angegebenen Mengen kann man beispielsweise durch ROBDDs für ihre charakteristischen Funktionen f_{on}, f_{off} und f_{dc} repräsentieren. Um die Darstellung von f_{on} und f_{off} in einem ROBDD zusammenzufassen, führen Chang/Marek–Sadowska [CCM94] außer den Eingangsvariablen $\{x_1,\ldots,x_n\}$ von f noch eine zusätzliche Variable z ein und definieren

$$ext(f) = \overline{z} \cdot f_{off} + z \cdot f_{on}. \tag{1.1}$$

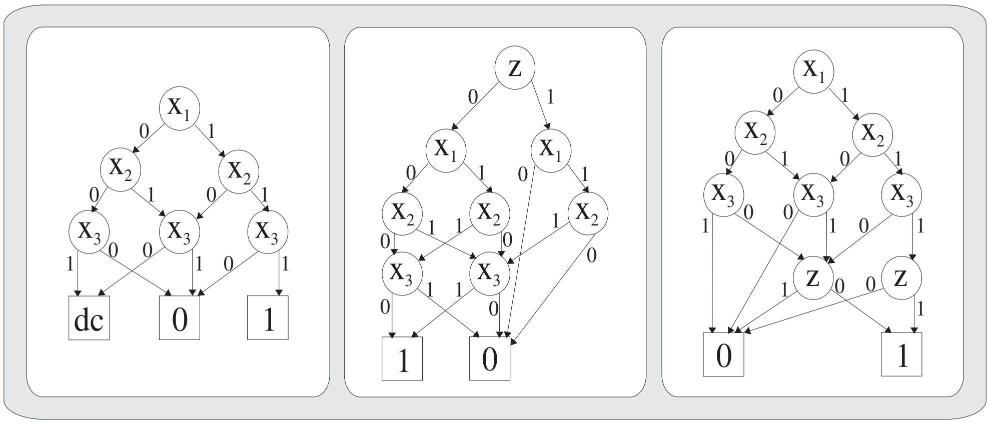


Abbildung 1.5: Darstellungen der partiellen Funktion f mit $f_{on}(x_1, x_2, x_3) = x_1 \cdot x_2 \cdot x_3$, $f_{off}(x_1, x_2, x_3) = \overline{x_1 \oplus x_2 \oplus x_3}$

Die partielle Funktion f wird dann durch den ROBDD für $ext(f)$ dargestellt. Der mittlere ROBDD aus Abbildung 1.5 zeigt die schon links dargestellte Beispielfunktion in dieser Form unter der Voraussetzung, daß die zusätzliche Variable z als erste in der Variablenordnung steht. Der rechte ROBDD zeigt die gleiche Funktion mit der Variablen z als letzte Variable in der Ordnung. Daß zwischen dieser Darstellung mit z als letzter Variable in der Ordnung und dem DCBDD kein prinzipieller Unterschied besteht, zeigt das folgende Lemma:

Lemma 1.2 *Sei $f \in S(D)$, $D \subseteq \{0,1\}^n$. Dann gilt für den DCBDD F_1 zu f mit der Variablenordnung $<_{index}$ und den ROBDD F_2 für $ext(f)$ mit der gleichen Variablenordnung (abgesehen davon, daß zusätzlich die Variable z als letzte in der Ordnung auftritt):*

$$size(F_2) - 2 \leq size(F_1) \leq size(F_2),$$

wobei $size(F_1)$ die Knotenanzahl in F_1 angibt, $size(F_2)$ die Knotenanzahl in F_2.

Beweis:
Sei $<_{index}$ die Variablenordnung von F_1.
Aus Gleichung 1.1 ergeben sich für ein $\epsilon_{index} \in \{0,1\}^n$ sofort folgende Äquivalenzen:
In F_1 wird durch ϵ_{index} der mit 0 beschriftete Knoten erreicht. $\Longleftrightarrow$ In F_2 wird durch ϵ_{index} der mit z beschriftete Knoten erreicht, dessen 0–Sohn mit 1 und dessen 1–Sohn mit 0 beschriftet ist.
In F_1 wird durch ϵ_{index} der mit 1 beschriftete Knoten erreicht. $\Longleftrightarrow$ In F_2 wird durch ϵ_{index} der mit z beschriftete Knoten erreicht, dessen 0–Sohn mit 0 und dessen 1–Sohn mit 1 beschriftet ist.
In F_1 wird durch ϵ_{index} der mit dc beschriftete Knoten erreicht. $\Longleftrightarrow$ In F_2 wird durch ϵ_{index}

der mit 0 beschriftete Knoten erreicht.
Man kann also durch folgende lokale Ersetzungen F_2 aus F_1 konstruieren:
Füge 2 neue Knoten v_0 mit $m(v_0) = 0$ und v_1 mit $m(v_1) = 1$ ein. Ersetze in F_1 den durch dc beschrifteten Knoten durch v_0. Ersetze (falls vorhanden) den alten Knoten aus F_1, der mit 0 beschriftet ist, durch einen neuen Knoten, der mit z beschriftet ist und dessen 0–Sohn v_1 und dessen 1–Sohn v_0 wird. Ersetze (falls vorhanden) den alten Knoten aus F_1, der mit 1 beschriftet ist, durch einen neuen Knoten, der mit z beschriftet ist und dessen 0–Sohn v_0 und dessen 1–Sohn v_1 wird. Falls v_0 oder v_1 keine eingehenden Kanten hat, lösche den entsprechenden Knoten wieder. □

1.2 Grundlagen zur Komplexität Boolescher Funktionen

Der nun folgende Abschnitt dient im wesentlichen zwei Zielen: Zum einen soll die Komplexität Boolescher Funktionen grundsätzlich untersucht werden, zum anderen soll geklärt werden, was man von Algorithmen zur Logiksynthese erwarten kann und was nicht.

Schon 1949 gelang es Shannon, mit einfachen Abzählargumenten folgenden Satz zu zeigen ([Sha49]):

Satz 1.2 (Shannon '49) *Für hinreichend große n haben „fast alle“ [2] Funktionen $f \in B_n$ eine Komplexität von*

$$C_{B_2}(f) \geq \frac{2^n}{n}.$$

Auf der anderen Seite gibt es einen Satz von Lupanov ([Lup58]), der folgendes besagt:

Satz 1.3 (Lupanov '58) *Zu jeder Funktion $f \in B_n$ gibt es eine Realisierung L_f (nämlich die Lupanov'sche k–s–Darstellung) mit*

$$C_{B_2}(L_f) \leq \frac{2^n}{n} + o(\frac{2^n}{n}).$$

Aus diesen beiden Sätzen kann man folgern, daß „fast alle“ Funktionen $f \in B_n$ fast die gleiche Komplexität wie die schwerste Funktion in B_n haben. Diesen Effekt nennt man nach Lupanov (70) den **Shannon–Effekt**.

An dieser Stelle könnte es auf den ersten Blick so aussehen, als ob sich eine weitere Behandlung des Themas erübrigen würde: Man kann zu fast allen Funktionen aus B_n mit der Lupanov'schen k–s–Darstellung eine nahezu optimale Realisierung angeben.

[2] Die Formulierung „fast alle Funktionen $f \in B_n$ haben Eigenschaft P“ ist gleichbedeutend mit der Aussage

$$|\{f \in B_n \mid f \text{ hat Eigenschaft } P \textit{ nicht}\}| \;/\; |B_n| \to 0 \text{ für } n \to \infty.$$

Daß man an dieser Stelle nicht aufhört, sondern nach anderen Syntheseverfahren sucht, hat folgende Gründe:

- Bei den eben vorgestellten Aussagen handelt es sich um *asymptotische* Resultate. Für kleine n kann die Komplexität einer Lupanov'schen k–s–Darstellung $\frac{2^n}{n}$ deutlich übersteigen.

- Bei den in der Praxis auftretenden Booleschen Funktionen handelt es sich meist *nicht* um zufällig gewürfelte Funktionen, sondern um Funktionen, die von einem menschlichen Benutzer spezifiziert worden sind (z.B. aus einem bestimmten algorithmischen Problem hervorgegangen sind). Daher weisen sie häufig gewisse *Regelmäßigkeiten* bzw. *Struktureigenschaften* auf. Sie sind fast immer mit weit weniger als $\frac{2^n}{n}$ Gattern zu realisieren.

Aufgabe von Logiksyntheseverfahren ist es nun, solche Struktureigenschaften zu erkennen und sie für die Realisierung auszunutzen.

Eine mögliche Struktureigenschaft einer Booleschen Funktion ist die *nichttriviale Zerlegbarkeit.* Ein Syntheseverfahren, das auf der Ausnutzung nichttrivialer Zerlegungen beruht, wird in Kapitel 3 vorgestellt.

Eine andere Möglichkeit ist die Ausnutzung von Symmetrien bzw. bestimmten Invarianzen bei der Synthese. Dies wird in den Kapiteln 2 und 3 näher erläutert. Eine ausführliche Darstellung von Logiksyntheseverfahren, die Symmetrien ausnutzen, findet sich in [Hot60] und [Hot74].

Ähnliche Resultate gelten auch für Funktionen mit mehreren Ausgängen (Funktionen aus $B_{n,m}$). Man kann nachweisen, daß „fast alle" Funktionen mit n Eingängen und m Ausgängen eine Komplexität $> m\frac{2^n}{n}$ haben. Allerdings gilt dies nur, wenn m nicht zu groß wird.

Mit Abzählargumenten (vgl. [Sch93a, SM93]) kann man folgenden Satz zeigen:

Satz 1.4 *Für genügend große n haben mindestens*

$$|B_{n,m}|\,(1-2^{-s(n)})$$

der $|B_{n,m}| = 2^{m\cdot 2^n}$ Funktionen aus $B_{n,m}$ eine Komplexität $> m\frac{2^n}{n}$, wobei $s(n) = \frac{2^n}{n^2}$[‡]*.*

Hierbei darf m allerdings nicht zu groß werden. Der Satz gilt für $m = o(n)$ oder für $m \le c\cdot n$ mit $c \le \frac{1}{(1+\delta)64e}$, $\delta > 0$.[§]

[‡]Man beachte, daß $2^{-s(n)}$ eine mit wachsendem n sehr schnell gegen 0 strebende Funktion ist. Die Formulierung „fast alle Funktionen haben eine Komplexität $> m\frac{2^n}{n}$" ist also berechtigt.

[§]Die Aussage des Satzes stimmt auch mit der Anschauung überein: Ist m nicht zu groß und wählt man für großes n m beliebige Funktionen $f_1, \ldots, f_m \in B_n$, so ist es unwahrscheinlich, daß man größere Teile aus der Realisierung einer Funktion f_i mit Vorteil bei der Realisierung von f_j $(j \ne i)$ verwenden kann. Die zufällig gewählten Funktionen sind „so verschieden", daß eine getrennte Realisierung der m Funktionen kaum teurer ist als eine gemeinsame Realisierung als Funktion mit m Ausgängen.

Ist m nicht zu groß, so haben also „fast alle“ Funktionen aus $B_{n,m}$ eine Komplexität $> m\frac{2^n}{n}$. Will man andererseits eine Funktion aus $B_{n,m}$ realisieren, so kann man leicht eine Realisierung mit Kosten $m\frac{2^n}{n} + o(m\frac{2^n}{n})$ finden, indem man die m Ausgangsfunktionen (unter Zuhilfenahme des Satzes von Lupanov) getrennt realisiert. Insofern tritt auch hier ein „Shannon–Effekt“ auf.

Bei zufällig gewählten Funktionen aus $B_{n,m}$ kann man infolgedessen nur für große m darauf hoffen, daß eine gemeinsame Realisierung der Ausgangsfunktionen Vorteile im Vergleich zu einer getrennten Realisierung hat.

Bei Funktionen, die in der Praxis auftreten, sieht die Situation allerdings ganz anders aus: Häufig weisen bei Funktionen mit mehreren Ausgängen die verschiedenen Ausgangsfunktionen eine ähnliche Struktur auf. Effiziente Realisierungen nutzen diese Tatsache aus, indem sie gleiche Teilschaltungen bei der Realisierung mehrerer Ausgangsfunktionen benutzen. Die Effizienz solcher Realisierungen beruht oft gerade auf der „Mehrfachverwendung“ gleicher Teilschaltungen.

Es lohnt sich also, bei der Logiksynthese nach Möglichkeiten zur Ausnutzung gemeinsamer Teilschaltungen für mehrere Ausgangsfunktionen zu suchen. Auch hierzu werden in Kapitel 3 Lösungen vorgestellt.

Kapitel 2

Symmetrien Boolescher Funktionen

Eine mögliche Struktureigenschaft Boolescher Funktionen, die bei der Logiksynthese mit Vorteil ausgenutzt werden kann, ist die Symmetrie. In diesem Kapitel soll im wesentlichen definiert werden, was Symmetrie ist und wie man bestimmte Symmetrien erkennen kann.

In Kapitel 3 wird dann unter anderem gezeigt, wie man Symmetrieeigenschaften bei der Logiksynthese ausnutzen kann.

Wie in Abschnitt 1.1.2.3 bereits angedeutet wurde, kann man Symmetrien Boolescher Funktionen bei der Minimierung von ROBDD–Größen ausnutzen. Dies ist eine wichtige Aufgabe, da ROBDDs eine grundlegende Datenstruktur sehr vieler Logiksynthesealgorithmen darstellen.

Der erste Teil des Kapitels beschäftigt sich mit Symmetrien totaler Funktionen. Mit Hinblick auf die Logiksynthese wird zu einer Booleschen Funktion f mit den Eingangsvariablen $x_1, \ldots, x_n$ eine möglichst kleine Partition $P = \{\lambda_1, \ldots, \lambda_k\}$ bestimmt, so daß für alle Variablenpaare (x_j, x_k) aus einer solchen Menge λ_i die Funktion invariant ist gegenüber Vertauschung von x_j und x_k.

Da bei vielen in der Praxis vorkommenden Problemen die Booleschen Funktionen aber nur unvollständig spezifiziert sind, ist der zweite Teil des Kapitels Symmetrien partieller Boolescher Funktionen gewidmet. Die Suche nach Erweiterungen partieller Boolescher Funktionen, die möglichst viele Symmetrieeigenschaften besitzen, ist deshalb von großer Bedeutung, weil symmetrische Funktionen im allgemeinen sehr viel leichter zu realisieren sind als allgemeine Funktionen (beispielsweise benötigt man für eine totalsymmetrische Funktion mit n Eingängen im worst case höchstens linear viele 2–Input–Gatter [Weg89], allgemein jedoch $\theta(\frac{2^n}{n})$ [Sha49]).
Bei partiellen Funktionen erweist sich jedoch das Problem, eine möglichst kleine Partition zu finden, in der die Funktion symmetrisch ist, als wesentlich schwieriger als bei totalen Funktionen. Anhand von Beispielen wird demonstriert, wo die Schwierigkeiten liegen. Es wird eine Heuristik angegeben, die (in Anlehnung an eine Heuristik zur Färbung von Graphen) zu einer partiellen Funktion f eine Belegung der don't cares liefert, so daß

die resultierende Funktion symmetrisch ist in einer möglichst kleinen Partition der Eingangsvariablen. Das hier eingeführte Konzept der *starken Symmetrie* partieller Boolescher Funktionen erfüllt hierbei die Aufgabe, den speziell im Zusammenhang mit Symmetrien *partieller* Funktionen auftretenden Problemen zu begegnen.
Die Resultate werden von Vertauschungssymmetrien verallgemeinert auch auf sogenannte Äquivalenzsymmetrien (d.h. Symmetrien, bei denen vor Vertauschung zweier Eingangsvariablen eine der beiden Variablen negiert wird).
Schließlich wird noch gezeigt, wie die Verfahren zur Minimierung von ROBDD–Größen partieller Boolescher Funktionen angewendet werden können.

2.1 Symmetrien totaler Boolescher Funktionen

Grundlegende Resultate zu Booleschen Algebren, Homomorphismen Boolescher Algebren und Symmetrien Boolescher Funktionen sind in [Hot74] zu finden. Hier sollen nur kurz einige Definitionen zu Symmetrien Boolescher Funktionen wiederholt werden.

Definition 2.1 (Invarianz unter τ) *Sei $f \in B_n$, τ sei eine Permutation auf $\{0,1\}^n$. f heißt* invariant unter τ *genau dann, wenn für alle $(x_1,\dots,x_n) \in \{0,1\}^n$ $f(x_1,\dots,x_n) = f(\tau(x_1,\dots,x_n))$.*

Definition 2.2 (Invarianz unter G) *Sei $f \in B_n$, $\mathbf{S}_{\{0,1\}^n}$ die Gruppe aller Permutationen auf $\{0,1\}^n$. Sei G eine Teilmenge von $\mathbf{S}_{\{0,1\}^n}$. f heißt* invariant unter G *genau dann, wenn f invariant ist unter allen $\tau \in G$.*

Wenn man Invarianz unter Teilmengen G von $\mathbf{S}_{\{0,1\}^n}$ betrachtet, dann kann man sich auf Untergruppen von $\mathbf{S}_{\{0,1\}^n}$ beschränken. Es gilt nämlich das folgende Lemma:

Lemma 2.1 *Falls $f \in B_n$ invariant ist unter $G \subseteq \mathbf{S}_{\{0,1\}^n}$, dann ist f auch invariant unter $<G>$, der durch G erzeugten Untergruppe von $\mathbf{S}_{\{0,1\}^n}$.*

Beweis: (siehe [Hot74])
Seien τ_1 und $\tau_2 \in G$.
f ist invariant unter $\tau_1 \circ \tau_2$, da $\forall (x_1,\dots,x_n) \in \{0,1\}^n$ gilt:

$$\begin{aligned} f((\tau_1 \circ \tau_2)(x_1,\dots,x_n)) &= f(\tau_1(\tau_2(x_1,\dots,x_n))) \\ &= f(\tau_2(x_1,\dots,x_n)) \qquad \text{(da } f \text{ invariant unter } \tau_1) \\ &= f(x_1,\dots,x_n) \qquad \text{(da } f \text{ invariant unter } \tau_2) \end{aligned}$$

Falls $\tau \in G$, dann ist f auch invariant unter τ^{-1}, da $\forall (x_1,\dots,x_n) \in \{0,1\}^n$ gilt:

$$\begin{aligned} f(\tau^{-1}(x_1,\dots,x_n)) &= f(\tau(\tau^{-1}(x_1,\dots,x_n))) \quad \text{(da } f \text{ invariant unter } \tau) \\ &= f(x_1,\dots,x_n) \end{aligned}$$

$\Longrightarrow \quad f$ ist invariant unter $<G>$. $\square$

Hat man eine Untergruppe G von $\mathbf{S}_{\{0,1\}^n}$ gegeben, so kann man Teilmengen von $\{0,1\}^n$ bestimmen, so daß jede Boolesche Funktion f, die invariant ist unter G, auf diesen Teilmengen den gleichen Funktionswert liefern muß. Unter diesen Teilmengen spielen die sog. **Orbits** eine besondere Rolle:
Die Anwendung von Permutationen aus einer Gruppe $G \subseteq \mathbf{S}_{\{0,1\}^n}$ auf die Elemente aus $\{0,1\}^n$ teilt $\{0,1\}^n$ in Orbits ein:
Ein Orbit eines Elementes $\alpha \in \{0,1\}^n$ ist definiert als

$$orbit_G(\alpha) = \{\beta \in \{0,1\}^n \mid \exists \tau \in G \text{ mit } \tau(\alpha) = \beta\}$$

Definiert man durch

$$„\alpha \sim \beta \iff \beta \in orbit_G(\alpha) \iff \exists \sigma \in G \text{ mit } \sigma(\alpha) = \beta“$$

eine Relation auf $\{0,1\}^n$, so folgt aus der Tatsache, daß G eine Gruppe ist, leicht, daß „$\sim$“ eine Äquivalenzrelation ist. Die Orbits der Elemente von $\{0,1\}^n$ sind gerade die Äquivalenzklassen dieser Relation und bilden folglich eine Partition auf $\{0,1\}^n$.

Sei $\beta \in orbit_G(\alpha)$ beliebig. Dann gibt es nach Definition ein $\tau \in G$, so daß $\tau(\alpha) = \beta$. Falls $f \in B_n$ invariant ist unter G, so muß gelten:

$$f(\beta) = f(\tau(\alpha)) \overset{f \text{ invariant unter } \tau}{=} f(\alpha)$$

Also gilt folgendes Lemma:

Lemma 2.2 *Falls G Untergruppe von $\mathbf{S}_{\{0,1\}^n}$, f invariant unter G, dann gilt $\forall \alpha \in \{0,1\}^n$:*

$$orbit_G(\alpha) \subseteq ON(f) \quad \textit{oder} \quad orbit_G(\alpha) \subseteq OFF(f).$$

Also gilt:

$$ON(f) = \bigcup_{\alpha \in ON(f)} orbit_G(\alpha) \quad \textit{und} \quad OFF(f) = \bigcup_{\alpha \in OFF(f)} orbit_G(\alpha).$$

Aus Lemma 2.2 folgt direkt, daß die Funktionen

$$f_{orbit_G(\alpha)} \text{ mit } f_{orbit_G(\alpha)}(x_1, \ldots, x_n) = 1 \text{ gdw. } (x_1, \ldots, x_n) \in orbit_G(\alpha) \quad (\alpha \in \{0,1\}^n)$$

unter den Funktionen, die invariant unter G sind, die einzigen sind, deren ON–Menge man nicht verkleinern kann, ohne daß die Invarianz unter G verloren geht.
Es folgt ebenfalls, daß sich jede Funktion f, die invariant ist unter G, eindeutig als Disjunktion von Funktionen $f_{orbit_G(\alpha)}$ darstellen läßt.

Bemerkung 2.1 *(vergleiche [Hot74])*

- *Die Funktionen* $f \in B_n$*, die invariant sind unter einer Untergruppe* G *von* $\mathbf{S}_{\{0,1\}^n}$*, bilden eine Unteralgebra von* B_n*. Die Funktionen* $f_{orbit_G(\alpha)}$ *sind gerade die Atome dieser Unteralgebra.*
- *Eine Permutation* $\tau \in \mathbf{S}_{\{0,1\}^n}$ *induziert einen Automorphismus* h_τ *auf* B_n*, der definiert ist durch:*

  $$h_\tau : B_n \to B_n, h_\tau(f) = g \text{ mit } g(\tau(x_1, \ldots, x_n)) = f(x_1, \ldots, x_n)\ \forall (x_1, \ldots, x_n) \in \{0,1\}^n.$$

Im folgenden soll wie in [Hot74] bei der Betrachtung der Permutationen τ eine Beschränkung auf eine spezielle Klasse von Permutationen erfolgen, nämlich auf solche, die durch „Variablennegierungen“ und „Variablenvertauschungen“ erzeugt werden. Die betrachteten Permutationen stammen dann aus einer Untergruppe $\mathbf{P}_n$ von $\mathbf{S}_{\{0,1\}^n}$:
$\mathbf{P}_n$ ist die Untergruppe von $\mathbf{S}_{\{0,1\}^n}$, die durch die Permutationen

$$\sigma_{ik}, \quad 1 \le i < k \le n, \text{ und}$$

$$\nu_i, \quad 1 \le i \le n$$

erzeugt wird, wobei $\forall \alpha \in \{0,1\}^n$

$$\sigma_{ik}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_k, \ldots, \alpha_n) = (\alpha_1, \ldots, \alpha_k, \ldots, \alpha_i, \ldots, \alpha_n),$$

$$\nu_i(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) = (\alpha_1, \ldots, \overline{\alpha_i}, \ldots, \alpha_n).$$

Funktionen, die invariant sind unter Untergruppen von $\mathbf{P}_n$, nennt man dann G–symmetrisch:

Definition 2.3 (G–Symmetrie)
$f \in B_n$ heißt G–symmetrisch für $G \subseteq \mathbf{P}_n$, falls f invariant ist unter G.

Etliche allgemein gebräuchliche Symmetriedefinitionen sind als Spezialfälle der G–Symmetrie anzusehen (vgl. [EH78]):

Bezeichnung 2.1

- *Ist* f $\{\sigma_{ik} \mid 1 \le i < k \le n\}$*–symmetrisch, so bezeichnet man* f *auch als totalsymmetrisch.*
  Es gilt dann

  $$f(\alpha_1, \ldots, \alpha_n) = f(\beta_1, \ldots, \beta_n)\ \forall \alpha, \beta \text{ mit } \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \beta_i.$$

- *Ist* f $\{\sigma_{ik}\}$*–symmetrisch für* $1 \le i < k \le n$*, so bezeichnet man* f *als vertauschungssymmetrisch (oder auch nur symmetrisch) in den Variablen* (x_i, x_k)*.*
  Es gilt dann $\forall \alpha_l \in \{0,1\}$, $l \in \{1, \ldots, n\} \setminus \{i,k\}$

  $$f(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_{k-1}, 1, \alpha_{k+1}, \ldots, \alpha_n) = f(\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_{k-1}, 0, \alpha_{k+1}, \ldots, \alpha_n).$$

- *Ist f $\{\sigma_{ik} \mid 1 \leq i < k \leq n;\ \ i,k \in \lambda\}$–symmetrisch ($\lambda \subseteq \{1,\ldots,n\}$, $|\lambda| > 1$), so bezeichnet man f als (vertauschungs)symmetrisch in der Variablenmenge λ.*
  Es gilt dann
  $$f(\alpha_1,\ldots,\alpha_n) = f(\beta_1,\ldots,\beta_n)$$
  $$\forall \alpha,\ \beta \ mit \sum_{i\in\lambda} \alpha_i = \sum_{i\in\lambda} \beta_i \ und \ \alpha_i = \beta_i \ für \ i \in \{1,\ldots,n\} \setminus \lambda.$$

- *Ist f $\{\sigma_{ik} \mid 1 \leq i < k \leq n;\ \ i,k \in \lambda_j, j \in \{1,\ldots,l\}\}$–symmetrisch ($\{\lambda_1,\ldots,\lambda_l\}$ Partition von $\{1,\ldots,n\}$), so bezeichnet man f als (vertauschungs)symmetrisch in der Variablenpartition $\lambda = \{\lambda_1,\ldots,\lambda_l\}$.*
  Es gilt dann
  $$f(\alpha_1,\ldots,\alpha_n) = f(\beta_1,\ldots,\beta_n)$$
  $$\forall \alpha,\ \beta \ mit \sum_{i\in\lambda_j} \alpha_i = \sum_{i\in\lambda_j} \beta_i \ für \ alle \ 1 \leq j \leq l.$$

- *Ist f $\{\nu_i \circ \sigma_{ik} \circ \nu_i\}$–symmetrisch (bzw. $\{\nu_k \circ \sigma_{ik} \circ \nu_k\}$–symmetrisch) für $1 \leq i < k \leq n$, so bezeichnet man f als äquivalenzsymmetrisch in den Variablen (x_i, x_k).*
  Es gilt dann $\forall \alpha_l \in \{0,1\}$, $l \in \{1,\ldots,n\} \setminus \{i,k\}$
  $$f(\alpha_1,\ldots,\alpha_{i-1},0,\alpha_{i+1},\ldots,\alpha_{k-1},0,\alpha_{k+1},\ldots,\alpha_n) = f(\alpha_1,\ldots,\alpha_{i-1},1,\alpha_{i+1},\ldots,\alpha_{k-1},1,\alpha_{k+1},\ldots,\alpha_n).$$

- *Ist f $\{\sigma_{ik}, \nu_i \circ \sigma_{ik} \circ \nu_i\}$–symmetrisch, d.h. sowohl vertauschungssymmetrisch in (x_i, x_k) als auch äquivalenzsymmetrisch in (x_i, x_k), dann bezeichnet man f als mehrfachsymmetrisch in (x_i, x_k).*

- *Ist f $\{\nu_i\}$–symmetrisch für $1 \leq i \leq n$, so ist f unabhängig von der Eingangsvariablen x_i.*
  Es gilt dann $\forall \alpha_l \in \{0,1\}$, $l \in \{1,\ldots,n\} \setminus \{i\}$
  $$f(\alpha_1,\ldots,\alpha_{i-1},0,\alpha_{i+1},\ldots,\alpha_n) = f(\alpha_1,\ldots,\alpha_{i-1},1,\alpha_{i+1},\ldots,\alpha_n).$$

Will man bei der Logiksynthese bestimmte Symmetrieeigenschaften Boolescher Funktionen (siehe Kapitel 3) ausnutzen, so muß man in einem ersten Schritt in der Lage sein, vorhandene Symmetrieeigenschaften einer Booleschen Funktion zu erkennen. In den beiden folgenden Abschnitten wird zunächst das Erkennen von Symmetrien, die durch Vertauschungssymmetrien erzeugt sind, behandelt, dann das Erkennen von Symmetrien, die durch Vertauschungssymmetrien oder Äquivalenzsymmetrien erzeugt sind.

2.1.1 Erkennen von Vertauschungssymmetrien

Eine Struktureigenschaft Boolescher Funktionen, die bei der Logiksynthese ausgenutzt werden kann, ist die (Vertauschungs)symmetrie in möglichst großen Variablenmengen bzw.

die Symmetrie in möglichst kleinen Variablenpartitionen (d.h. in Variablenpartitionen mit möglichst wenig verschiedenen Mengen) (siehe Kapitel 3).

Bei der Suche nach möglichst großen Variablenmengen bzw. möglichst kleinen Variablenpartitionen, in denen eine Funktion symmetrisch ist, ist es von Vorteil, daß die Symmetrie in Variablenpaaren transitiv ist. Ist eine Funktion f symmetrisch in den Variablenpaaren (x_i, x_j) und (x_j, x_k), so ist sie auch symmetrisch in (x_i, x_k) und folglich in der Variablenmenge $\{x_i, x_j, x_k\}$. Definiert man für eine Funktion $f \in B_n$ die Relation $\sim_{sym}$ auf der Variablenmenge $X = \{x_1, \ldots, x_n\}$ durch

$$x_i \sim_{sym} x_j \quad \Longleftrightarrow \quad f \text{ ist (vertauschungs)symmetrisch in } (x_i, x_j),$$

dann ist $\sim_{sym}$ eine Äquivalenzrelation auf X. Die Menge der Äquivalenzklassen dieser Relation stellt eine Partition auf X dar. Die einzelnen Äquivalenzklassen sind die größtmöglichen Variablenmengen, in denen f symmetrisch ist. Angenommen, die Symmetrierelation $\sim_{sym}$ liefert auf den Eingangsvariablen die Partition $P_{\sim_{sym}} = \{\mu_1, \ldots, \mu_k\}$. Dann ist f genau dann in einer gegebenen Variablenmenge λ symmetrisch, wenn $\lambda \subseteq \mu_i$ für $1 \leq i \leq k$. $P_{\sim_{sym}}$ ist die minimale Partition, in der f symmetrisch ist, und $P_{\sim_{sym}}$ gibt Aufschluß über die maximalen Mengen μ_i, in denen f symmetrisch ist.

Der Symmetriegraph $G_{sym} = (X, E)$ mit $(x_i, x_j) \in E$ genau dann, wenn $x_i \sim_{sym} x_j$, hat demzufolge bei totalen Funktionen (im Gegensatz zu partiellen Funktionen, siehe Abschnitt 2.2) eine spezielle Struktur: Die Zusammenhangskomponenten von G_{sym} sind zugleich Cliquen des Graphen.

Das Problem wurde also darauf reduziert, auf Symmetrie in Variablenpaaren zu testen. Dazu genügen $\binom{n}{2} = \frac{1}{2}(n^2 - n)$ Symmetrietests. Um $P_{\sim_{sym}}$ zu berechnen, beginnt man mit einer Partition der Eingangsvariablen, bei der jede Variable in einer eigenen Menge ist. Nun führt man für sämtliche Variablenpaare einen Symmetrietest durch. Stellt man Symmetrie in (x_i, x_j) fest, so vereinigt man die beiden Mengen der Partition, die x_i und x_j enthalten. (Zur praktischen Durchführung bietet sich hierzu eine Union–Find–Datenstruktur an.) Am Ende erhält man die Äquivalenzklasseneinteilung $P_{\sim_{sym}} = \{\mu_1, \ldots, \mu_k\}$.

Ist f durch einen ROBDD G_f gegeben, so genügt es zum Test auf Symmetrie in (x_i, x_j), die beiden Kofaktoren $f_{x_i \overline{x_j}}$ und $f_{\overline{x_i} x_j}$ zu berechnen und zu testen, ob die beiden Kofaktoren gleich sind. Somit kann der Test in $O(size(G_f) \cdot \log(size(G_f)))$ durchgeführt werden (vergleiche Kapitel 1). In [MMW93] ist ein Verfahren angegeben, das nicht für alle $\binom{n}{2} = \frac{1}{2}(n^2 - n)$ Variablenpaare diese Kofaktorberechnung durchführt, sondern schon vorher anhand einfacherer Tests hinsichtlich der Struktur des ROBDD G_f Variablenpaare ausschließt, in denen f nicht symmetrisch sein kann. Erst dann werden die noch verbleibenden Variablenpaare auf Symmetrie getestet.

2.1.2 Erkennen von Vertauschungssymmetrien *und* Äquivalenzsymmetrien

Im vorliegenden Abschnitt wird untersucht, wie sich die beschriebene Situation verändert, wenn man außer auf Vertauschungssymmetrie auch auf Äquivalenzsymmetrie untersucht.

Leider liefert die paarweise Äquivalenzsymmetrie auf einer Variablenmenge X keine Äquivalenzrelation. Dies wird durch das folgende Gegenbeispiel belegt:

Beispiel 2.1

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Man sieht leicht, daß f sowohl in (x_1, x_2) als auch in (x_2, x_3) äquivalenzsymmetrisch ist. f ist aber wegen $f(0,0,0) \neq f(1,0,1)$ *nicht* äquivalenzsymmetrisch in (x_1, x_3).
f ist aber vertauschungssymmetrisch in (x_1, x_3). Das folgende Lemma zeigt, daß dies allgemein gilt:

Lemma 2.3 *Sei $f \in B_n$. Falls f äquivalenzsymmetrisch in den Variablenpaaren (x_i, x_j) und (x_j, x_k) (x_i, x_j, x_k alle verschieden), dann ist f vertauschungssymmetrisch in (x_i, x_k).*

Beweis: Z.z.: $\forall \epsilon_l \in \{0,1\}$, $l \in \{1, \ldots, n\} \setminus \{i, k\}$

$$f(\epsilon_1, \ldots, \epsilon_{i-1}, 0, \epsilon_{i+1}, \ldots, \epsilon_{k-1}, 1, \epsilon_{k+1}, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_{i-1}, 1, \epsilon_{i+1}, \ldots, \epsilon_{k-1}, 0, \epsilon_{k+1}, \ldots, \epsilon_n)$$

1. Fall: $\epsilon_j = 0$

$$\begin{aligned} f(\ldots, \underbrace{0}_{\epsilon_i}, \ldots, \underbrace{0}_{\epsilon_j}, \ldots, \underbrace{1}_{\epsilon_k}, \ldots) &= f(\ldots, \underbrace{1}_{\epsilon_i}, \ldots, \underbrace{1}_{\epsilon_j}, \ldots, \underbrace{1}_{\epsilon_k}, \ldots) \\ &\quad \text{(da } f \text{ äquivalenzsymm. in } (x_i, x_j)) \\ &= f(\ldots, \underbrace{1}_{\epsilon_i}, \ldots, \underbrace{0}_{\epsilon_j}, \ldots, \underbrace{0}_{\epsilon_k}, \ldots) \\ &\quad \text{(da } f \text{ äquivalenzsymm. in } (x_j, x_k)) \end{aligned}$$

2. Fall: $\epsilon_j = 1$

$$\begin{aligned} f(\ldots,\underbrace{0}_{\epsilon_i},\ldots,\underbrace{1}_{\epsilon_j},\ldots,\underbrace{1}_{\epsilon_k},\ldots) &= f(\ldots,\underbrace{0}_{\epsilon_i},\ldots,\underbrace{0}_{\epsilon_j},\ldots,\underbrace{0}_{\epsilon_k},\ldots) \\ &\quad \text{(da } f \text{ äquivalenzsymm. in } (x_j, x_k)) \\ &= f(\ldots,\underbrace{1}_{\epsilon_i},\ldots,\underbrace{1}_{\epsilon_j},\ldots,\underbrace{0}_{\epsilon_k},\ldots) \\ &\quad \text{(da } f \text{ äquivalenzsymm. in } (x_i, x_j)) \end{aligned}$$

□

Analog zeigt man:

Lemma 2.4 *Sei $f \in B_n$. Falls f äquivalenzsymmetrisch in (x_i, x_j) und vertauschungssymmetrisch in (x_j, x_k), dann ist f äquivalenzsymmetrisch in (x_i, x_k).*

Falls eine Funktion f äquivalenzsymmetrisch in einem Variablenpaar (x_i, x_j) ist, dann ist die Funktion f', die man erhält, wenn man die Variable x_i negiert (oder x_j negiert), (vertauschungs)symmetrisch in (x_i, x_j). Wenn man in Beispiel 2.1 x_1 und x_3 negiert (oder x_2 negiert), so ist die resultierende Funktion (vertauschungs)symmetrisch in (x_1, x_2) und (x_2, x_3), also totalsymmetrisch.

Ziel ist es nun, durch Negation einer bestimmten Auswahl an Eingangsvariablen zu einer Funktion zu kommen, die in einer möglichst kleinen Partition vertauschungssymmetrisch ist.
Im folgenden wird gezeigt, daß es hierbei eine eindeutige minimale Partition gibt (in dem Sinne, daß jede andere solche Partition eine Verfeinerung davon ist) und wie man eine solche Partition erhält (zusammen mit einer geeigneten Negation von Variablen).

Definition 2.4 *Gegeben sei eine Boolesche Funktion $f \in B_n$ und ein „Polaritätsvektor“ $P = (p_1, \ldots, p_n)$. Die aus f durch den Polaritätsvektor P erzeugte Funktion $f_P \in B_n$ ist definiert durch $f_P(x_1, \ldots, x_n) = f(x_1^{p_1}, \ldots, x_n^{p_n})\ \forall (x_1, \ldots, x_n) \in \{0,1\}^n$.*

Gesucht ist also ein „optimaler“ Polaritätsvektor und die zugehörige Variablenpartition.
Zu diesem Zweck wird „erweiterte Symmetrie“ definiert:

Definition 2.5 *Eine Funktion $f \in B_n$ heißt* erweitert symmetrisch *im Variablenpaar (x_i, x_j) genau dann, wenn f vertauschungssymmetrisch* oder *äquivalenzsymmetrisch in (x_i, x_j) ist.*

Lemma 2.5 *Sei $f \in B_n$. Die Relation $\sim_{esym}$ auf der Variablenmenge $X = \{x_1, \ldots, x_n\}$, die definiert ist durch*

$$x_i \sim_{esym} x_j \iff f \text{ ist erweitert symmetrisch in } (x_i, x_j) \quad \forall i, j \in \{1, \ldots n\},$$

ist eine Äquivalenzrelation auf X.

Beweis: Die Aussage folgt aus der Tatsache, daß $\sim_{sym}$ transitiv ist und aus den Lemmata 2.3 und 2.4. □

Der Symmetriegraph G_{esym} für erweiterte Symmetrie ist definiert wie der Symmetriegraph im Fall von Vertauschungssymmetrie. Zusätzlich wird jeder Kante noch ein Typ zugeordnet, nämlich „vertauschungssymmetrisch", falls f *nicht* äquivalenzsymmetrisch im zugehörigen Variablenpaar ist, „äquivalenzsymmetrisch", falls f *nicht* vertauschungssymmetrisch in dem Variablenpaar ist und „mehrfachsymmetrisch" sonst.

Satz 2.1 *Sei $f \in B_n$. Es gibt einen Polaritätsvektor P, so daß f_P vertauschungssymmetrisch ist in der Partition $P_{\sim_{esym}}$, die durch die Äquivalenzklassen der Relation $\sim_{esym}$ zu f gebildet wird.*

Beweis: Konstruiere Polaritätsvektor P, so daß f_P vertauschungssymmetrisch in $P_{\sim_{esym}}$.
Sei $\mu_i = \{x_{i_1}, \ldots, x_{i_k}\} \in P_{\sim_{esym}}$.
Wähle dann $p_{i_1} = \epsilon$ beliebig. Falls f in (x_{i_1}, x_{i_2}) vertauschungssymmetrisch, dann wähle $p_{i_2} = \epsilon$, falls f in (x_{i_1}, x_{i_2}) äquivalenzsymmetrisch, dann wähle $p_{i_2} = \overline{\epsilon}$. Führe das Verfahren analog fort mit (x_{i_2}, x_{i_3}) usw. bis p_{i_k} bestimmt ist.
Offensichtlich ist f_P dann vertauschungssymmetrisch in μ_i. □

Wählt man einen Polaritätsvektor nach der Konstruktion im vorangegangenen Beweis, so hat der Symmetriegraph G_{sym} für f_P genau die gleichen Kanten wie der Symmetriegraph G_{esym} für f (hinsichtlich erweiterter Symmetrie). Wählt man einen anderen Polaritätsvektor P', so können in G_{sym} im Vergleich zu G_{esym} höchstens Kanten wegfallen, nämlich Kanten, die in G_{esym} den Typ „vertauschungssymmetrisch" haben, aber Knoten verbinden, denen unterschiedliche Polarität zugeordnet wurde und Kanten, die in G_{esym} den Typ „äquivalenzsymmetrisch" haben, und Knoten verbinden, denen gleiche Polarität zugeordnet wurde. Folglich kann $f_{P'}$ nur in Verfeinerungen von $P_{\sim_{esym}}$ vertauschungssymmetrisch sein. Abbildung 2.1 illustriert dies anhand eines Beispielgraphen G_{esym} einer Funktion $f \in B_9$ und des dazugehörigen Symmetriegraphen G_{sym} zu $f_{P'}$ mit $P' = (1, 0, 0, 1, 1, 1, 1, 1, 0)$.

Zum Abschluß dieses Abschnitts soll noch (in Hinblick auf eine Anwendung in Kapitel 3 über Zerlegungen) die Struktur von Symmetriegraphen für erweiterte Symmetrie genauer betrachtet werden:

In Abbildung 2.2 sind alle möglichen Teilgraphen mit 3 Knoten eines solchen Symmetriegraphen dargestellt.

Daß keine anderen Teilgraphen auftreten können, folgt aus der Transitivität von $\sim_{sym}$, den Lemmata 2.3 und 2.4 und dem nun folgenden Lemma.

Lemma 2.6 *Ist f mehrfachsymmetrisch in (x_i, x_j) und vertauschungssymmetrisch (äquivalenzsymmetrisch) in (x_j, x_k), dann ist f auch mehrfachsymmetrisch in (x_i, x_k) und in (x_j, x_k).*

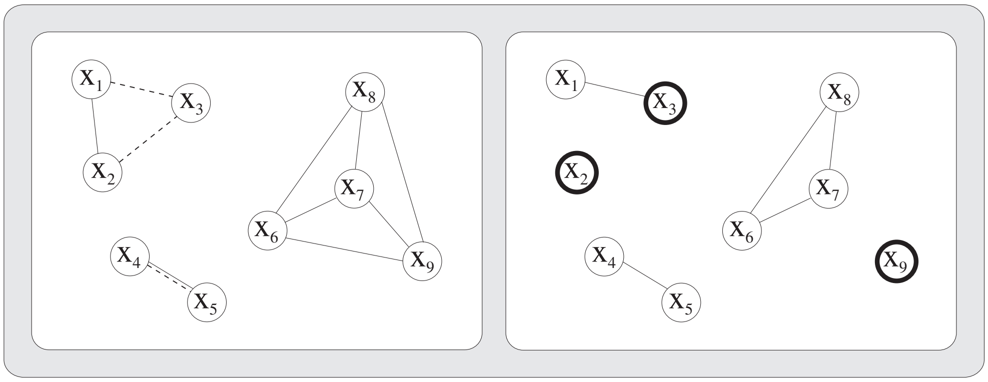


Abbildung 2.1: Linker Graph: G_{esym} einer Funktion $f \in B_9$. Durchgezogene Kanten stehen für „vertauschungssymmetrisch", gestrichelte Kanten für „äquivalenzsymmetrisch" und „Doppelkanten" (sowohl durchgezogen als auch gestrichelte) für „mehrfachsymmetrisch". Rechter Graph: dazugehöriger Symmetriegraph G_{sym} zu $f_{P'}$ mit $P' = (1, 0, 0, 1, 1, 1, 1, 1, 0)$. Negierte Variablen sind durch fett umrandete Knoten dargestellt.

Beweis: Die Aussage folgt aus der Transitivität von $\sim_{sym}$ und aus den Lemmata 2.3 und 2.4. □

Folglich können nur 2 Arten von Cliquen im Symmetriegraph auftreten:

1. Cliquen, in denen vertauschungssymmetrische und äquivalenzsymmetrische Variablenpaare auftreten, aber keine mehrfachsymmetrischen.

2. Cliquen, in denen ausschließlich mehrfachsymmetrische Variablenpaare auftreten.

2.2 Symmetrien partieller Boolescher Funktionen

Im folgenden Abschnitt wird die Betrachtung von Symmetrie auf partielle Boolesche Funktionen erweitert. Ein Teil der vorliegenden Resultate ist im Rahmen einer Diplomarbeit [Mel96] in Zusammenarbeit mit Herrn Stefan Melchior entstanden.

Ein zentraler Begriff im Zusammenhang mit partiellen Booleschen Funktionen ist die *Erweiterung* einer Booleschen Funktion.

Definition 2.6 (Erweiterung einer partiellen Booleschen Funktion)
Sei $f \in S(D)$, $D \subseteq \{0,1\}^n$. $f' \in S(D')$ mit $D \subseteq D'$ heißt Erweiterung von f, *falls $f'(x) = f(x)$ $\forall x \in D$ $(f'|_D = f)$. Falls $D' = \{0,1\}^n$, dann heißt f'* totale Erweiterung von f.

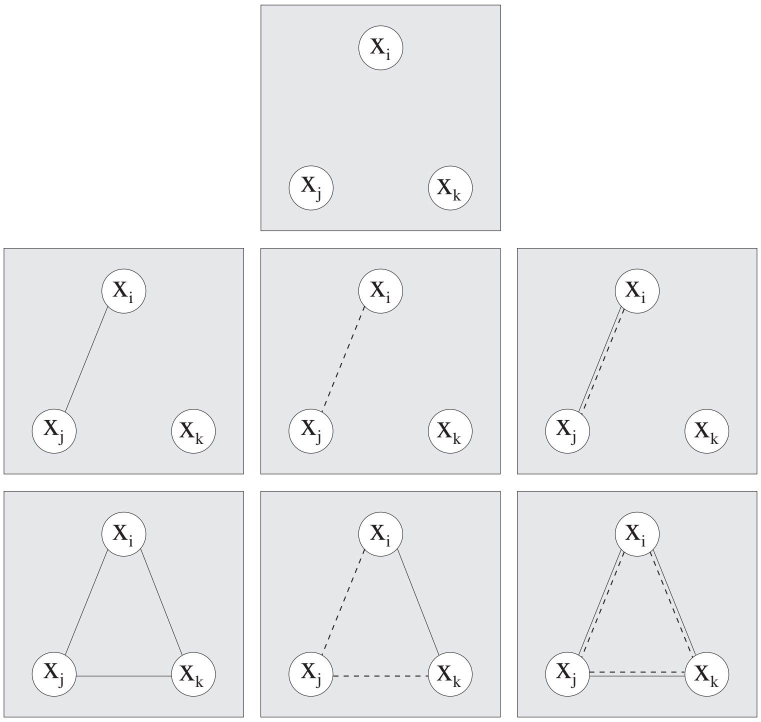


Abbildung 2.2: Alle möglichen Teilgraphen von G_{esym} mit 3 Knoten. Durchgezogene Kanten stehen für „vertauschungssymmetrisch“, gestrichelte Kanten für „äquivalenzsymmetrisch“ und „Doppelkanten“ (sowohl durchgezogen als auch gestrichelte) für „mehrfachsymmetrisch“.

Will man einen Schaltkreis zu einer partiellen Booleschen Funktion finden, so realisiert man eine totale Erweiterung der Funktion. Aufgrund der Tatsache, daß verschiedene totale Erweiterungen Boolescher Funktionen sich stark in ihrer Komplexität unterscheiden können, ist es von großer Wichtigkeit, sich für eine geeignete totale Erweiterung zu entscheiden. Da symmetrische Funktionen im allgemeinen sehr viel leichter zu realisieren sind als allgemeine Funktionen (beispielsweise benötigt man für eine totalsymmetrische Funktion mit n Eingängen im worst case höchstens linear viele 2–Input–Gatter [Weg89], allgemein jedoch $\theta(\frac{2^n}{n})$ [Sha49]), wird hier versucht, solche totale Erweiterungen zu finden, die möglichst viele Symmetrieeigenschaften besitzen. Dabei werden Vertauschungssymmetrien und Äquivalenzsymmetrien betrachtet.

Zur Definition von Symmetrien partieller Funktionen zieht man sich auf die Symmetrie totaler Erweiterungen zurück.

Definition 2.7 (Symmetrie partieller Boolescher Funktionen) *Eine partielle Boolesche Funktion $f \in S(D)$ ($D \subseteq \{0,1\}^n$) heißt (vertauschungs)symmetrisch (äquivalenzsymmetrisch) in den Variablen (x_i, x_k) genau dann, wenn es eine totale Erweiterung f'*

von f gibt, die (vertauschungs)symmetrisch (äquivalenzsymmetrisch) in (x_i, x_k) ist. Analog heißt f symmetrisch in einer Teilmenge λ der Variablenmenge $X = \{x_1, \ldots, x_n\}$, falls es eine totale Erweiterung gibt, die symmetrisch in λ ist, und f heißt symmetrisch in einer Partition P von X, falls es eine totale Erweiterung gibt, die symmetrisch in P ist.

2.2.1 Vertauschungssymmetrie partieller Boolescher Funktionen

Zunächst soll lediglich Vertauschungssymmetrie partieller Boolescher Funktionen betrachtet werden. Im nächsten Abschnitt werden dann die Betrachtungen auf Äquivalenzsymmetrien ausgedehnt.

Zu Beginn wird anhand von Beispielen demonstriert, welche zusätzlichen Schwierigkeiten bei der Bestimmung von Symmetrien *partieller* Boolescher Funktionen auftreten.

Ähnlich wie bei totalen Funktionen ist der Test einer Funktion $f \in S(D)$ auf (Vertauschungs)symmetrie in zwei Variablen (x_i, x_k) relativ einfach: Es darf kein Paar

$$e_1 = (\epsilon_1, \ldots, \epsilon_{i-1}, 0, \epsilon_{i+1}, \ldots, \epsilon_{k-1}, 1, \epsilon_{k+1}, \ldots, \epsilon_n)$$

und

$$e_2 = (\epsilon_1, \ldots, \epsilon_{i-1}, 1, \epsilon_{i+1}, \ldots, \epsilon_{k-1}, 0, \epsilon_{k+1}, \ldots, \epsilon_n)$$

in D geben, so daß $f(e_1) = 0$ und $f(e_2) = 1$ oder umgekehrt. Alternativ kann man diese Bedingung unter Verwendung der entsprechenden Kofaktoren von f_{on} und f_{off} formulieren. Es muß gelten:

$$(f_{on})_{x_i\overline{x_k}} \cdot (f_{off})_{\overline{x_i}x_k} = (f_{on})_{\overline{x_i}x_k} \cdot (f_{off})_{x_i\overline{x_k}} = 0.$$

Ist dies der Fall, so kann man die don't care–Stellen so belegen, daß für die resultierende totale Erweiterung f' gilt: ${f'}_{x_i\overline{x_k}} = {f'}_{\overline{x_i}x_k}$.

Allerdings läßt sich aus den Symmetrien in Variablenpaaren nicht so einfach wie bei totalen Funktionen eine minimale Partition bestimmen, in der die Funktion symmetrisch ist. Der Grund dafür liegt in der Tatsache, daß die Symmetrie in Variablenpaaren aus X für partielle Boolesche Funktionen *keine* Äquivalenzrelation auf X darstellt!
Dies wird anhand des folgenden Beispiels deutlich:

Beispiel 2.2

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	*dc*
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

f ist definiert auf $\{0,1\}^3 \setminus \{(0,1,0)\}$. Offensichtlich ist f symmetrisch in (x_1, x_2) (für die totale Erweiterung f', die symmetrisch ist in (x_1, x_2) muß $f'(0,1,0) = 1$ gelten) und f ist symmetrisch in (x_2, x_3) (für die totale Erweiterung f', die symmetrisch ist in (x_2, x_3) muß $f'(0,1,0) = 0$ gelten). Aber f ist *nicht* symmetrisch in (x_1, x_3) (wegen $f(0,0,1) \neq f(1,0,0)$). Es gibt also auch keine totale Erweiterung, die symmetrisch ist in $\{x_1, x_2, x_3\}$.

Bei partiellen Funktionen wird es also wesentlich schwieriger werden, aus Symmetrien in Variablenpaaren auf Symmetrien in größeren Variablenmengen zu schließen.

Aufgrund der Tatsache, daß die Symmetrie in Variablenpaaren bei totalen Funktionen eine Äquivalenzrelation darstellt, ergibt sich eine spezielle Struktur der zugehörigen Symmetriegraphen: Die Zusammenhangskomponenten stellen Cliquen des Graphen dar. Obwohl bei partiellen Funktionen die Symmetrie in Variablenpaaren keine Äquivalenzrelation darstellt, wäre es trotzdem möglich, daß auch hier die Symmetriegraphen spezielle Struktureigenschaften haben, die man bei der Entwicklung eines Algorithmus zur Bestimmung einer möglichst kleinen Partition, in der die Funktion symmetrisch ist, ausnutzen kann. Dies ist aber nicht der Fall. Aus dem Beweis von Satz 2.2 (Seite 52) wird hervorgehen, daß man zu jedem Graph G mit n Knoten eine partielle Boolesche Funktion $f : D \to \{0,1\}$ $(D \subseteq \{0,1\}^n)$ konstruieren kann, so daß der Symmetriegraph mit G übereinstimmt. Es können also *alle* Graphen als Symmetriegraphen partieller Boolescher Funktionen auftreten.

Auch wenn f symmetrisch ist in *allen* Variablenpaaren (x_i, x_j) aus einer Teilmenge λ der Variablenmenge von f, dann ist f nicht notwendigerweise symmetrisch in λ. Dies wird durch das folgende Beispiel illustriert:

Beispiel 2.3

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	*dc*
0	1	1	0	*dc*
0	1	1	1	0
1	0	0	0	0
1	0	0	1	*dc*
1	0	1	0	*dc*
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

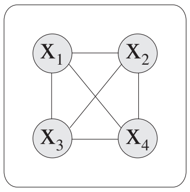


Abbildung 2.3: Symmetriegraph zur Beispielfunktion aus Beispiel 2.3

Man sieht leicht, daß f symmetrisch ist in allen Variablenpaaren (x_i, x_j), $i, j \in \{1, 2, 3, 4\}$. Der Symmetriegraph von f ist in Abbildung 2.3 angegeben. Es handelt sich um den vollständigen Graph. Für jede Erweiterung f' von f, die symmetrisch ist in (x_1, x_3), gilt $f'(0, 1, 1, 0) = 0$ und für jede Erweiterung f'', die symmetrisch ist in (x_2, x_4), gilt $f''(0, 1, 1, 0) = 1$. Folglich gibt es keine Erweiterung von f, die sowohl in (x_1, x_3) als auch in (x_2, x_4) symmetrisch ist und damit auch keine Erweiterung, die in $\{x_1, \ldots, x_4\}$ symmetrisch ist.

Anhand von Beispiel 2.3 wird auch folgende Tatsache deutlich: Wenn f symmetrisch ist in einer Variablenmenge λ_1 und symmetrisch in einer Variablenmenge λ_2, so gibt es nicht notwendigerweise eine Erweiterung von f, die *gleichzeitig* symmetrisch ist in λ_1 *und* in λ_2.

2.2.1.1 Starke Symmetrie

Die Probleme bei der Bestimmung von größeren Symmetriemengen partieller Funktionen ergeben sich aus der Tatsache, daß Symmetrie in Variablenpaaren für partielle Funktionen keine Äquivalenzrelation ist. Ändert man die Definition der Symmetrie partieller Funktionen jedoch gemäß Definition 2.8 ab, so wird die Symmetrie in Variablenpaaren wieder zu einer Äquivalenzrelation.

Definition 2.8 (Starke Symmetrie) *Eine partielle Boolesche Funktion $f \in S(D)$, $D \subseteq \{0,1\}^n$ heißt* **stark symmetrisch** *in den Variablen (x_i, x_k) genau dann, wenn für alle $(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ gilt:*

- *entweder*

$$(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) \notin D \quad \text{und} \quad (\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n) \notin D$$

- *oder*

$$(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) \in D \quad \text{und} \quad (\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n) \in D$$

  und

$$f(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n).$$

Im Gegensatz zur *starken* Symmetrie partieller Funktionen wird die bisher definierte Symmetrie partieller Funktionen gelegentlich zur Verdeutlichung auch als *schwache* Symmetrie bezeichnet.

Wie man leicht sieht, gilt für die starke Symmetrie in 2 Variablen:

Lemma 2.7 *Starke Symmetrie auf Paaren von Variablen aus der Variablenmenge X einer partiellen Booleschen Funktion stellt eine Äquivalenzrelation auf X dar.*

Da starke Symmetrie wiederum eine Äquivalenzrelation darstellt, gibt es auch eine eindeutige minimale Partition P der Menge X der Eingangsvariablen, so daß f stark symmetrisch ist in P. (Starke Symmetrie in einer Variablenmenge liegt wie bei totalen Funktionen vor, wenn die Funktion stark symmetrisch in allen Variablenpaaren aus dieser Menge ist, starke Symmetrie in einer Partition genau dann, wenn die Funktion stark symmetrisch in allen Mengen dieser Partition ist.)

Ist eine Funktion $f \in S(D)$ (schwach) symmetrisch in (x_i, x_j), so gibt es nach Definition (mindestens) eine totale Erweiterung f' von f, die symmetrisch in (x_i, x_j) ist und daher auch stark symmetrisch ist (bei totalen Funktionen stimmen schwache und starke Symmetrie offensichtlich überein). Die nun folgende Prozedur *make_strongly_symm* bestimmt dagegen die Erweiterung von f, die mit einer minimalen Belegung von don't care–Stellen auskommt, d.h. die unter allen in (x_i, x_j) stark symmetrischen Funktionen diejenige mit der kleinsten Definitionsmenge D' ist. Diese Funktion wird im folgenden auch als die *minimale in (x_i, x_j) stark symmetrische Erweiterung* von f bezeichnet.

Prozedur *make_strongly_symm*

Eingabe: $f \in S(D)$, repräsentiert durch f_{on}, f_{off}, f_{dc}.
f ist (schwach) symmetrisch in (x_i, x_j).

Ausgabe: minimale Erweiterung f' von f (repräsentiert durch f'_{on}, f'_{off}, f'_{dc}), die *stark* symmetrisch in (x_i, x_j) ist.

Algorithmus:

1. $f'_{on} = \overline{x_i}\,\overline{x_j} f_{on_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{on_{x_i x_j}} + (x_i \overline{x_j} + \overline{x_i} x_j)(f_{on_{x_i \overline{x_j}}} + f_{on_{\overline{x_i} x_j}})$
2. $f'_{off} = \overline{x_i}\,\overline{x_j} f_{off_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{off_{x_i x_j}} + (x_i \overline{x_j} + \overline{x_i} x_j)(f_{off_{x_i \overline{x_j}}} + f_{off_{\overline{x_i} x_j}})$
3. $f'_{dc} = \overline{f'_{on} + f'_{off}}$

Die Prozedur *make_strongly_symm* belegt alle don't care–Stellen $\epsilon \notin D$ mit $\sigma_{ij}(\epsilon) \in D$ mit dem Wert $f(\sigma_{ij}(\epsilon))$, ansonsten bleibt die don't care–Stelle erhalten.

Bevor in den nächsten Abschnitten die Komplexität des Problems, zu einer gegebenen partiellen Booleschen Funktion eine möglichst kleine Partition zu bestimmen, in der die Funktion symmetrisch ist, näher untersucht wird und schließlich ein Algorithmus zur Lösung dieses Problems angegeben wird, sollen zunächst schwache und starke Symmetrie in Partitionen der Variablenmenge etwas genauer charakterisiert werden. Dazu wird der Begriff der „Gewichtsklasse“ zu einer gegebenen Partition P benötigt.

Definition 2.9 (Gewichtsklasse zu einer Partition P)
Sei $P = \{\lambda_1, \ldots, \lambda_k\}$ eine Partition von $\{x_1, \ldots, x_n\}$.
Für $(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ heißt $w^1(\epsilon_1, \ldots, \epsilon_n) = \sum_{i=1}^n \epsilon_i$ das 1–Gewicht von $(\epsilon_1, \ldots, \epsilon_n)$, $w^0(\epsilon_1, \ldots, \epsilon_n) = n - w^1(\epsilon_1, \ldots, \epsilon_n)$ das 0–Gewicht von $(\epsilon_1, \ldots, \epsilon_n)$.
Für $\lambda_i = \{x_{i_1}, \ldots, x_{i_l}\}$ ist $w^1_{\lambda_i}(\epsilon_1, \ldots, \epsilon_n) = \sum_{j \in \{i_1, \ldots, i_l\}} \epsilon_j$ das 1–Gewicht des „λ_i–Teils“ von $(\epsilon_1, \ldots, \epsilon_n)$, $w^0_{\lambda_i}(\epsilon_1, \ldots, \epsilon_n) = |\lambda_i| - w^1_{\lambda_i}(\epsilon_1, \ldots, \epsilon_n)$ das 0–Gewicht des „λ_i–Teils“ von $(\epsilon_1, \ldots, \epsilon_n)$.
Dann heißt

$$C^P_{w_1, \ldots, w_k} = \{(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n \mid w^1_{\lambda_i}(\epsilon_1, \ldots, \epsilon_n) = w_i, 1 \leq i \leq k\}$$

Gewichtsklasse *zu der Partition P mit den Gewichten $(w_1, \ldots, w_k)$.*

Beispiel 2.4 Sei $P = \{\{x_1, x_2\}, \{x_3, x_4, x_5\}\}$. Dann ist $C^P_{1,2}$ die Teilmenge aller Vektoren aus $\{0,1\}^n$, deren $\{x_1, x_2\}$–Gewicht 1 ist und deren $\{x_3, x_4, x_5\}$–Gewicht 2 ist, d.h. die Teilmenge aller Vektoren, bei denen auf den ersten beiden Komponenten genau eine 1 vorkommt und auf den restlichen Komponenten genau 2 Einsen vorkommen.
$C^P_{1,2} = \{(0,1,0,1,1), (0,1,1,0,1), (0,1,1,1,0), (1,0,0,1,1), (1,0,1,0,1), (1,0,1,1,0)\}$.

Bemerkung 2.2 *Ist $P = \{\lambda_1, \ldots, \lambda_k\}$ eine Partition von $\{x_1, \ldots, x_n\}$, so sind die Gewichtsklassen zu P gerade die* Orbits*, die durch $\{\sigma_{il} \mid \exists \lambda_j \in P \text{ mit } x_i, x_l \in \lambda_j\}$ auf $\{0,1\}^n$ erzeugt werden.*

Mit Hilfe dieser „Gewichtsklassen“ lassen sich starke und schwache Symmetrie in Partitionen leicht charakterisieren:

Lemma 2.8 *Ist $P = \{\lambda_1, \ldots, \lambda_k\}$ Partition von $\{x_1, \ldots, x_n\}$, dann ist $f : D \to \{0,1\}$ ($D \subseteq \{0,1\}^n$)*

1. stark symmetrisch *in P genau dann, wenn*

$$\forall 0 \leq w_i \leq |\lambda_i| \ (1 \leq i \leq k) \quad f(C^P_{w_1, \ldots, w_k}) = \begin{cases} \{0\} & \text{oder} \\ \{1\} & \text{oder} \\ \{dc\} & \end{cases}$$

2. (schwach) symmetrisch *in P genau dann, wenn*

$$\forall 0 \leq w_i \leq |\lambda_i| \ (1 \leq i \leq k) \quad \{0,1\} \not\subseteq f(C^P_{w_1, \ldots, w_k})$$

Beweis:

zu 1. „$\Longleftarrow$“: Angenommen f ist nicht stark symmetrisch in P. Dann muß es ein $\lambda_i \in P$ geben, so daß f nicht stark symmetrisch in λ_i und ein Variablenpaar $(x_i, x_j) \in \lambda_i$, so daß f nicht stark symmetrisch in (x_i, x_j). Dann muß es nach Definition $e_1 = (\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n)$ und $e_2 = (\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$ geben, so daß $e_1 \in D$ und $e_2 \notin D$ oder $e_1, e_2 \in D$ und $f(e_1) \neq f(e_2)$. Allerdings sind e_1 und e_2 in derselben Gewichtsklasse C zu P. In beiden Fällen ergäbe sich ein Widerspruch: Im ersten Fall wäre $\{dc, 1\}$ oder $\{dc, 0\} \subseteq f(C)$, im zweiten Fall wäre $\{0, 1\} \subseteq f(C)$.

„$\Longrightarrow$“: Wenn f stark symmetrisch ist in P, dann gilt für alle $\sigma \in \Sigma = \{\sigma_{il} \mid \exists \lambda_j \in P \text{ mit } x_i, x_l \in \lambda_j\}$: $\forall e = (\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n \; f(e) = f(\sigma(e))$ (einschließlich der erweiterten Interpretation bei $f(e) = f(\sigma(e)) = dc$). Seien e_1 und e_2 aus einer beliebigen Gewichtsklasse C zu P. Dann gibt es eine Folge von Permutationen $\sigma_1 \ldots \sigma_l \in \Sigma$ mit $e_2 = (\sigma_1 \circ \ldots \circ \sigma_l)(e_1)$. Folglich muß $f(e_1) = f(e_2)$ gelten, so daß $f(C) = \{0\}$ oder $f(C) = \{1\}$ oder $f(C) = \{dc\}$.

zu 2. „$\Longleftarrow$“: Es gelte $\forall 0 \leq w_i \leq |\lambda_i| (1 \leq i \leq k) \; \{0,1\} \not\subseteq f(C^P_{w_1,\ldots,w_k})$.
Z.z.: Es gibt eine totale Erweiterung f' von f, die symmetrisch in P ist.
Definiere f' wie folgt:
Falls für eine Gewichtsklasse C $f(C) = \{\epsilon\}$ ($\epsilon \in \{0,1\}$), dann $f'(C) = f(C)$.
Falls für eine Gewichtsklasse C $f(C) = \{dc\}$, dann $f'(C) = 0$.
Falls für eine Gewichtsklasse C $f(C) = \{\epsilon, dc\}$ ($\epsilon \in \{0,1\}$), dann $f'(C) = \epsilon$.
Dann ist f' eine totale Funktion und nach Teil 1. des Satzes stark symmetrisch in P, also auch symmetrisch in P gemäß der Definition von Symmetrie für totale Funktionen.

„$\Longrightarrow$“: Sei f (schwach) symmetrisch in P. Dann gibt es eine totale Erweiterung f' von f, die symmetrisch in P ist. Gäbe es eine Gewichtsklasse C mit $\{0,1\} \subseteq f(C)$, so müßte auch $\{0,1\} \subseteq f'(C)$ sein, da f' eine Erweiterung von f ist. Da f' total ist, gilt nach Teil 1. des Satzes aber für alle Gewichtsklassen C zu P: $f'(C) = \{0\}$ oder $f'(C) = \{1\}$. Es kann also keine solche Gewichtsklasse geben.

$\square$

2.2.1.2 Das Problem MSP

Nach diesen Vorüberlegungen soll nun das Problem untersucht werden, zu einer gegebenen partiellen Booleschen Funktion eine möglichst kleine Partition zu bestimmen, in der die Funktion symmetrisch ist. Das Problem ist folgendermaßen definiert:

Problem MSP (Minimal Symmetry Partition)

Gegeben: Partielle Funktion $f \in S(D)$, repräsentiert durch die beiden ROBDDs f_{on} und f_{dc}.

Gesucht: Partition P der Menge $X = \{x_1, \ldots, x_n\}$ der Eingangsvariablen von f, so daß f symmetrisch ist in P und für alle Partitionen P' von X, in denen f symmetrisch ist, gilt: $|P| \leq |P'|$.

Meines Wissens nach gibt es keine Arbeiten in der Literatur, die sich mit einer systematischen Lösung des genannten Problems befassen. Es sind lediglich Arbeiten bekannt, die bei partiellen Funktionen für Variablenpaare oder Teilmengen der Menge aller Eingangsvariablen testen können, ob eine Funktion symmetrisch ist in dem angegebenen Variablenpaar oder in der angegebenen Variablenmenge (z.B. [DS67, KD91]). Die Tatsache, daß (schwache) Symmetrie für partielle Funktionen keine Äquivalenzrelation ist, führt aber dazu, daß man nicht in einfacher Weise aus kleinen „Symmetriemengen" größere zusammenbauen kann. Da es exponentiell viele Teilmengen der Menge der Eingangsvariablen gibt, muß man gegebenfalls für sehr viele solche Mengen auf Symmetrie testen, wenn man lediglich in der Lage ist, zu einer vorgegebenen Variablenmenge zu testen, ob die Funktion in dieser Variablenmenge symmetrisch ist. Ist man darüber hinaus nicht nur an einzelnen Symmetriemengen interessiert, sondern an Partitionen, in denen die Funktion symmetrisch ist, tritt zusätzlich noch das Problem auf, daß man aus der Tatsache, daß eine Funktion symmetrisch ist in 2 disjunkten Variablenmengen λ_1 und λ_2, *nicht* schließen kann, daß sie „zugleich" in λ_1 *und* λ_2 symmetrisch ist.

Die Hoffnung, leicht einen polynomiellen Algorithmus finden zu können, der MSP *exakt* löst, wird durch folgenden Satz zunichte gemacht:

Satz 2.2 *Das Problem* **MSP** *ist NP–hart.*

Satz 2.2 wird bewiesen durch Angabe einer Polynomzeittransformation vom *NP*–vollständigen Problem „Partition into Cliques" (PC) nach MSP.

Das Problem PC lautet (siehe [GJ79], S. 193):

Partition into Cliques (PC)

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $K \leq |V|$.

Gesucht: Kann V partitioniert werden in K disjunkte Mengen $V_1, \ldots, V_K$, so daß für $1 \leq i \leq K$ der Teilgraph, der alle Knoten aus V_i umfaßt, ein vollständiger Graph ist?

Es folgt der Beweis zu Satz 2.2:

Beweis: Gegeben sei eine Instanz des Problems PC, also ein Graph $G = (V, E)$ und eine natürliche Zahl $K \leq |V|$. Es werden nun ROBDDs f_{on} und f_{dc} einer partiellen Booleschen

Funktion f in Polynomzeit aus G konstruiert mit der Eigenschaft, daß es genau dann eine Partition von G in K Cliquen gibt, wenn es eine Partition P der Variablenmenge X von f gibt mit $|P| = K$, so daß f symmetrisch ist in P.

Sei o.B.d.A. $V = \{x_1, \ldots, x_n\} = X$. Die Funktion $f \in S(D)(D \subseteq \{0,1\}^n)$ wird definiert durch

$$f_G(\epsilon_1, \ldots, \epsilon_n) = \begin{cases} 1 & \text{falls} \quad \epsilon_1 = \ldots = \epsilon_i = 1,\ \epsilon_{i+1} = \ldots = \epsilon_n = 0,\ 1 \leq i \leq n-1 \\ 0 & \text{falls} \quad \epsilon_1 = \ldots = \epsilon_{i-1} = 1,\ \epsilon_i = \ldots = \epsilon_{j-1} = 0,\ \epsilon_j = 1, \\ & \qquad\ \ \epsilon_{j+1} = \ldots = \epsilon_n = 0, \\ & \qquad\ \ 1 \leq i \leq n-1, j > i \text{ und } \{x_i, x_j\} \notin E \\ dc & \text{sonst} \end{cases}$$

Der Symmetriegraph G_{sym} zu f_G ist nun identisch mit G:

- Die Kantenmenge von G_{sym} ist auf jeden Fall eine Teilmenge der Kantenmenge von G. Es gibt in G_{sym} keine Kante zwischen zwei Knoten x_i und x_j, wenn es in G keine Kante zwischen x_i und x_j gibt. f_G ist gerade so konstruiert, daß die Belegung für die Elemente $(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ mit dem Gewicht $w^1(\epsilon_1, \ldots, \epsilon_n) = i$ gewährleistet, daß es keine Kanten in G_{sym} zwischen x_i und irgendeiner Variablen x_j $(j > i)$ gibt, falls $\{x_i, x_j\} \notin E$. Denn in diesem Fall ist

$$f_G(\underbrace{1, \ldots, 1}_{i \text{ mal}}, 0 \ldots 0) = 1 \text{ und } f_G(\underbrace{1, \ldots, 1}_{i-1 \text{ mal}}, 0, \underbrace{0, \ldots, 0}_{j-i-1 \text{ mal}}, 1, 0, \ldots, 0) = 0,$$

  also f_G nicht symmetrisch in (x_i, x_j).

- Falls es eine Kante $\{x_i, x_j\}$ in G gibt, dann gibt es auch eine Kante $\{x_i, x_j\}$ in G_{sym}, d.h. dann ist f_G symmetrisch in (x_i, x_j). Angenommen, $\{x_i, x_j\} \in G$ und f_G nicht symmetrisch in (x_i, x_j). (O.B.d.A.: $i < j$.) Dann muß es $\epsilon = (\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ geben, so daß $f_G(\epsilon_1, \ldots, \epsilon_{i-1}, \epsilon_i, \epsilon_{i+1}, \ldots, \epsilon_{j-1}, \epsilon_j, \epsilon_{j+1}, \ldots, \epsilon_n) = 1$ und
  $f_G(\epsilon_1, \ldots, \epsilon_{i-1}, \epsilon_j, \epsilon_{i+1}, \ldots, \epsilon_{j-1}, \epsilon_i, \epsilon_{j+1}, \ldots, \epsilon_n) = 0$.
  Nach Definition von f_G haben die 1–Stellen alle die Form $(1, \ldots, 1, 0, \ldots, 0)$. Außerdem muß $\epsilon_i \neq \epsilon_j$ sein. Also gilt $\epsilon_1 = \ldots = \epsilon_k = 1$, $\epsilon_{k+1} = \ldots = \epsilon_n = 0$ für $i \leq k < n$. Die Nullstellen von f_G haben alle die Form $(1, \ldots, 1, 0, 0, \ldots, 0, 1, 0, \ldots, 0)$. Folglich kann nur $k = i$ gelten. Falls $\sigma_{ij}(\epsilon)$ wirklich eine Nullstelle von f_G ist, dann muß nach Definition von f_G aber $\{x_i, x_j\} \notin E$ sein, im Widerspruch zur Annahme.

Die beiden Graphen G und G_{sym} sind aber nicht nur identisch, es gilt auch folgende Hilfsbehauptung:

Ist $P = \{\lambda_1, \ldots, \lambda_k\}$ eine Partition von V, so daß die Teilgraphen mit allen Knoten aus λ_i jeweils vollständige Teilgraphen von G sind (d.h. Cliquen in G), dann ist f_G symmetrisch in der Partition P, d.h. „f_G ist symmetrisch in jeder Partition des Symmetriegraphen in Cliquen“.

Beweis der Hilfsbehauptung:
Annahme: Es gibt eine Partition Q des Symmetriegraphen in Cliquen, so daß f_G nicht symmetrisch ist in Q.
Dann gibt es nach Lemma 2.8 eine Gewichtsklasse $C^Q_{w_1,\ldots,w_k}$ zu Q mit $\{0,1\} \subseteq f_G(C^Q_{w_1,\ldots,w_k})$. Es wird nun analog zur obigen Argumentation ein Widerspruch hergeleitet aus der speziellen Struktur der 0– und 1–Stellen von f_G.
Ist $w = \sum_{i=1}^k w_i$, so ist $\epsilon^{(1)} = (\underbrace{1,\ldots,1}_{w \text{ mal}}, 0,\ldots,0)$ die einzige 1–Stelle von f_G in $C^Q_{w_1,\ldots,w_k}$.
Nach Annahme muß es eine 0–Stelle in $C^Q_{w_1,\ldots,w_k}$ geben. Diese hat nach Definition von f_G die Form $\epsilon^{(0)} = (\underbrace{1,\ldots,1}_{w-1 \text{ mal}}, 0,0,\ldots,0, \underbrace{1}_{\epsilon_j^{(0)}}, 0,\ldots,0)$, da $\epsilon^{(1)}$ und $\epsilon^{(0)}$ als Elemente der gleichen Gewichtsklasse gleiches 1–Gewicht haben müssen.
Nach Definition von f_G gilt $(x_w, x_j) \notin E$ und f_G nicht symmetrisch in (x_w, x_j). (*)
Sei λ' die Menge aus Q mit $x_w \in \lambda'$. Wäre $x_j \notin \lambda'$, dann wäre $w^1_{\lambda'}(\epsilon^{(0)}) = w^1_{\lambda'}(\epsilon^{(1)}) - 1$. Da aber $\epsilon^{(0)}$ und $\epsilon^{(1)}$ in derselben Gewichtsklasse sind, kann dies nicht der Fall sein und $x_j \in \lambda'$. Dies ist aber wegen (*) ein Widerspruch zur Annahme, daß die Elemente von λ' eine Clique im Symmetriegraph bilden. Damit ist die Hilfsbehauptung bewiesen.

Umgekehrt gilt natürlich (wie für alle partielle Funktionen) auch für f_G folgende Aussage: Falls f_G symmetrisch ist in einer Partition $P = \{\lambda_1, \ldots, \lambda_k\}$, dann liefert P eine Überdeckung des Symmetriegraphen von f_G durch Cliquen. (Es gibt dann nämlich eine totale Erweiterung ${f_G}'$ von f_G, die symmetrisch ist in P. λ_i $(1 \leq i \leq k)$ bilden Cliquen im Symmetriegraph von ${f_G}'$. Da der Symmetriegraph von ${f_G}'$ höchstens durch Streichen von Kanten aus dem Symmetriegraph von f_G hervorgeht, bilden λ_i auch Cliquen im Symmetriegraph von f_G.)

Es gibt also genau dann eine Überdeckung von G mit k Cliquen, wenn die zu G konstruierte partielle Funktion f_G symmetrisch ist in einer Partition von X mit k Mengen.

Es bleibt noch zu zeigen, daß die Darstellung der partiellen Funktion f_G durch die ROBDDs $f_{G_{on}}$ und $f_{G_{dc}}$ aus G in Polynomzeit berechnet werden kann. Dabei wird ausgenutzt, daß die ON–Menge von $f_{G_{on}}$ und die OFF–Menge von $f_{G_{dc}}$ polynomielle Größe haben: Es gilt $|ON(f_{G_{on}})| = n - 1$ und $|OFF(f_{G_{dc}})| \leq (n-1) + \frac{1}{2}n(n-1)$.

Allgemein gilt: Falls die ON–Menge $ON(f)$ (oder die OFF–Menge $OFF(f)$) einer Funktion $f \in B_n$ die Größe N hat, dann kann man einen ROBDD zu f (unabhängig von der Variablenordnung) in Zeit $O(nN \log(nN))$ konstruieren.

Die folgende rekursive Prozedur liefert bei Aufruf *build_obdd*$(ON(f), \{x_1, \ldots, x_n\})$ (einen (evtl. noch nicht reduzierten) OBDD für f:

Prozedur *build_obdd*

Eingabe: ON–Menge $ON(f)$ von $f \in B_n$, Variablenmenge $\{x_1, \ldots, x_n\}$

Ausgabe: OBDD zu f

Algorithmus:

1. Falls die Variablenanzahl $n = 1$ und
   - $ON(f) = \emptyset$, dann liefere den ROBDD $\boxed{0}$ zurück.
   - $ON(f) = \{0, 1\}$, dann liefere den ROBDD $\boxed{1}$ zurück.
   - $ON(f) = \{1\}$, dann liefere den ROBDD für die Variable x_n zurück.
   - $ON(f) = \{0\}$, dann liefere den ROBDD für die Negation der Variablen x_n zurück.

   Sonst:
2. Bestimme
   $ON(f^0) = \{(\epsilon_2, \ldots, \epsilon_n) \mid (0, \epsilon_2, \ldots, \epsilon_n) \in ON(f)\}$ und
   $ON(f^1) = \{(\epsilon_2, \ldots, \epsilon_n) \mid (1, \epsilon_2, \ldots, \epsilon_n) \in ON(f)\}$.
3. Falls $ON(f^0) \neq \emptyset$,
   dann $G_{low} =$ *build_obdd*$(ON(f^0), \{x_2, \ldots, x_n\})$,
   sonst $G_{low} = \boxed{0}$.
4. Falls $ON(f^1) \neq \emptyset$,
   dann $G_{high} =$ *build_obdd*$(ON(f^1), \{x_2, \ldots, x_n\})$,
   sonst $G_{high} = \boxed{0}$.
5. Sei v ein neuer Knoten mit Beschriftung x_1 und der Wurzel von G_{low} als 0–Sohn und der Wurzel von G_{high} als 1–Sohn. Liefere den OBDD mit Wurzel v zurück.

Falls man die Vektoren aus $ON(f)$ durch lineare Listen darstellt und im Rumpf der Prozedur in Zeile 2 beim Aufteilen lediglich die ersten Komponenten dieser Listen ansieht und dann löscht, benötigt man für sämtliche rekursiven Aufrufe einer Tiefe i insgesamt Zeit $O(N)$. Da die Rekursion spätestens bei Tiefe $n-1$ abbricht, wenn nur noch 1 Variable vorhanden ist (Zeile 1), benötigt man insgesamt Zeit $O(n \cdot N)$.

Eine analoge Prozedur kann man für vorgegebenes $OFF(f)$ (statt $ON(f)$) angeben.

Der resultierende OBDD kann dann in Zeit $O(n \cdot N \cdot \log(nN))$ zu einem ROBDD reduziert werden[1] (vgl. Kapitel 1). Insgesamt wird der ROBDD also in $O(n \cdot N \cdot \log(nN))$ aufgebaut.

Somit ist gezeigt, daß die ROBDDs zu $f_{G_{on}}$ und $f_{G_{dc}}$ in polynomieller Zeit aufgebaut werden können. □

Bemerkung 2.3 *Die Tatsache, daß f_G im Beweis von Satz 2.2 in jeder Partition des Symmetriegraphen in Cliquen symmetrisch ist, ist eine spezielle Eigenschaft von f_G. Wie u.a. in Beispiel 2.3 deutlich wurde, ist dies im allgemeinen* nicht *der Fall.*

[1] In den gängigen Implementierungen für ROBDDs (z.B. [BRB90]) wird allerdings nicht mit einem expliziten Reduzieren nicht–reduzierter OBDDs gearbeitet. Mit Hilfe einer Hash-Tabelle („unique table“) wird schon gleich beim Aufbau der ROBDDs dafür gesorgt, daß nie isomorphe Graphen im System vorkommen können.

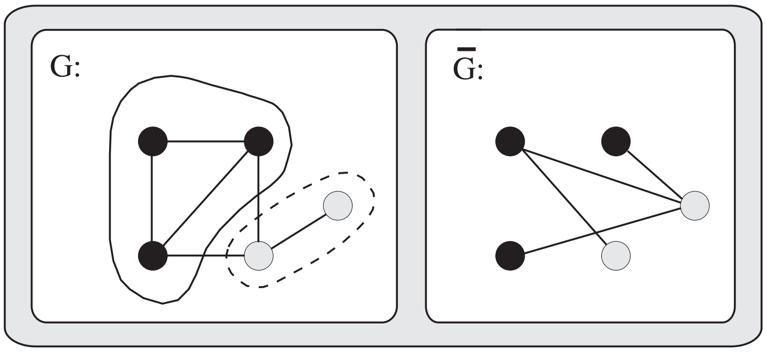


Abbildung 2.4: Graph $\bar{G}$ kann mit 2 Farben gefärbt werden (schwarze und graue Knoten). G kann daher in 2 Cliquen aufgeteilt werden.

2.2.1.3 Ein Algorithmus für MSP

Der Beweis von Satz 2.2 legt nahe, zur (näherungsweisen) Lösung des Problems MSP eine Heuristik für das Problem „Partition into Cliques“ zu verwenden. Allerdings muß eine solche Heuristik so abgeändert werden, daß gewährleistet werden kann, daß die berechnete Partitionierung des Symmetriegraphen in Cliquen auch wirklich zu einer Partition der Eingangsvariablen führt, in der die gegebene partielle Funktion symmetrisch ist.

Will man das Problem PC lösen, so kann man von der folgenden bekannten Äquivalenz Gebrauch machen:

$G = (V, E)$ kann in k disjunkte Cliquen partitioniert werden.

$$\Longleftrightarrow$$

$\bar{G} = (V, \bar{E})$ kann mit k Farben gefärbt werden.

Hierbei ist mit $\bar{G}$ der zu G inverse Graph gemeint, der die gleiche Knotenmenge wie G hat, aber genau dann eine Kante zwischen zwei Knoten aufweist, wenn es in G keine Kante zwischen den beiden Knoten gibt, d.h. $\bar{E} = \{\{v, w\} \mid v, w \in V, \{v, w\} \notin E\}$.
Ist $\{V_1, \ldots, V_k\}$ eine Partition von V in k disjunkte Cliquen, so kann man in $\bar{G}$ die Knoten aus V_i gleich färben, da diese in $\bar{G}$ ein „independent set “ bilden. Umgekehrt bilden gleich gefärbte Knoten in $\bar{G}$ ein independent set und damit eine Clique in G (siehe Abbildung 2.4).

Heuristiken zur Färbung eines Graphen können also direkt zur Lösung von „Partition into Cliques“ eingesetzt werden. Ein solcher Algorithmus ist der Algorithmus von Brélaz [Bre79], der in einer Implementierung von Morgenstern [Mor92] eine Laufzeit von $O(n)$ aufweist (n ist die Anzahl der Knoten in dem zu färbenden Graphen). Es handelt sich

Eingabe: Partielle Funktion $f \in S(D)$, $D \subseteq \{0,1\}^n$, repräsentiert durch f_{on} und f_{dc}

Ausgabe: Partition P von $\{x_1, \ldots, x_n\}$, so daß f symmetrisch ist in P

Algorithmus:

```
1   Berechne Symmetriegraph G_sym = (V, E) zu f (oder \overline{G_sym} = (V, Ē)).
2   ∀1 ≤ k ≤ n : color(x_k) := undef.
3   P = {{x_1}, {x_2}, ..., {x_n}}
4   node_candidate_set := {x_1, ..., x_n}
5   while (node_candidate_set ≠ ∅) do
6       /* f ist stark symmetrisch in P */
7       Wähle x_i ∈ node_candidate_set gemäß Brélaz/Morgenstern–Kriterium
8       color_candidate_set := {c | 1 ≤ c ≤ n, ∄x_j mit {x_i, x_j} ∈ Ē und color(x_j) = c}
9       while (color(x_i) = undef.) do
10          curr_color := min(color_candidate_set)
11          color(x_i) := curr_color
12          if (∃ gefärbter Knoten x_j mit color(x_j) = color(x_i))
13            then
14              if (f symmetrisch in (x_i, x_j))
15                then
16                  P := P \ {[x_j], {x_i}} ⋃ {[x_j] ∪ {x_i}}
17                  /* f ist symmetrisch in P */                       (*)
18                  Mache f stark symmetrisch in P.                    (**)
19                else
20                  color_candidate_set := color_candidate_set \ {curr_color}
21                  color(x_i) := undef.
22              fi
23          fi
24      od
25      node_candidate_set := node_candidate_set \ {x_i}
26  od
```

Abbildung 2.5: Algorithmus zur Lösung von MSP.

hierbei um einen Greedy-Algorithmus, der Knoten für Knoten färbt und (zumindest in der nicht–iterativen Ursprungsversion des Algorithmus) die Farbe einmal gefärbter Knoten nicht wieder verändert. Brélaz/Morgenstern benutzen bestimmte Kriterien [Bre79, Mor92] für eine geschickte Auswahl des nächsten zu färbenden Knotens und der dafür zu wählenden Farbe.

In Abbildung 2.5 ist eine Heuristik für MSP angegeben, die von der Brélaz/Morgenstern–Heuristik zum Knotenfärben abgeleitet ist.

Zunächst wird der Symmetriegraph G_{sym} zu f bzw. der dazu inverse Graph $\overline{G_{sym}}$ bestimmt (Zeile 1). Die Knoten des Graphen $\overline{G_{sym}}$ sind die Variablen $x_1, \ldots, x_n$. Diese Knoten werden im Verlauf des Algorithmus gefärbt. Gleich gefärbte Knoten bilden eine Clique in G_{sym}. Zunächst sind alle Knoten ungefärbt (Zeile 2). Die Partition P (vgl. Zeile 3) hat immer die Eigenschaft, daß sich ungefärbte Knoten x_k in einer eigenen Menge $\{x_k\}$ befinden und gleich gefärbte Knoten sich in der gleichen Menge der Partition befinden. Im Verlauf des Algorithmus wird immer die Invariante aus Zeile 6 („f ist stark symmetrisch in P") aufrechterhalten. Es ist klar, daß f diese Invariante beim ersten Durchlauf erfüllt, da zu diesem Zeitpunkt sich alle Variablen in einer eigenen Menge von P befinden. Die Knoten werden nacheinander gefärbt, wobei sich der nächste zu färbende Knoten aus dem Kriterium von Brélaz/Morgenstern ([Bre79, Mor92]) ergibt.[2] Die Menge der möglichen Farben für den nächsten zu färbenden Knoten x_i ist die Menge aller „Farben" zwischen 1 und n außer den Farben von Knoten, die zu x_i benachbart sind (Zeile 8). Im Originalalgorithmus von Brélaz/Morgenstern wird nun als Farbe für x_i unter diesen Farbe die minimale Farbe ausgewählt. (Der dieser Heuristik zugrundeliegende Gedanke beruht darauf, daß man die Farben „von unten her auffüllen" will, um bei einem aktuell zu färbenden Knoten den Erwartungswert für die Anzahl der Farben in der Nachbarschaft, d.h. den Erwartungswert für die Anzahl der „Konfliktfälle" beim Färben so gering wie möglich zu halten und so mit einer möglichst geringen Zahl von Farben auszukommen.) An dieser Stelle ist nun zu beachten, daß wir gewährleisten müssen, daß f in der sich durch die Färbung ergebenden Partition P symmetrisch ist. Falls also *curr_color* der aktuelle Kandidat für die Farbe von x_i ist (Zeilen 10 und 11) und es schon einen Knoten x_j gibt, der mit *curr_color* gefärbt ist, dann muß f symmetrisch sein in der Partition P', die sich ergibt, wenn man in P die Menge $\{x_i\}$ und die Menge $[x_j]$[3], die x_j enthält, vereinigt. Gibt es also einen Knoten x_j, der schon mit der Farbe *curr_color* gefärbt ist, so wird getestet, ob f symmetrisch ist in (x_i, x_j) (Zeile 14). Dies ist im ersten und zweiten Durchlauf der Schleife in Zeilen 5–26 aufgrund der Berechnung von G_{sym} bzw. von $\overline{G_{sym}}$ auf jeden Fall gegeben, muß aber in den nächsten Durchläufen der Schleife nicht mehr wahr sein, da die don't care–Menge von f im Verlauf des Algorithmus verändert wird. Ist f nicht symmetrisch in (x_i, x_j), so wird *curr_color* aus der Menge der „Farbkandidaten" für x_i entfernt (Zeile 20) und als neuer Kandidat wird die minimale Farbe in der nun verbleibenden Menge gewählt (Zeile 10). Die Suche nach einer Farbe für x_i endet entweder mit einer noch nicht verwendeten Farbe, wenn die Bedingung aus Zeile 12 falsch ist oder mit einer schon verwendeten Farbe *curr_color*, für die die Bedingung aus Zeile 14 wahr ist. Ist die Bedingung aus Zeile 14 wahr, so ergibt sich die neue Partition P durch Vereinigung der beiden Mengen $\{x_i\}$ und $[x_j]$ in der alten Partition P (Zeile 16). Es gilt nun, daß f symmetrisch ist in der neuen Partition P (Invariante (*) in Zeile 17) und f kann stark symmetrisch gemacht werden in P (Zeile 18). Daß aufgrund der im Algorithmus gegebenen Bedingungen an dieser Stelle tatsächlich gilt, daß f symmetrisch ist in P, wird durch den Beweis des noch folgenden Lemmas 2.9 gezeigt. Außerdem muß noch gezeigt werden, wie f in P stark symmetrisch gemacht werden kann. Das Wesentliche an der Invariante (*) „f symmetrisch in P" und der Tatsache, daß f stark

[2] Zur Zeit werden in einer laufenden Diplomarbeit [Mel96] auch andere Kriterien zur Knotenauswahl erprobt. Alternativen bestehen z.B. darin, den nächsten zu färbenden Knoten so auszuwählen, daß die Anzahl der neu belegten don't cares in Zeile 18 des Algorithmus minimiert wird, oder so, daß durch Belegen von don't cares in Zeile 18 möglichst wenig Symmetrien in anderen Variablenpaaren zerstört werden.

[3] Ist $P = \{\lambda_1, \ldots, \lambda_k\}$ eine Partition von $\{x_1, \ldots, x_n\}$, dann wird λ_i mit $x_j \in \lambda_i$ mit $[x_j]$ bezeichnet.

symmetrisch gemacht werden kann in P, besteht darin, daß im Verlauf des Algorithmus Symmetrien, die einmal erkannt worden sind, nicht mehr zerstört werden. Sind die noch offenen Punkte (*) und (**) aus Zeilen 17 und 18 geklärt, so sieht man, daß in einem erneuten Durchlauf der **while**–Schleife zum Färben des nächsten Knotens die Invariante „f ist stark symmetrisch in P“ weiterhin wahr ist und daß f nach Ablauf des Algorithmus stark symmetrisch in der gefundenen Partition P ist.

Es ist noch zu zeigen, daß die Invariante (*) aus Zeile 17 des Algorithmus korrekt ist. Dies geht aus dem folgenden Lemma hervor:

Lemma 2.9 *Ist $f \in S(D)$ $(D \subseteq \{0,1\}^n)$* stark symmetrisch *in P, $[x_i], [x_j] \in P$, $||[x_i]|| = 1$ und f symmetrisch in (x_i, x_j), dann ist f symmetrisch in*

$$P' = P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}.$$

Bemerkung 2.4 *Die Aussage des Lemmas ist* nicht *korrekt,*

- *wenn f nur als schwach symmetrisch in P (nicht als* stark *symmetrisch in P) vorausgesetzt wird oder*
- *wenn man nicht $||[x_i]|| = 1$ voraussetzt.*

Die gegebenen Voraussetzungen entsprechen aber genau den im Algorithmus vorhandenen Voraussetzungen.

Beispiel 2.5 Die folgende Funktion f liefert ein Beispiel dafür, daß es in Lemma 2.9 nicht ausreicht anzunehmen, daß f *schwach* symmetrisch in P ist.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	dc
0	0	0	1	dc
0	0	1	0	dc
0	0	1	1	dc
0	1	0	0	dc
0	1	0	1	0
0	1	1	0	dc
0	1	1	1	dc
1	0	0	0	dc
1	0	0	1	dc
1	0	1	0	1
1	0	1	1	dc
1	1	0	0	dc
1	1	0	1	dc
1	1	1	0	dc
1	1	1	1	dc

Man sieht leicht, daß f symmetrisch ist in $\{\{x_1\}, \{x_2, x_3, x_4\}\}$. Die einzigen Gewichtsklassen hinsichtlich dieser Partition, die nicht komplett auf dc abgebildet werden, sind $C_{0,2} = \{(0,0,1,1), (0,1,0,1), (0,1,1,0)\}$ und $C_{1,1} = \{(1,0,0,1), (1,0,1,0), (1,1,0,0)\}$. Im Bild der ersten Klasse unter f kommt aber nur dc und 0 vor, im Bild der zweiten Klasse unter f nur dc und 1. Weiterhin ist f offensichtlich auch symmetrisch in (x_1, x_2). f ist aber nicht symmetrisch in $\{x_1, \ldots, x_4\}$, da $f(0,1,0,1) = 0$ und $f(1,0,1,0) = 1$.

Beispiel 2.6 Anhand der folgenden Funktion f sieht man, daß Lemma 2.9 für $|[x_i]| > 1$ im allgemeinen falsch ist:

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	dc
0	0	0	1	dc
0	0	1	0	dc
0	0	1	1	0
0	1	0	0	dc
0	1	0	1	dc
0	1	1	0	dc
0	1	1	1	dc
1	0	0	0	dc
1	0	0	1	dc
1	0	1	0	dc
1	0	1	1	dc
1	1	0	0	1
1	1	0	1	dc
1	1	1	0	dc
1	1	1	1	dc

Man sieht leicht, daß f stark symmetrisch ist in $\{\{x_1, x_2\}, \{x_3, x_4\}\}$. Die Gewichtsklasse $C_{0,2} = \{(0,0,1,1)\}$ wird durch f auf 0 abgebildet, die Gewichtsklasse $C_{2,0} = \{(1,1,0,0)\}$ auf 1 und alle anderen Gewichtsklassen dieser Partition werden auf dc abgebildet. f ist außerdem offensichtlich symmetrisch in (x_1, x_3). f ist aber nicht symmetrisch in $\{x_1, \ldots, x_4\}$, da $f(0,0,1,1) = 0$ und $f(1,1,0,0) = 1$.

Es folgt der Beweis des Lemmas:

Beweis: Sei $P = \{\lambda_1, \ldots, \lambda_k\}$ und sei o.B.d.A. $\lambda_1 = \{x_i\}$, $\lambda_2 = [x_j]$. Dann ist $P' = \{\lambda_1 \cup \lambda_2, \lambda_3, \ldots, \lambda_k\}$.
Nach Lemma 2.8 genügt es zum Nachweis der Symmetrie von f in P' zu zeigen, daß das Bild keiner Gewichtsklasse zu P' unter f gleichzeitig 0 und 1 enthält.
Jede Gewichtsklasse $C^{P'}_{w_2,\ldots,w_k}$ zu P' mit $w_2 \geq 1$ läßt sich als disjunkte Vereinigung zweier Gewichtsklassen zu P schreiben:

$$C^{P'}_{w_2,\ldots,w_k} = C^{P}_{0,w_2,\ldots,w_k} \cup C^{P}_{1,w_2-1,w_3,\ldots,w_k}.$$

Da f stark symmetrisch ist in P, gilt nach Lemma 2.8

$$|f(C^P_{0,w_2,\ldots,w_k})| = |f(C^P_{1,w_2-1,w_3,\ldots,w_k})| = 1.$$

Falls $\{0,1\} \subseteq f(C^{P'}_{w_2,\ldots,w_k})$, dann müßte $f(C^P_{1,w_2-1,\ldots,w_k}) = c$ und $f(C^P_{0,w_2,\ldots,w_k}) = \overline{c}$, $c \in \{0,1\}$ gelten.
Es gibt aber $\epsilon \in \{0,1\}^n$ mit $\epsilon \in C^P_{0,w_2,\ldots,w_k}$ und $\sigma_{ij}(\epsilon) \in C^P_{1,w_2-1,w_3,\ldots,w_k}$. Da f aber symmetrisch ist in (x_i, x_j), muß $f(\epsilon) = c$ und $f(\sigma_{i,j}(\epsilon)) = \overline{c}$ falsch sein und infolgedessen gilt $\{0,1\} \not\subseteq f(C^{P'}_{w_2,\ldots,w_k})$.
Ist bei der Gewichtsklasse $C^{P'}_{w_2,\ldots,w_k}$ zu P' $w_2 = 0$, so gilt

$$C^{P'}_{w_2,\ldots,w_k} = C^P_{0,w_2,\ldots,w_k},$$

und $\{0,1\} \not\subseteq f(C^{P'}_{w_2,\ldots,w_k})$ folgt aus der starken Symmetrie von f in P.

Folglich gibt es keine Gewichtsklasse zu P', deren Bild unter f sowohl 0 als auch 1 enthält und nach Lemma 2.8 ist f symmetrisch in P'. □

Es bleibt noch zu zeigen, wie man in Schritt (**) (Zeile 18) des Algorithmus die Funktion f, die auf jeden Fall *schwach* symmetrisch ist in P, *stark* symmetrisch in P macht. Wegen Lemma 2.8 ist es klar, daß man eine in P stark symmetrische Erweiterung zu f finden kann, wenn man weiß, daß f schwach symmetrisch in P ist. Da die Gewichtsklassen zu P disjunkt sind, kann man einfach don't cares innerhalb von Gewichtsklassen, die durch f auf $\{0, dc\}$ bzw. $\{1, dc\}$ abgebildet werden, so festlegen, daß diese Gewichtsklassen auf $\{0\}$ bzw. $\{1\}$ abgebildet werden. Es bleibt noch das Problem, wie man die don't care–Belegung zur Erzeugung starker Symmetrie in P effizient durchführen kann.

In Abschnitt 2.2.1.1 wurde schon eine Prozedur *make_strongly_symm* angegeben, die es ermöglicht, ausgehend von einer partiellen Funktion f (repräsentiert durch f_{on}, f_{off} und f_{dc}), die symmetrisch ist in (x_i, x_j), unter Anwendung einfacher Boolescher Operationen auf Kofaktoren von f_{on}, f_{off} und f_{dc} die minimale in (x_i, x_j) stark symmetrische Erweiterung f' (repräsentiert durch f'_{on}, f'_{off} und f'_{dc}) zu berechnen. Sind f_{on}, f_{off} bzw. f_{dc} als kompakte ROBDD–Repräsentationen gegeben, so lassen sich f'_{on}, f'_{off} und f'_{dc} effizient berechnen.

Auch in Schritt (**) des Algorithmus soll eine *minimale Erweiterung* von f berechnet werden, die stark symmetrisch in P ist. Die Berechnung dieser minimalen Erweiterung soll auf eine Folge von Aufrufen von *make_strongly_symm* zurückgeführt werden. Der folgende Satz gibt eine Folge solcher Aufrufe an, die zu der minimalen Erweiterung von f führen, die stark symmetrisch ist in P.

Satz 2.3 (Melchior, Scholl '96) *Sei $f \in S(D)$ ($D \subseteq \{0,1\}^n$) stark symmetrisch in P, $\{x_i\}, [x_{j_1}] \in P$, $[x_{j_1}] = \{x_{j_1}, \ldots, x_{j_k}\}$, $f =: f^{(0)}$ symmetrisch in (x_i, x_{j_1}).*

$$\begin{aligned} f^{(1)} &= \mathit{make\_strongly\_symm}(f^{(0)}, x_i, x_{j_1}) \\ f^{(2)} &= \mathit{make\_strongly\_symm}(f^{(1)}, x_i, x_{j_2}) \end{aligned}$$

$$\vdots$$
$$f^{(k)} \;=\; make\_strongly\_symm(f^{(k-1)}, x_i, x_{j_k}).$$

Dann ist $f^{(k)}$ *stark symmetrisch in*

$$P' = P \setminus \{[x_{j_1}], \{x_i\}\} \bigcup \{[x_{j_1}] \cup \{x_i\}\}.$$

Zum Beweis des Satzes benötigt man zunächst das folgende Lemma:

Lemma 2.10 *Seien* $f \in S(D)$*,* $D \subseteq \{0,1\}^n$*,* $\{x_k, x_l\} \cap \{x_i, x_j\} = \emptyset$*,*
f *stark symmetrisch in* (x_k, x_l) *und* f *symmetrisch in* (x_i, x_j)
und sei $f' \in S(D')$ $(D \subseteq D')$ *das Ergebnis von* $make\_strongly\_symm(f, x_i, x_j)$*.*
Dann ist f' *weiterhin stark symmetrisch in* (x_k, x_l)*.*

Beweis: Annahme: f' ist nicht stark symmetrisch in (x_k, x_l).
Dann gibt es $\epsilon \in D' \setminus D$ mit $\sigma_{kl}(\epsilon) \notin D'$ oder $\sigma_{kl}(\epsilon) \in D'$, $f'(\sigma_{kl}(\epsilon)) \neq f'(\epsilon)$.
Dann gilt aber:

$$\begin{array}{llcll}
 & f'(\epsilon) & = & c, & c \in \{0,1\} \\
\Longrightarrow & f(\sigma_{ij}(\epsilon)) & = & c, & \text{da } f' \textit{ minimale} \text{ in } (x_i, x_j) \text{ stark symmetrische} \\
 & & & & \text{Erweiterung von } f \text{ ist} \\
\Longrightarrow & f(\sigma_{kl}(\sigma_{ij}(\epsilon))) & = & c, & \text{da } f \text{ stark symmetrisch in } (x_k, x_l) \\
\Longrightarrow & f'(\sigma_{ij}(\sigma_{kl}(\sigma_{ij}(\epsilon)))) & = & c, & \text{da } f' \text{ eine in } (x_i, x_j) \text{ stark symmetrische Erweiterung} \\
 & & & & \text{von } f \text{ ist} \\
\Longrightarrow & f'(\sigma_{kl}(\epsilon)) & = & c, & \text{da } \{x_k, x_l\} \cap \{x_i, x_j\} = \emptyset \\
 & & & & \text{und daher } \sigma_{ij} \circ \sigma_{kl} \circ \sigma_{ij} = \sigma_{kl}.
\end{array}$$

Somit ergibt sich ein Widerspruch zur Annahme. □

Es folgt der Beweis von Satz 2.3:

Beweis: Aus Lemma 2.10 ergibt sich, daß alle Funktionen $f^{(i)}$ $(1 \leq i \leq k)$ stark symmetrisch sind in (x_k, x_l) mit $x_k, x_l \notin [x_{j_1}]$, $x_k, x_l \neq x_i$, $[x_k] = [x_l]$.
Es bleibt noch zu zeigen, daß $f^{(k)}$ stark symmetrisch in $[x_{j_1}] \cup \{x_i\}$ ist.
Seien $\lambda_1 = \{x_i\}$, $\lambda_2 = \{x_{j_1}, \ldots, x_{j_k}\}$, $Q = \{\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{n+1-k}\}$ eine Partition von $\{x_1, \ldots, x_n\}$ mit $|\lambda_p| = 1$ für $3 \leq p \leq n+1-k$.
f ist nach Voraussetzung stark symmetrisch in λ_2 und damit auch stark symmetrisch in Q.
Dann bleibt zu zeigen, daß $f^{(k)}$ stark symmetrisch ist in $Q' = \{\lambda_1 \cup \lambda_2, \lambda_3, \ldots, \lambda_{n+1-k}\}$.
Daß $f^{(k)}$ dann auch stark symmetrisch ist in P', ergibt sich einfach aus Lemma 2.10, da starke Symmetrien in Variablenpaaren aus Klassen, die von $[x_{j_1}]$ verschieden sind, erhalten bleiben.
Es ist nach Lemma 2.8 also zu zeigen, daß für alle Gewichtsklassen $C^{Q'}_{w_{1,2}, w_3, \ldots, w_{n+1-k}}$ zu Q' gilt:

$$f^{(k)}(C^{Q'}_{w_{1,2}, w_3, \ldots, w_{n+1-k}}) = \begin{cases} \{0\} & \text{oder} \\ \{1\} & \text{oder} \\ \{dc\} & \end{cases}$$

Fall 1: $w_{1,2} = 0$ bzw. $w_{1,2} = k+1$

Dann gilt: $C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}} = C^{Q}_{0,0,w_3,\ldots,w_{n+1-k}}$ bzw. $C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}} = C^{Q}_{1,k,w_3,\ldots,w_{n+1-k}}$ und somit $|f(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}})| = 1$ wegen der starken Symmetrie von f in Q.

Falls nun $f(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = c$, $c \in \{0,1\}$, dann gilt $f^{(p)}(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = \{c\}$ für alle $1 \le p \le k$, da $f^{(p)}$ eine Erweiterung von f ist.

Falls $f(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = \{dc\}$, dann gilt auch $f^{(p)}(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = \{dc\}$ für alle $1 \le p \le k$, da *make_strongly_symm*$(f^{(p-1)}, x_i, x_{j_p})$ eine *minimale* Erweiterung liefert, die stark symmetrisch in (x_i, x_{j_p}) ist und $w^1_{\lambda_1}(\epsilon) = w^1_{\lambda_2}(\epsilon) = 0$ bzw. $w^0_{\lambda_1}(\epsilon) = w^0_{\lambda_2}(\epsilon) = 0$ für alle $\epsilon \in C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}$.

Fall 2: $1 \le w_{1,2} \le k$

Dann gilt $C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}} = C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}} \cup C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}$.

Nach Voraussetzung gilt

$$f(C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}) = \begin{cases} \{0\} \text{ oder} \\ \{1\} \text{ oder} \\ \{dc\} \end{cases} \quad \text{und} \quad f(C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}) = \begin{cases} \{0\} \text{ oder} \\ \{1\} \text{ oder} \\ \{dc\} \end{cases}$$

Fall 2.1: $f(C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}) = f(C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}})$

Da die Aufrufe *make_strongly_symm*$(f^{(p-1)}, x_i, x_{j_p})$ minimale Erweiterungen liefern, die stark symmetrisch in (x_i, x_{j_p}) sind, wird die Belegung für $C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}$ und $C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}$ nicht verändert.

Es gilt: $f^{(p)}(C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}) = f^{(p)}(C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}})$ und damit

$$f^{(p)}(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = \begin{cases} \{0\} & \text{oder} \\ \{1\} & \text{oder} \\ \{dc\} \end{cases}$$

Fall 2.2: $f(C^{Q}_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}) \ne f(C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}})$

Da f symmetrisch in (x_i, x_{j_1}) ist, gibt es $c \in \{0,1\}$ und $u \in \{0,1\}$, so daß $f(C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}) = \{dc\}$ und $f(C^{Q}_{\bar{u},w_{1,2}-\bar{u},w_3,\ldots,w_{n+1-k}}) = \{c\}$.

Nach Definition von *make_strongly_symm* ist klar, daß für alle $1 \le p \le k$ $f^{(p)}(\epsilon) \in \{c, dc\}\ \forall \epsilon \in C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$.

Ein Aufruf von *make_strongly_symm*$(f^{(p)}, x_i, x_{j_{p+1}})$ ordnet Vektoren $\epsilon \in C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ mit $\epsilon_i = u$ und $\epsilon_{j_{p+1}} = \overline{u}$ den Funktionswert $f^{(p)}(\sigma_{ij_{p+1}}(\epsilon)) = c$ zu $(\sigma_{ij_{p+1}}(\epsilon) \in C^{Q}_{\bar{u},w_{1,2}-\bar{u},w_3,\ldots,w_{n+1-k}})$.

Es bleibt noch zu zeigen, daß $f^{(k)}(\epsilon) = c\ \forall \epsilon \in C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$, d.h. daß die Folge der k Aufrufe genügt, um *alle* Elemente von $C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ mit Funktionswert c zu belegen.

Folgende Aussage wird induktiv gezeigt:

$f^{(p)}(\epsilon) = c\ \forall \epsilon \in C^{Q}_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ mit $\epsilon_{j_1} = \bar{u}$ oder $\epsilon_{j_2} = \bar{u}$ oder $\ldots$ oder $\epsilon_{j_p} = \bar{u}$.

$p = 0$: Nichts zu zeigen.

$p \to p + 1$:
Wegen Induktionsvoraussetzung und da $f^{(p+1)}$ Erweiterung von $f^{(p)}$ ist, gilt: $f^{(p+1)}(\epsilon) = c\ \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ mit $\epsilon_{j_1} = \bar{u}$ oder $\epsilon_{j_2} = \bar{u}$ oder ... oder $\epsilon_{j_p} = \bar{u}$.

Es bleibt zu zeigen, daß $f^{(p+1)}(\epsilon) = c\ \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ mit $\epsilon_{j_{p+1}} = \bar{u}$.

Sei nun $\delta \in C^Q_{\bar{u},w_{1,2}-\bar{u},w_3,\ldots,w_{n+1-k}}$ mit

$$\delta_i = \overline{\epsilon_i} = \bar{u}^4, \delta_{j_{p+1}} = \overline{\epsilon_{j_{p+1}}} = u \text{ und } \delta_l = \epsilon_l \text{ für } l \neq i, j_{p+1},$$

also $\delta = \sigma_{i,j_{p+1}}(\epsilon)$. (Ein solches $\delta \in C^Q_{\bar{u},w_{1,2}-\bar{u},w_3,\ldots,w_{n+1-k}}$ gibt es wegen $1 \leq w_{1,2} \leq k$).
Es gilt

$$f^{(p)}(\delta) = f(\delta) = c$$

und folglich

$$f^{(p+1)}(\epsilon) = f^{(p)}(\sigma_{i,j_{p+1}}(\epsilon)) = f^{(p)}(\delta) = c.$$

Aus der induktiv gezeigten Behauptung folgt:

$$f^{(k)}(\epsilon) = c\ \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}} \text{ mit } \epsilon_{j_1} = \bar{u} \text{ oder } \ldots \text{ oder } \epsilon_{j_k} = \bar{u}$$

bzw.

$$f^{(k)}(\epsilon) = c\ \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}} \text{ mit } w^{\bar{u}}_{\lambda_2}(\epsilon) \geq 1.$$

$w^{\bar{u}}_{\lambda_2}(\epsilon) \geq 1$ gilt aber für *alle* $\epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$:
Falls $u = 0$, dann gilt für alle $\epsilon \in C^Q_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}$:

$$w^{\bar{u}}_{\lambda_2}(\epsilon) = w^1_{\lambda_2}(\epsilon) = w_{1,2} \overset{Fallannahme 2.2}{\geq} 1.$$

Falls $u = 1$, dann gilt für alle $\epsilon \in C^Q_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}$:

$$w^{\bar{u}}_{\lambda_2}(\epsilon) = w^0_{\lambda_2}(\epsilon) = k - w^1_{\lambda_2}(\epsilon) = k - (w_{1,2} - 1) \overset{Fallannahme 2.2}{\geq} 1.$$

Folglich ist $f^{(k)}(\epsilon) = c\ \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ gezeigt.

□

Das folgende Beispiel zeigt, daß es partielle Funktionen f gibt, die den Voraussetzungen des Satzes 2.3 genügen und für die es nötig ist, die komplette Folge aller k Aufrufe von *make_strongly_symm* auszuführen:

[4]Für alle Elemente ϵ der Gewichtsklasse $C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$ ist $\epsilon_i = u$.

Beispiel 2.7 Sei eine Partition $P = \{\{x_1\}, \{x_2, \ldots, x_n\}\}$ gegeben. Betrachte eine wie folgt definierte Beispielfunktion $f \in S(D)$, $D \subseteq \{0,1\}^n$:

$$f(\epsilon) = \begin{cases} 0 & \text{falls } \epsilon \in C^P_{1,0}, \text{ d.h. } \epsilon = (1, 0, \ldots, 0) \\ dc & \text{falls } \epsilon \in C^P_{0,1} \\ 1 & \text{sonst} \end{cases}$$

Wie man leicht sieht, ist f stark symmetrisch in P und symmetrisch in (x_1, x_2). Um f stark symmetrisch in $\{\{x_1, \ldots, x_n\}\}$ zu machen, müssen alle dc–Stellen aus $C^P_{0,1}$ mit 0 belegt werden. Betrachte nun die Aufruffolge

$$\begin{aligned} f^{(1)} &= \mathit{make\_strongly\_symm}(f, x_1, x_2) \\ f^{(2)} &= \mathit{make\_strongly\_symm}(f^{(1)}, x_1, x_3) \\ &\vdots \\ f^{(n-1)} &= \mathit{make\_strongly\_symm}(f^{(n-2)}, x_1, x_n). \end{aligned}$$

Es ist klar, daß beim Aufruf von $\mathit{make\_strongly\_symm}(f^{(i-2)}, x_1, x_i)$ exakt eine dc–Stelle aus $C^P_{0,1}$ mit 0 belegt wird, nämlich die Stelle $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ mit $\epsilon_1 = 0$, $\epsilon_i = 1$ und $\epsilon_k = 0$ für $k \in \{2, \ldots, n\} \setminus \{i\}$. Alle $n-1$ Aufrufe werden also benötigt.

Betrachtet man sich den Beweis zu Satz 2.3 genauer, so sieht man, daß abhängig von der speziellen Wahl von f die Folge der *make_strongly_symm*–Aufrufe schon vorzeitig zu einem stabilen Ergebnis führen kann. Die minimale Länge dieser Aufruffolge, die man benötigt, um die minimale in P' stark symmetrische Erweiterung von f zu berechnen, kann im voraus für f bestimmt werden und ist in Satz 2.4 angegeben:

Satz 2.4 (Melchior/Scholl '96) *Sei $f \in S(D)$ ($D \subseteq \{0,1\}^n$) stark symmetrisch in P, $\{x_i\}, [x_{j_1}] \in P$, $[x_{j_1}] = \{x_{j_1}, \ldots, x_{j_k}\}$, $\lambda := [x_{j_1}] \setminus \{x_{j_1}\}$. $f =: f^{(0)}$ symmetrisch in (x_i, x_{j_1}). Die Funktionen $change_{10}$ und $change_{01} \in B_n$ sind definiert durch*

$$change_{10} = f_{on_{\overline{x_i} x_{j_1}}} \cdot f_{dc_{x_i \overline{x_{j_1}}}} + f_{off_{\overline{x_i} x_{j_1}}} \cdot f_{dc_{x_i \overline{x_{j_1}}}}$$

und

$$change_{01} = f_{on_{x_i \overline{x_{j_1}}}} \cdot f_{dc_{\overline{x_i} x_{j_1}}} + f_{off_{x_i \overline{x_{j_1}}}} \cdot f_{dc_{\overline{x_i} x_{j_1}}}.$$

Seien

$$ml_1 = \begin{cases} \max_{x \in ON(change_{10})}(w^1_\lambda(x)) + 1 & \text{falls } ON(change_{10}) \neq \emptyset \\ 0 & \text{falls } ON(change_{10}) = \emptyset, \end{cases}$$

$$ml_2 = \begin{cases} \max_{x \in ON(change_{01})}(w^0_\lambda(x)) + 1 & \text{falls } ON(change_{01}) \neq \emptyset \\ 0 & \text{falls } ON(change_{01}) = \emptyset, \end{cases}$$

$$ml = \max(ml_1, ml_2).$$

$$\begin{aligned} f^{(1)} &= make\_strongly\_symm(f^{(0)}, x_i, x_{j_1}) \\ f^{(2)} &= make\_strongly\_symm(f^{(1)}, x_i, x_{j_2}) \\ &\vdots \\ f^{(ml)} &= make\_strongly\_symm(f^{(ml-1)}, x_i, x_{j_{ml}}). \end{aligned}$$

Dann ist $f^{(ml)}$ stark symmetrisch in

$$P' = P \setminus \{[x_{j_1}], \{x_i\}\} \bigcup \{[x_{j_1}] \cup \{x_i\}\}.$$

Beweis: Der Beweis erfolgt analog zum Beweis von Satz 2.3 und mit den gleichen Bezeichnungen wie im Beweis zu Satz 2.3. Der Beweis für die Fälle 1 und 2.1 ist identisch. Für Fall 2.2 zeigt man (wie im Beweis von Satz 2.3 durch Induktion), daß gilt:

$$f^{(ml)}(\epsilon) = c \; \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}} \text{ mit } \epsilon_{j_1} = \bar{u} \text{ oder } \ldots \text{ oder } \epsilon_{j_{ml}} = \bar{u}. \qquad (\star)$$

Analog zum Beweis zu Satz 2.3 ist nun noch zu zeigen, daß damit schon

$$f^{(ml)}(\epsilon) = c \; \forall \epsilon \in C^Q_{u,w_{1,2}-u,w_3,\ldots,w_{n+1-k}}$$

gilt.

Für $u = 1$ gilt: Wegen $w_{1,2} \leq k$ gibt es $\delta \in C^Q_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}$ mit $\delta_{j_1} = 0$.

Es gilt weiterhin $\delta_i = 1$ und $\sigma_{ij_1}(\delta) \in C^Q_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}$.

Es gilt also (gemäß den Annahmen und Bezeichnungen aus dem Beweis zu Satz 2.3)

$$f(\delta) = dc \text{ und } f(\sigma_{ij_1}(\delta)) = c.$$

Folglich gilt $\delta \in ON(change_{10})$.

$$\begin{aligned} &\Longrightarrow w^1_\lambda(\delta) &\leq\; & ml_1 - 1 &\leq\; & ml - 1 & \\ &\Longrightarrow w^1_{\lambda_2}(\delta) &=\; & w^1_\lambda(\delta) &\leq\; & ml - 1 & (\lambda_2 = \lambda \cup \{x_{j_1}\}) \\ &\Longrightarrow w^1_{\lambda_2}(\epsilon) &=\; & w^1_{\lambda_2}(\delta) &\leq\; & ml - 1 & \forall \epsilon \in C^Q_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}. \end{aligned}$$

$$\begin{aligned} &\Longrightarrow \neg(\epsilon_{j_1} = 1 \wedge \ldots \wedge \epsilon_{j_{ml}} = 1) \\ &\Longrightarrow \epsilon_{j_1} = 0 \vee \ldots \vee \epsilon_{j_{ml}} = 0 \end{aligned}$$

Und folglich wegen $(\star)$:

$$f^{(ml)}(\epsilon) = c \; \forall \epsilon \in C^Q_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}.$$

Für $u = 0$ gilt genau analog: Wegen $w_{1,2} \geq 1$ gibt es $\delta \in C^Q_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}$ mit $\delta_{j_1} = 1$.

Es gilt weiterhin $\delta_i = 0$ und $\sigma_{ij_1}(\delta) \in C^Q_{1,w_{1,2}-1,w_3,\ldots,w_{n+1-k}}$.

Es gilt also

$$f(\delta) = dc \text{ und } f(\sigma_{ij_1}(\delta)) = c.$$

Folglich gilt $\delta \in ON(change_{01})$.

$$\begin{array}{llllll} \Longrightarrow & w^0_\lambda(\delta) & \leq & ml_2 - 1 & \leq & ml - 1 \\ \Longrightarrow & w^0_{\lambda_2}(\delta) & = & w^0_\lambda(\delta) & \leq & ml - 1 \quad (\lambda_2 = \lambda \cup \{x_{j_1}\}) \\ \Longrightarrow & w^0_{\lambda_2}(\epsilon) & = & w^0_{\lambda_2}(\delta) & \leq & ml - 1 \quad \forall \epsilon \in C^Q_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}. \end{array}$$

$$\begin{array}{ll} \Longrightarrow & \neg(\epsilon_{j_1} = 0 \wedge \ldots \wedge \epsilon_{j_{ml}} = 0) \\ \Longrightarrow & \epsilon_{j_1} = 1 \vee \ldots \vee \epsilon_{j_{ml}} = 1 \end{array}$$

Und folglich wegen $(\star)$:

$$f^{(ml)}(\epsilon) = c \;\forall \epsilon \in C^Q_{0,w_{1,2},w_3,\ldots,w_{n+1-k}}.$$

□

Es bieten sich nun 2 Vorgehensweisen zur Berechnung einer minimalen in $P' = P \setminus \{[x_{j_1}], \{x_i\}\} \bigcup \{[x_{j_1}] \cup \{x_i\}\}$ symmetrischen Erweiterung von f an:

- Man berechnet ROBDDs zu $change_{01}$ und $change_{10}$ und bestimmt $\max_{x \in ON(change_{10})}(w^1_\lambda(x))$ und $\max_{x \in ON(change_{01})}(w^0_\lambda(x))$. Dies kann mit einem bottom up–Durchlauf der entsprechenden ROBDDs geschehen. Danach erfolgen nur so viele Aufrufe von *make_strongly_symm* wie benötigt werden („ml“ aus Satz 2.4).
- Man führt die in Satz 2.3 angegebene Folge von Aufrufen von *make_strongly_symm* aus. Falls sich die Funktion bei einem Aufruf von *make_strongly_symm* nicht mehr ändert, bricht man die Aufruffolge ab. Anhand des Beweises zu Satz 2.4 sieht man leicht ein, daß man dann genau $ml + 1$ Aufrufe durchgeführt hat:
  Es ist klar, daß nach ml Aufrufen von *make_strongly_symm* $|f^{(ml)}(C^{Q'}_{w_{1,2},w_3,\ldots,w_{n+1-k}})| = 1$ (mit den Bezeichnungen aus dem Beweis zu Satz 2.4) und daß daher $f^{(ml)} = \ldots = f^{(k)}$.
  Es bleibt zu zeigen, daß $f^{(0)} \neq f^{(1)} \neq \ldots \neq f^{(ml)}$:
  Sei $ml > 0$ und o.B.d.A. $ml = ml_1$ (für $ml = ml_2$ analog). Dann gibt es $\epsilon \in \{0,1\}^n$ mit $\epsilon_i = 1$, $\epsilon_{j_1} = 0$, $w^1_\lambda(\epsilon) = ml_1 - 1$ und $f(\epsilon) = dc$, $f(\sigma_{ij_1}(\epsilon)) = c \in \{0,1\}$.
  Da f stark symmetrisch in $\{x_{j_1}, \ldots, x_{j_k}\}$, gilt auch für $\epsilon^{(p)} \in \{0,1\}^n$ $(1 \leq p \leq ml_1)$ mit $\epsilon^{(p)}_i = 1$, $\epsilon^{(p)}_{j_1} = \ldots = \epsilon^{(p)}_{j_{p-1}} = 1$, $\epsilon^{(p)}_{j_p} = 0, \epsilon^{(p)}_{j_{p+1}} = \ldots = \epsilon^{(p)}_{j_{ml_1}} = 1, \epsilon^{(p)}_{j_{ml_1+1}} = \ldots = \epsilon^{(p)}_{j_k} = 0, \epsilon^{(p)}_j = \epsilon_j$ sonst:
  $$f(\epsilon^{(p)}) = dc \text{ und } f(\sigma_{ij_p}(\epsilon^{(p)})) = c.$$
  Wegen $\sigma_{ij_l}(\epsilon^{(p)}) = \epsilon^{(p)}$ für $l < p$ gilt $f^{(0)}(\epsilon^{(p)}) = \ldots = f^{(p-1)}(\epsilon^{(p)}) = dc$.
  Der Aufruf *make_strongly_symm*$(f^{(p-1)}, x_i, x_{j_p})$ liefert $f^{(p)}(\epsilon^{(p)}) = f^{(p-1)}(\sigma_{ij_p}(\epsilon^{(p)})) = c$ und damit $f^{(p-1)} \neq f^{(p)}$ für $1 \leq p \leq ml_1$.

2.2.2 Erweiterung auf Äquivalenzsymmetrie

Der vorgestellte Ansatz zur Bestimmung von Vertauschungssymmetrien partieller Funktionen läßt sich leicht erweitern, so daß auch nach Äquivalenzsymmetrien gesucht wird. Nach Abschnitt 2.1.2 kann die Suche nach Vertauschungssymmetrien und Äquivalenzsymmetrien einer Funktion f auf die Suche nach einem Polaritätsvektor POL und einer Partition P zurückgeführt werden, so daß f_{POL} vertauschungssymmetrisch ist in P.

Aufgrund dieser Tatsache läßt sich das Verfahren aus Abbildung 2.5 in Hinblick auf eine Suche auch nach Äquivalenzsymmetrien leicht abändern. Es arbeitet jetzt nicht mehr auf dem Symmetriegraphen G_{sym} (bzw. dem dazu inversen Graphen $\overline{G_{sym}}$ von f), sondern auf G_{esym} (vgl. Abschnitt 2.1.2).

In Abbildung 2.6 ist ein Algorithmus zur Lösung des auf Äquivalenzsymmetrien erweiterten Problems angegeben. Zusätzlich zu der aktuellen Partition P wird ein Polaritätsvektor $POL = (pol_1, \ldots, pol_n)$ mitgeführt. Der Polaritätsvektor ist mit $(1, \ldots, 1)$ initialisiert (Zeile 4). Wird ein Knoten x_i gefärbt, so wird die Polarität von x_i auf pol_j festgelegt, falls es einen Knoten x_j mit gleicher Farbe gibt und f vertauschungssymmetrisch ist in (x_i, x_j) (Zeile 18). Ansonsten wird die die Polarität von x_i auf $\overline{pol_j}$ festgelegt, falls es einen Knoten x_j mit gleicher Farbe gibt und f äquivalenzsymmetrisch ist in (x_i, x_j) (Zeile 25). Nach Ablauf des Algorithmus erhält man eine Partition P, einen Polaritätsvektor POL und eine Erweiterung f der ursprünglichen Funktion mit der Eigenschaft, daß f_{POL} stark symmetrisch ist in P.

2.2.3 Anwendung auf die Minimierung von ROBDDs für partielle Funktionen

Wie schon in Abschnitt 1.1.2.3 angedeutet wurde, kann man die Bestimmung einer möglichst kleinen Partition, in der eine partielle Funktion f symmetrisch ist, dazu ausnutzen, eine möglichst kleine ROBDD–Darstellung zu f zu finden.

In sehr vielen Anwendungen, die mit Booleschen Funktionen arbeiten, werden die Booleschen Funktionen intern durch ROBDDs dargestellt. Laufzeit und Speicherplatzbedarf dieser Anwendungen hängen daher in hohem Maße davon ab, wie kompakt diese Darstellung erfolgen kann. Die auftretenden Booleschen Funktionen sind in vielen Fällen nur partiell, so daß sich das Problem stellt, totale Erweiterungen dieser Booleschen Funktionen zu finden, die möglichst kleine ROBDD–Darstellungen besitzen. Beispiele für solche Anwendungen sind der Äquivalenztest zweier endlicher Automaten wie er von Coudert/Berthet/Madre [CBM89] vorgestellt wurde oder die Minimierung der Übergangsfunktion endlicher Automaten im Hinblick auf nicht erreichbare Zustände. Eine weitere Anwendung ist in Kapitel 3 beschrieben: Treten bei der Logiksynthese Boolescher Funktionen partielle Funktionen auf, so ist eine gute Belegung der don't cares nicht nur für die Laufzeit der Verfahren, sondern auch für die Güte der Resultate von essentieller Bedeutung. Es wird sich herausstellen, daß es für

Eingabe: Partielle Funktion $f \in S(D)$, $D \subseteq \{0,1\}^n$, repräsentiert durch f_{on} und f_{dc}

Ausgabe: Partition P von $\{x_1, \ldots, x_n\}$ und Polaritätsvektor $POL \in \{0,1\}^n$, so daß f_{POL} (vertauschungs)symmetrisch ist in P

Algorithmus:

Berechne Symmetriegraph $G_{esym} = (V, E)$ zu f (oder $\overline{G_{esym}} = (V, \bar{E})$).
$\forall 1 \leq k \leq n : color(x_k) := undef.$
$P = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$
$POL = (pol_1, \ldots, pol_n) = (1, \ldots, 1)$
$node\_candidate\_set := \{x_1, \ldots, x_n\}$
while ($node\_candidate\_set \neq \emptyset$) **do**
  /* f_{POL} ist stark symmetrisch in P */
  Wähle $x_i \in node\_candidate\_set$ gemäß Brélaz/Morgenstern–Kriterium
  $color\_candidate\_set := \{c \,|\, 1 \leq c \leq n, \nexists x_j$ mit $\{x_i, x_j\} \in \bar{E}$ und $color(x_j) = c\}$
  while ($color(x_i) = undef.$) **do**
    $curr\_color := \min(color\_candidate\_set)$
    $color(x_i) := curr\_color$
    if ($\exists$ gefärbter Knoten x_j mit $color(x_j) = color(x_i)$)
     then
      if (f symmetrisch in (x_i, x_j))
       then
        $P := P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$
        $pol_i = pol_j$
        /* f_{POL} ist symmetrisch in P */
        Mache f_{POL} *stark symmetrisch* in P.
       else
        if (f äquivalenzsymmetrisch in (x_i, x_j))
         then
          $P := P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$
          $pol_i = \overline{pol_j}$
          /* f_{POL} ist symmetrisch in P */
          Mache f_{POL} *stark symmetrisch* in P.
         else
          $color\_candidate\_set := color\_candidate\_set \setminus \{curr\_color\}$
          $color(x_i) := undef.$
        fi
      fi
    fi
  od
  $node\_candidate\_set := node\_candidate\_set \setminus \{x_i\}$
od

Abbildung 2.6: Algorithmus zur Bestimmung von Vertauschungssymmetrien *und* Äquivalenzsymmetrien.

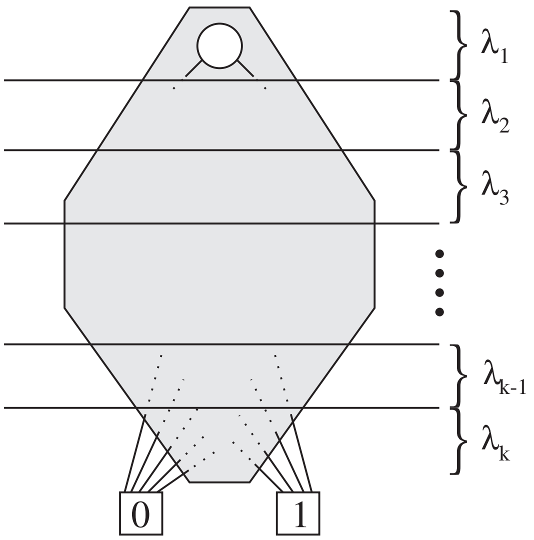


Abbildung 2.7: ROBDD mit symmetrischer Ordnung zu f, f ist symmetrisch in $\{\lambda_1, \ldots, \lambda_k\}$.

die Zerlegungsverfahren aus Kapitel 3 sehr günstig ist, für partielle Boolesche Funktionen zunächst Erweiterungen mit möglichst kleinen ROBDD–Darstellungen zu bestimmen.

In Abschnitt 1.1.2.3 wurde schon festgehalten, daß es bei der Suche nach guten Variablenordnungen für ROBDDs eine gute Heuristik ist, nach „symmetrischen Ordnungen" zu suchen. Aus diesem Grund kann man bei der ROBDD–Minimierung partieller Funktionen folgendermaßen vorgehen:

- Zunächst bestimmt man zu einer partiellen Booleschen Funktion f eine möglichst kleine Partition $P = \{\lambda_1, \ldots, \lambda_k\}$, in der diese Funktion symmetrisch ist. (Hierbei ist es durchaus sinnvoll, „erweiterte Symmetrie" (d.h. Vertauschungs- *oder* Äquivalenzsymmetrie) zu betrachten.)

- Es wird eine Erweiterung f' von f bestimmt, die stark symmetrisch ist in P. Ist f' noch nicht total, so wird z.B. eine totale Erweiterung f'' von f' gewählt, die alle verbleibenden don't cares auf den gleichen Wert setzt, so daß diese totale Erweiterung symmetrisch ist in P.

- Nun wird f'' durch einen ROBDD dargestellt mit einer Variablenordnung, in der die Variablen aus den Mengen λ_i benachbart sind (siehe Abbildung 2.7). Man kann nun noch versuchen, die Variablenordnung zu verbessern, indem man die Variablen aus den Mengen λ_i *blockweise* vertauscht. Dazu bietet sich beispielsweise „symmetrisches sifting" [MMD94, PSP94] an.

- Nach der Bestimmung einer Variablenordnung läßt sich die ROBDD–Darstellung der partiellen Booleschen Funktion evtl. noch weiter verkleinern, falls die minimale in P stark symmetrische Erweiterung von f noch don't care–Stellen hatte. Die verbleibenden don't cares können unter Beibehaltung der Variablenordnung noch so belegt werden, daß die resultierende Erweiterung von f eine kleinere ROBDD–Darstellung hat, aber weiterhin stark symmetrisch in P bleibt. In Kapitel 3 wird ein Verfahren angegeben, das diese Aufgabe erfüllt.

Experimente anhand von Benchmarkfunktionen ergeben insgesamt eine starke Verkleinerung der ROBDD–Größen zur Repräsentation partieller Boolescher Funktionen (siehe [SMHM96] bzw. Tabelle auf Seite 194).

Kapitel 3

Zerlegungen

In diesem Kapitel wird die Realisierung Boolescher Funktionen durch (rekursive) Zerlegungen betrachtet.

Teile der hier angegebenen Resultate haben Eingang in die Veröffentlichungen [SM93, MS94, SM94, SM95a, SM95b, SMHM96] gefunden.

Die Durchführung von disjunkten Zerlegungen bei der Logiksynthese wurde schon in Arbeiten von Ashenhurst [Ash59], Curtis [Cur61] und Karp [Kar63] erstmals betrachtet.

Die Ansätze von Ashenhurst, Curtis und Karp wurden vor allem in folgenden Punkten entscheidend weiterentwickelt:

- Es wird einer essentiellen Problemstellung bei der mehrstufigen Logiksynthese Rechnung getragen: *der Identifizierung von Teillogik, die in mehreren Schaltungsteilen mit Vorteil verwendet werden kann.* Dies gelingt durch die Ausnutzung von Freiheiten bei der Wahl der sog. Zerlegungsfunktionen im Rahmen des Zerlegungsverfahrens. Bei der Behandlung von Funktionen $f = (f_1, \ldots, f_m)$ hat man hierbei das Problem zu lösen, die Zerlegungsfunktionen so zu berechnen, daß möglichst viele von ihnen in der Zerlegung möglichst vieler Ausgangsfunktionen f_i verwendet werden können.

- Bei der Auswahl von Zerlegungsfunktionen erfolgt eine Beschränkung auf eine bestimmte Teilklasse von Funktionen, nämlich die *strikten* Zerlegungsfunktionen. Es konnte gezeigt werden, daß sich bei strikten Zerlegungsfunktionen Struktureigenschaften der ursprünglichen Funktion auf die Zerlegungsfunktionen übertragen. Dies ist deshalb wichtig, da Funktionen aus praktischen Beispielen in der Regel keine „zufällig gewählten“ Funktionen sind, sondern Regelmäßigkeiten und Struktureigenschaften aufweisen.

- Symmetrien werden bei der Zerlegung ausgenutzt, indem bei der Aufteilung der Eingangsvariablen in „freie“ und „gebundene“ Variablen der Zerlegung Mengen symmetrischer Variablen möglichst als Ganzes in die Menge der gebundenen Variablen aufgenommen werden. Durch Symmetrien in der Menge der gebundenen Variablen kann man zu einer erheblichen Reduzierung der Anzahl der benötigten Zerlegungsfunktionen kommen.

- Es wurden Verfahren zur Behandlung *partieller* Funktionen entwickelt. Da bei praktischen Problemen häufig partielle Funktionen auftreten und sich die Güte der Realisierungen je nach Belegung der don't cares stark unterscheiden kann, ist es wichtig, bei dem Logiksyntheseverfahren don't cares ausnutzen zu können. Die Belegung von don't cares wird nicht nur mit Hilfe des im vorangegangenen Kapitel entwickelten Verfahrens zur Erzeugung starker Symmetrien durchgeführt, sondern auch durch ein Verfahren zur Minimierung der Anzahl der Zerlegungsfunktionen im aktuellen Zerlegungsschritt. Unter Beachtung bestimmter Voraussetzungen wird die Verträglichkeit der beiden Verfahren bewiesen. Darüber hinaus wird ein Verfahren zur don't care–Belegung im Hinblick auf die Mehrfachverwendbarkeit bestimmter Teilschaltkreise angegeben.

Betrachtet man die praktische Anwendbarkeit der Algorithmen auf größere Beispiele, so ist im Vergleich zu [MS94] in [SM94] bzw. [SM95b] ein entscheidender Durchbruch gelungen. Der Grund liegt in der Verwendung von ROBDDs zur Repräsentation Boolescher Funktionen und in der Tatsache, daß sämtliche Algorithmen auf Grundlage von ROBDDs formuliert werden konnten. Damit kann für viele praktische Beispiele auf einer kompakten Darstellung gearbeitet werden.

Das Kapitel beginnt mit einem Abschnitt über Zerlegungen von Funktionen mit einem Ausgang, in dem hauptsächlich Grundlagen zur Zerlegung Boolescher Funktionen behandelt werden.

Danach folgt ein Abschnitt, der sich mit don't care–Belegungen bei der Zerlegung partieller Funktionen beschäftigt.

In einem Abschnitt über Zerlegungen von Funktionen mit mehreren Ausgängen wird ein Verfahren hergeleitet, das eingesetzt wird, um Teilschaltkreise zu finden, die bei der Realisierung *mehrerer* Ausgänge mit Vorteil verwendet werden können.

Im nächsten Abschnitt wird dann festgehalten, wie man Symmetrien im Rahmen des Zerlegungsverfahrens nutzen kann (sowohl bei totalen als auch bei partiellen Funktionen).

Schließlich werden die Überlegungen auf sogenannte nichtdisjunkte Zerlegungen ausgedehnt. Es wird gezeigt, daß sich die Verwendung nichtdisjunkter Zerlegungen ohne größere Probleme in die bisher entwickelten Syntheseverfahren integrieren läßt.

Das Kapitel endet mit einem Abschnitt über die Berechnung von Fertigungstests zu rekursiv zerlegten Schaltungen. Es wird gezeigt, wie man parallel zur rekursiven Zerlegung einen vollständigen Test hinsichtlich des Stuck-at–Fehlermodells bzw. des Zellenfehlermodells berechnen kann. Dazu ist es notwendig, zu einer Booleschen Funktion f, die im Rahmen des rekursiven Zerlegungsverfahrens realisiert werden soll, jeweils noch zusätzlich die sogenannte „Freiheitsrelation“ zu f zu berechnen.

3.1 Zerlegungen von Funktionen mit 1 Ausgang

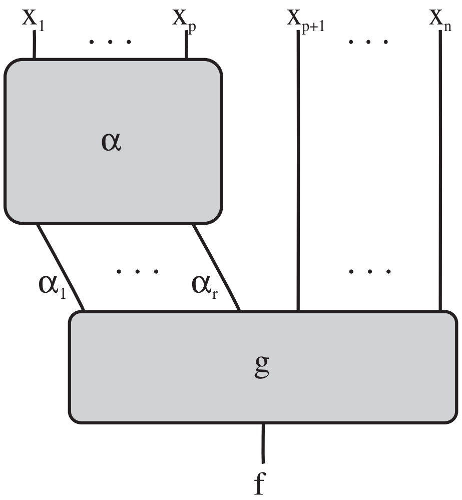
In diesem Abschnitt werden anhand von Funktionen mit einem Ausgang Grundlagen zur Zerlegung Boolescher Funktionen behandelt. Nach grundlegenden Definitionen und allgemeinen Untersuchungen zur nichttrivialen Zerlegbarkeit Boolescher Funktionen wird speziell auf die Zerlegung von Funktionen eingegangen, die durch ROBDD–Darstellungen gegeben sind. Außerdem wird gezeigt, inwiefern man aus der Beschränkung auf die Teilklasse der „strikten“ Zerlegungsfunktionen Nutzen ziehen kann.

Zunächst werden 2 Arten von (disjunkten) Zerlegungen definiert: die einseitige und die zweiseitige Zerlegung.

Definition 3.1 (Einseitige Zerlegung) *Eine* **einseitige (disjunkte) Zerlegung** *einer Booleschen Funktion* $f \in B_n$ *mit den Eingangsvariablen* $x_1, \ldots, x_p,\ x_{p+1}, \ldots, x_n$ $(0 < p < n)$ *hinsichtlich einer Variablenteilmenge* $\{x_1, \ \ldots, x_p\}$ *ist eine Darstellung von* f *in der Form*

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n).$$

Die einseitige Zerlegung von f läßt sich wie folgt graphisch darstellen:



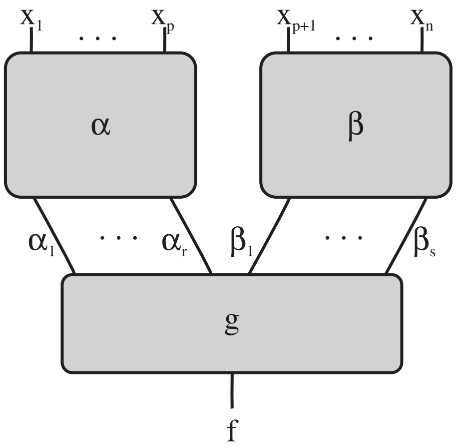
Die Funktionen $\alpha_1, \ldots, \alpha_r$ berechnen auf den Eingangsvariablen $x_1, \ldots, x_p$ ein „Zwischenergebnis“ und die Funktion g berechnet aus diesem Zwischenergebnis und den restlichen Eingangsvariablen $x_{p+1}, \ldots, x_n$ den endgültigen Funktionswert von f auf den Eingangsvariablen $x_1, \ldots, x_p,\ x_{p+1}, \ldots, x_n$.

Definition 3.2 (Zweiseitige Zerlegung)
Eine **zweiseitige (disjunkte) Zerlegung** *einer Booleschen Funktion* $f \in B_n$ *mit den*

Eingangsvariablen $x_1, \ldots, x_p$, $x_{p+1}, \ldots, x_n$ $(0 < p < n)$ *hinsichtlich einer Variablenaufteilung* $\{\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_n\}\}$ *ist eine Darstellung von* f *in der Form*

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) =$$
$$g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), \beta_1(x_{p+1}, \ldots, x_n), \ldots, \beta_s(x_{p+1}, \ldots, x_n))$$

Die zweiseitige Zerlegung wird im folgenden Bild veranschaulicht:



Hier berechnen also die Funktionen $\alpha_1, \ldots, \alpha_r$ auf den Eingangsvariablen $x_1, \ldots,$ x_p ein vorläufiges Ergebnis und die Funktionen $\beta_1, \ldots, \beta_s$ auf den Eingangsvariablen $x_{p+1}, \ldots, x_n$ ein vorläufiges Ergebnis. Die Funktion g berechnet aus diesen beiden Zwischenergebnissen den endgültigen Funktionswert von f.

Bezeichnung 3.1 *Die Funktionen* α_i $(1 \leq i \leq r)$ *(und* β_j $(1 \leq j \leq s)$ *im Falle zweiseitiger Zerlegung) werden als* **Zerlegungsfunktionen** *bezeichnet, die Funktion* g *wird als* **Zusammensetzungsfunktion** *bezeichnet.*
Im Fall von einseitigen Zerlegungen werden die Variablen aus der Menge $\{x_1, \ldots, x_p\}$ *der obigen Definition auch als „gebundene" Variablen, die Variablen aus* $\{x_{p+1}, \ldots, x_n\}$ *als „freie" Variablen bezeichnet.*

Will man Zerlegungen bei der Logiksynthese ausnutzen, so ist es günstig, nach Zerlegungen mit möglichst geringer Anzahl von Zerlegungsfunktionen zu suchen. Dies hat folgende Vorteile:

- Ist die Anzahl der Zerlegungsfunktionen gering, so hat man bei rekursiver Anwendung des Verfahrens weniger (Zerlegungs-)Funktionen zu realisieren. Es besteht die Hoffnung, daß die Gesamtkosten des resultierenden Schaltkreises somit möglichst gering bleiben.

- Je weniger Zerlegungsfunktionen bei einer Zerlegung auftreten, desto weniger Eingänge hat die Zusammensetzungsfunktion g. Wenn g weniger Eingänge hat, dann kann man hoffen, daß die Komplexität von g auch geringer ist.

- Aus der Netzliste (bzw. dem Ω–Schaltkreis), die die Logiksynthese liefert, wird das Layout eines Schaltkreises bestimmt, der die gesuchte Funktion realisiert. Häufig beobachtet man Schaltungen, bei denen die Fläche des Layouts nicht unbedingt durch die Anzahl der zu realisierenden Gatter bestimmt wird, sondern durch die Anzahl der zu realisierenden globalen Verbindungsleitungen. Eine Zerlegung mit wenig Zerlegungsfunktionen legt gerade eine Realisierung mit wenig globalen Verbindungsleitungen nahe.

In diesem Sinne sind solche Zerlegungen interessant, bei denen die Anzahl der Zerlegungsfunktionen kleiner ist als die Anzahl der Eingangsvariablen (im Fall zweiseitiger Zerlegungen) bzw. bei denen die Anzahl der Zerlegungsfunktionen kleiner ist als die Anzahl der Eingangsvariablen, von denen die Zerlegungsfunktionen abhängen (im Fall einseitiger Zerlegungen). Man bezeichnet diese Zerlegungen als *nichttrivial*.

Definition 3.3 (Nichttriviale Zerlegung)

- *Eine einseitige Zerlegung*

  $$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$$

  heißt **nichttrivial**, *wenn* $r < p$.

- *Eine zweiseitige Zerlegung*

  $$\begin{aligned} &f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = \\ &\quad g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), \beta_1(x_{p+1}, \ldots, x_n), \ldots, \beta_s(x_{p+1}, \ldots, x_n)) \end{aligned}$$

  heißt **nichttrivial**, *wenn* $r + s < n$.

Verwendet man nur nichttriviale Zerlegungen von f und wendet man das Verfahren rekursiv an, um eine Realisierung für die Zerlegungsfunktionen und die Zusammensetzungsfunktion zu finden, so haben alle Funktionen, die auf der nächsten Rekursionsstufe behandelt werden, eine geringere Anzahl an Eingangsvariablen als f.

Zerlegungen, die mit einer *minimalen* Anzahl von Zerlegungsfunktionen auskommen, bezeichnet man als „kommunikationsminimal“:

Definition 3.4 (Kommunikationsminimale Zerlegung)

- *Eine einseitige Zerlegung*

$$f(x_1, \dots, x_p, x_{p+1}, \dots, x_n) = g(\alpha_1(x_1, \dots, x_p), \dots, \alpha_r(x_1, \dots, x_p), x_{p+1}, \dots, x_n)$$

heißt **kommunikationsminimal** *hinsichtlich* $\{x_1, \dots, x_p\}$*, falls für alle einseitigen Zerlegungen*

$$f(x_1, \dots, x_p, x_{p+1}, \dots, x_n) = g'(\alpha'_1(x_1, \dots, x_p), \dots, \alpha'_{r'}(x_1, \dots, x_p), x_{p+1}, \dots, x_n)$$

gilt: $r \leq r'$*.*

- *Eine zweiseitige Zerlegung*

$$f(x_1, \dots, x_p, x_{p+1}, \dots, x_n) = \\ g(\alpha_1(x_1, \dots, x_p), \dots, \alpha_r(x_1, \dots, x_p), \beta_1(x_{p+1}, \dots, x_n), \dots, \beta_s(x_{p+1}, \dots, x_n))$$

heißt **kommunikationsminimal** *hinsichtlich* $\{\{x_1, \dots, x_p\}, \{x_{p+1}, \dots, x_n\}\}$ *, falls für alle zweiseitigen Zerlegungen*

$$f(x_1, \dots, x_p, x_{p+1}, \dots, x_n) = \\ g'(\alpha'_1(x_1, \dots, x_p), \dots, \alpha'_{r'}(x_1, \dots, x_p), \beta'_1(x_{p+1}, \dots, x_n), \dots, \beta'_{s'}(x_{p+1}, \dots, x_n))$$

gilt: $r \leq r'$ *und* $s \leq s'$*.*

Im folgenden werden wir uns bei der Logiksynthese mit nichttrivialen *und* kommunikationsminimalen Zerlegungen befassen.

3.1.1 Minimale Anzahl von Zerlegungsfunktionen

Aus den oben schon genannten Gründen ist es interessant, Zerlegungen mit einer möglichst geringen Anzahl von Zerlegungsfunktionen zu betrachten. Gesucht ist nun eine Möglichkeit, eine kommunikationsminimale Zerlegung hinsichtlich einer Variablenteilmenge $X^{(1)}$ zu finden, wenn $X^{(1)}$ schon vorgegeben ist.

Als Hilfsmittel dazu wird der Begriff der *Zerlegungsmatrix* definiert.

Definition 3.5 (Zerlegungsmatrix) *Die Zerlegungsmatrix einer (partiellen) Funktion* $f \in S(D)$ *mit den Eingangsvariablen* $x_1, \dots, x_p,\ x_{p+1}, \dots, x_n$ *hinsichtlich der Variablenteilmenge* $X^{(1)} = \{x_1, \dots, x_p\}$ *ist definiert als* $(2^p \times 2^{n-p})$*–Matrix* $Z(X^{(1)})$*, wobei für alle* i, j *mit* $0 \leq i \leq 2^p - 1$ *und* $0 \leq j \leq 2^{n-p} - 1$ *gilt:*

$$Z(X^{(1)})_{ij} = \begin{cases} f(bin_p(i), bin_{n-p}(j)), & falls\ f\ \text{definiert an dieser Stelle} \\ \star, & falls\ f\ \text{undefiniert an dieser Stelle} \end{cases}$$

(Hierbei liefert $bin_p(i)$ *ein* p*–Tupel über* $\{0, 1\}$*, das der Binärdarstellung von* i *entspricht,* $bin_{n-p}(j)$ *liefert ein* $(n-p)$*–Tupel über* $\{0, 1\}$*, das eine Binärdarstellung von* j *darstellt.)*

In diesem Abschnitt ist zunächst nur von Zerlegungsmatrizen zu totalen Funktionen die Rede. Für totale Funktionen werden folgende Bezeichnungen definiert:

Bezeichnung 3.2 *Sei $f \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_p,\ x_{p+1}, \ldots, x_n$. Sei $Z(X^{(1)})$ die Zerlegungsmatrix von f hinsichtlich der Variablenteilmenge $X^{(1)} := \{x_1, \ldots, x_p\}$. Dann wird die Anzahl der verschiedenen Zeilen von $Z(X^{(1)})$ mit $vz(X^{(1)}, f)$ bezeichnet, die Anzahl der verschiedenen Spalten von $Z(X^{(1)})$ mit $vs(X^{(1)}, f)$.*

Bemerkung 3.1 *Ist $Z(X^{(1)})$ Zerlegungsmatrix der (totalen) Funktion f hinsichtlich der Variablenteilmenge $X^{(1)}$, so liefert die Gleichheit auf den Zeilen von $Z(X^{(1)})$ eine Äquivalenzklasseneinteilung von $\{0,1\}^{|X^{(1)}|}$, wenn man für $x^{(1)}, x^{(2)} \in \{0,1\}^{|X^{(1)}|}$ definiert: $x^{(1)} \equiv x^{(2)}$ genau dann, wenn die Zeilen mit den Indizes $int(x^{(1)})^*$ und $int(x^{(2)})$ gleich sind.*

Bevor die minimale Anzahl von Zerlegungsfunktionen bei einer Zerlegung von f hinsichtlich $X^{(1)}$ bestimmt wird, sollen die Begriffe anhand eines Beispiels verdeutlicht werden:

Beispiel 3.1 Sei die Funktion vgl_4 definiert als

$$vgl_4(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4) = \begin{cases} 1, & falls\ (x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) \\ 0, & falls\ (x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4). \end{cases}$$

vgl_4 vergleicht also die Binärzahl gebildet aus den 4 ersten Eingangsvariablen mit der Binärzahl gebildet aus den 4 letzten Eingangsvariablen.

Eine mögliche zweiseitige Zerlegung hinsichtlich $\{\{x_1, x_2, y_1, y_2\}, \{x_3, x_4,\ y_3, y_4\}\}$ ist

$$vgl_4(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4) = and_2(vgl_2(x_1, x_2, y_1, y_2), vgl_2(x_3, x_4, y_3, y_4))$$

$$\text{mit } vgl_2(x_1, x_2, y_1, y_2) = \begin{cases} 1, & falls\ (x_1, x_2) = (y_1, y_2) \\ 0, & falls\ (x_1, x_2) \neq (y_1, y_2) \end{cases}$$

Auf den Variablen $\{x_1, x_2, y_1, y_2\}$ wird also berechnet, ob die ersten beiden Bits der zu vergleichenden Binärzahlen übereinstimmen oder nicht (2 Informationen, kodiert durch 1 Bit der Zerlegungsfunktion), auf den Variablen $\{x_3, x_4, y_3, y_4\}$ wird entsprechend berechnet, ob die beiden letzten Bits der zu vergleichenden Zahlen übereinstimmen. Die Zusammensetzungsfunktion (and_2) setzt die beiden Zwischenergebnisse zum endgültigen Funktionswert zusammen.

Die Anzahl der Informationen, die durch die Zerlegungsfunktionen auf den Variablen der beiden Mengen berechnet werden müssen, kann man auch aus der Zerlegungsmatrix in Bild 3.1 ablesen.

Die Zerlegungsmatrix hat 2 verschiedene Zeilen. Die 0–Zeilen geben jeweils an, daß $(x_1, x_2) \neq (y_1, y_2)$, die anderen Zeilen geben an, daß $(x_1, x_2) = (y_1, y_2)$. (Analog für die beiden verschiedenen Spalten.)

$^*int(x)$ gibt den dezimalen Wert der Binärzahl x an.

x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_4	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
y_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
y_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$x_1x_2y_1y_2$																
0 0 0 0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
0 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 1 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 1 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 1 0 1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 1 1 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 0 1 0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
1 0 1 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 1 1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Abbildung 3.1: Zerlegungsmatrix zu vgl_4

Zur minimalen Anzahl von Zerlegungsfunktionen gilt allgemein:

Satz 3.1 (Curtis '61) *Sei $f \in B_n$ eine Funktion mit den Eingangsvariablen $x_1, \ldots, x_p$, $x_{p+1}, \ldots, x_n$. Dann gibt es genau dann eine einseitige Zerlegung von f hinsichtlich der Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ der Form*

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n),$$

wenn

$$r \geq \log(vz(X^{(1)}, f)).$$

Beweis:

„$\Longrightarrow$": Sei eine Zerlegung gegeben durch

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) =$$
$$g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$$

Dann kann die Funktion

$$\alpha = (\alpha_1, \ldots, \alpha_r) \in B_{p,r}$$

höchstens 2^r verschiedene Funktionswerte annehmen.
Nehme nun an, daß

$$r < \log(vz(X^{(1)}, f), d.h.\ 2^r < vz(X^{(1)}, f).$$

Dann muß es $\delta^{(1)} = (\delta_1^{(1)}, \ldots, \delta_p^{(1)})$ und $\delta^{(2)} = (\delta_1^{(2)}, \ldots, \delta_p^{(2)})$ geben, so daß

$$\alpha(\delta^{(1)}) = \alpha(\delta^{(2)}),$$

aber die zugehörigen Zerlegungsmatrixzeilen verschieden sind.
Wenn die zugehörigen Zerlegungsmatrixzeilen verschieden sind, so bedeutet dies, daß es $\epsilon^{(0)} = (\epsilon_{p+1}^{(0)}, \ldots, \epsilon_n^{(0)})$ gibt mit

$$f(\delta_1^{(1)}, \ldots, \delta_p^{(1)}, \epsilon_{p+1}^{(0)}, \ldots, \epsilon_n^{(0)}) \neq f(\delta_1^{(2)}, \ldots, \delta_p^{(2)}, \epsilon_{p+1}^{(0)}, \ldots, \epsilon_n^{(0)}).$$

Trotzdem gilt unabhängig von der Wahl von g:

$$g(\alpha_1(\delta_1^{(1)}, \ldots, \delta_p^{(1)}), \ldots, \alpha_r(\delta_1^{(1)}, \ldots, \delta_p^{(1)}), \epsilon_{p+1}^{(0)}, \ldots, \epsilon_n^{(0)}) = \\ g(\alpha_1(\delta_1^{(2)}, \ldots, \delta_p^{(2)}), \ldots, \alpha_r(\delta_1^{(2)}, \ldots, \delta_p^{(2)}), \epsilon_{p+1}^{(0)}, \ldots, \epsilon_n^{(0)}),$$

da $\alpha(\delta^{(1)}) = \alpha(\delta^{(2)})$. $\Longrightarrow$ Widerspruch. Es muß also $r \geq \log(vz(X, f)$ gelten.

„$\Longleftarrow$“: Sei

$$r \geq \log(vz(X, f)), d.h.\ 2^r \geq vz(X, f).$$

Dann kann man $\alpha \in B_{p,r}$ definieren, so daß

$$\alpha(\delta^{(1)}) \neq \alpha(\delta^{(2)})$$

für alle $\delta^{(1)}, \delta^{(2)} \in \{0, 1\}^p$, für die die zugehörigen Zerlegungsmatrixzeilen verschieden sind.
Die Zusammensetzungsfunktion g kann man definieren nach folgender Vorschrift:
Für alle $(x_1, \ldots, x_n) \in \{0, 1\}^n$ ist

$$g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n) = f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n). \quad (\star)$$

Falls $(a_1, \ldots, a_r)$ nicht im Bild von α auftritt, so ist $g(a_1, \ldots, a_r,\ x_{p+1}, \ldots, x_n)$ undefiniert bzw. frei wählbar.

g ist nach Vorschrift $(\star)$ wohldefiniert.
Wäre g nicht wohldefiniert, dann gäbe es $\delta^{(1)}$ und $\delta^{(2)}$ aus $\{0, 1\}^p$ mit

$$\alpha(\delta^{(1)}) = \alpha(\delta^{(2)})$$

und ein $\epsilon^{(0)} \in \{0, 1\}^{n-p}$ mit

$$f(\delta^{(1)}, \epsilon^{(0)}) \neq f(\delta^{(2)}, \epsilon^{(0)}).$$

Dann wären aber die Zerlegungsmatrixzeilen, die zu $\delta^{(1)}$ und $\delta^{(2)}$ gehören verschieden und damit nach Definition von α

$$\alpha(\delta^{(1)}) \neq \alpha(\delta^{(2)}).$$

$\Longrightarrow$ Widerspruch. g muß wohldefiniert sein.

$\square$

Anhand des Beweises zu Satz 3.1 ergibt sich folgende Aussage:

Korollar 3.1 *Sei $f \in B_n$ eine Funktion mit den Eingangsvariablen $x_1, \ldots, x_p$, $x_{p+1}, \ldots$, x_n, seien für $1 \leq i \leq r$ $\alpha_i \in B_p$ und sei $Z(X^{(1)})$ die Zerlegungsmatrix von f hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. Dann gibt es genau dann eine einseitige Zerlegung von f hinsichtlich der Variablenteilmenge $X^{(1)}$ in der Form*

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n),$$

wenn die Funktion $\alpha = (\alpha_1, \ldots, \alpha_r)$ folgende Eigenschaft hat:
Falls es zu $\delta^{(1)}, \delta^{(2)} \in \{0,1\}^p$ ein $\epsilon^{(0)} \in \{0,1\}^{n-p}$ gibt mit

$$f(\delta^{(1)}, \epsilon^{(0)}) \neq f(\delta^{(2)}, \epsilon^{(0)})$$

(d.h. falls die Zeilen $int(\delta^{(1)})$ und $int(\delta^{(2)})$ von $Z(X^{(1)})$ verschieden sind), dann ist

$$\alpha(\delta^{(1)}) \neq \alpha(\delta^{(2)}).$$

Entsprechend gilt für zweiseitige Zerlegungen:

Satz 3.2 *Sei $f \in B_n$ eine Funktion mit den Eingangsvariablen $x_1, \ldots, x_p$, $x_{p+1}, \ldots, x_n$. Sei $X^{(1)} = \{x_1, \ldots, x_p\}$ und $X^{(2)} = \{x_{p+1}, \ldots, x_n\}$. Dann gibt es genau dann eine zweiseitige Zerlegung von f hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$ der Form*

$$\begin{aligned} &f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = \\ &\quad g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), \beta_1(x_{p+1}, \ldots, x_n), \ldots, \beta_s(x_{p+1}, \ldots, x_n)) \end{aligned}$$

wenn

$$r \geq \log(vz(X^{(1)}, f)) \;\; \textit{und} \;\; s \geq \log(vs(X^{(1)}, f)).$$

Beweis:

„$\Longrightarrow$“: Nimmt man

$$r < \log(vz(X^{(1)}, f)) \text{ oder } s < \log(vs(X^{(1)}, f))$$

an, so kann man analog zum Beweis des vorhergehenden Satzes einen Widerspruch herleiten.

„$\Longleftarrow$“: Sei

$$r \geq \log(vz(X^{(1)}, f)), d.h.\ 2^r \geq vz(X^{(1)}, f) \text{ und}$$
$$s \geq \log(vs(X^{(1)}, f)), d.h.\ 2^s \geq vs(X^{(1)}, f).$$

Dann kann man $\alpha \in B_{p,r}$ definieren, so daß

$$\alpha(\delta^{(1)}) \neq \alpha(\delta^{(2)})$$

für alle $\delta^{(1)}, \delta^{(2)} \in \{0,1\}^p$, für die die zugehörigen Zerlegungsmatrixzeilen verschieden sind.
Ebenso kann man $\beta \in B_{q,s}$ definieren, so daß

$$\beta(\epsilon^{(1)}) \neq \beta(\epsilon^{(2)})$$

für alle $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^{n-p}$, für die die zugehörigen Zerlegungsmatrixspalten verschieden sind.
Dann kann man die Zusammensetzungsfunktion g definieren nach der nun folgenden Vorschrift $(\star)$

$$g(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_r(x_1,\ldots,x_p),\beta_1(x_{p+1},\ldots,x_n),\ldots,\beta_s(x_{p+1},\ldots,x_n)) = f(x_1,\ldots,x_p,x_{p+1},\ldots,x_n)$$

für alle $(x_1,\ldots,x_p,x_{p+1},\ldots,x_n) \in \{0,1\}^n$.
Falls $(a_1,\ldots,a_r)$ nicht im Bild von α auftritt oder $(b_1,\ldots,b_s)$ nicht im Bild von β auftritt, so ist $g(a_1,\ldots,a_r,b_1,\ldots,b_s)$ undefiniert bzw. frei wählbar.

g ist nach Vorschrift $(\star)$ wohldefiniert.
Angenommen g wäre nicht wohldefiniert, dann gäbe es $(\delta^{(1)}, \epsilon^{(1)})$ und $(\delta^{(2)}, \epsilon^{(2)})$ aus $\{0,1\}^n$ mit

$$\alpha(\delta^{(1)}) = \alpha(\delta^{(2)}) \text{ und } \beta(\epsilon^{(1)}) = \beta(\epsilon^{(2)}) \qquad (**)$$

und

$$f(\delta^{(1)}, \epsilon^{(1)}) \neq f(\delta^{(2)}, \epsilon^{(2)}).$$

D.h. es würde gelten

$$f(\delta^{(1)}, \epsilon^{(1)}) = \epsilon \in \{0,1\} \text{ und } f(\delta^{(2)}, \epsilon^{(2)}) = \overline{\epsilon}.$$

Es kann nun *nicht* sein, daß *sowohl* die zu $\delta^{(1)}$ und $\delta^{(2)}$ gehörigen Zeilen der Zerlegungsmatrix *als auch* die zu $\epsilon^{(1)}$ und $\epsilon^{(2)}$ gehörigen Spalten gleich sind:
Angenommen die Zeilen zu $\delta^{(1)}$ und $\delta^{(2)}$ sind gleich. Dann gilt

$$f(\delta^{(2)}, \epsilon^{(1)}) = f(\delta^{(1)}, \epsilon^{(1)}) = \epsilon.$$

Wegen

$$f(\delta^{(2)}, \epsilon^{(2)}) = \overline{\epsilon}, \quad \text{also } f(\delta^{(2)}, \epsilon^{(1)}) \neq f(\delta^{(2)}, \epsilon^{(2)}),$$

sind dann die zu $\epsilon^{(1)}$ und $\epsilon^{(2)}$ gehörigen Spalten ungleich.
Analog folgt aus der Tatsache, daß die Spalten zu $\epsilon^{(1)}$ und $\epsilon^{(2)}$ gleich sind:

$$f(\delta^{(1)}, \epsilon^{(2)}) = f(\delta^{(1)}, \epsilon^{(1)}) = \epsilon.$$

und somit

$$f(\delta^{(1)}, \epsilon^{(2)}) \neq f(\delta^{(2)}, \epsilon^{(2)}) = \overline{\epsilon}.$$

Also sind die zu $\delta^{(1)}$ und $\delta^{(2)}$ gehörigen Zeilen ungleich.

Die zu $\delta^{(1)}$ und $\delta^{(2)}$ gehörigen Zeilen oder die zu $\epsilon^{(1)}$ und $\epsilon^{(2)}$ gehörigen Spalten sind ungleich.

Im ersten Fall gilt nach Definition von α

$$\alpha(\delta^{(1)}) \neq \alpha(\delta^{(2)}) \quad \Longrightarrow \text{ Widerspruch zu (**)},$$

im zweiten Fall gilt nach Definition von β

$$\beta(\epsilon^{(1)}) \neq \beta(\epsilon^{(2)}) \quad \Longrightarrow \text{ Widerspruch zu (**)}.$$

Die Annahme, daß g durch Vorschrift $(\star)$ nicht wohldefiniert ist, ist also falsch!

□

Die Sätze 3.1 bzw. 3.2 geben an, wieviel Zerlegungsfunktionen man bei der einseitigen bzw. zweiseitigen Zerlegung mindestens benötigt, wenn die Variablenteilmenge $X^{(1)}$ bzw. Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$ schon vorgegeben ist. Die Anzahl der Zerlegungsfunktionen in einer kommunikationsminimalen Zerlegung ist im allgemeinen natürlich stark abhängig von der Wahl der Variablenteilmenge bzw. Variablenaufteilung, hinsichtlich der zerlegt werden soll.

Beispiel 3.1 (Fortsetzung):
Wählt man beispielsweise für die Funktion vgl_4 aus Beispiel 3.1 die Variablenaufteilung als $\{\{x_1, x_2, x_3, x_4\}, \{y_1, y_2, y_3, y_4\}\}$ statt statt $\{\{x_1, x_2, y_1, y_2\}, \{x_3, x_4, y_3, y_4\}\}$, so benötigt man für die entsprechende zweiseitige Zerlegung jeweils 4 Zerlegungsfunktionen auf $\{x_1, \ldots, x_4\}$ und 4 Zerlegungsfunktionen auf $\{y_1, \ldots, y_4\}$. Die beiden zu vergleichenden 4–Bit–Zahlen befinden sich dann in getrennten Variablenteilmengen und durch die Zerlegungsfunktionen kann keine sinnvolle Vorberechnung durchgeführt werden. Eine mögliche Wahl der Zerlegungsfunktionen wäre

$$\alpha_i(x_1, \ldots, x_4) = x_i \quad \text{und} \quad \beta_i(y_1, \ldots, y_4) = y_i$$

für $1 \leq i \leq 4$, so daß sich als Zusammensetzungsfunktion wieder vgl_4 selbst ergeben würde. Diesen Sachverhalt kann man auch anhand der Zerlegungsmatrix zu dieser Zerlegung erkennen:

y_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
y_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
y_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
y_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$x_1x_2x_3x_4$																
0 0 0 0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 0 1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 1 0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 1 1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0 1 0 0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0 1 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0 1 1 0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0 1 1 1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1 0 0 0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1 0 0 1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1 0 1 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1 1 0 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1 1 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1 1 1 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Die Zerlegungsmatrix ist eine Einheitsmatrix mit 16 Zeilen und 16 Spalten. Also benötigt man 4 Zerlegungsfunktionen auf $\{x_1, \ldots, x_4\}$ und 4 Zerlegungsfunktionen auf $\{y_1, \ldots, y_4\}$.

3.1.2 Allgemeine Untersuchungen zur nichttrivialen Zerlegbarkeit Boolescher Funktionen

Im vorliegenden Abschnitt soll die nichttriviale Zerlegbarkeit Boolescher Funktionen allgemein untersucht werden. Dabei genügt es im wesentlichen, sich auf nichttriviale einseitige Zerlegungen zu beschränken, da aus der Definition nichttrivialer Zerlegbarkeit direkt folgt: Gibt es zu einer Funktion $f \in B_n$ mit der Menge von Eingangsvariablen $X^{(1)} \cup X^{(2)}$ ($X^{(1)} \cap X^{(2)} = \emptyset$) eine nichttriviale einseitige Zerlegung hinsichtlich der Variablenteilmenge $X^{(1)}$, so gibt es auch eine nichttriviale zweiseitige Zerlegung hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$. Gibt es zu f eine nichttriviale zweiseitige Zerlegung hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$, so gibt es auch eine nichttriviale einseitige Zerlegung hinsichtlich $X^{(1)}$ oder eine nichttriviale einseitige Zerlegung hinsichtlich $X^{(2)}$.

Aus den Sätzen 3.1 und 3.2 ergibt sich direkt folgendes Korollar:

Korollar 3.2 *$f \in B_n$ ist genau dann nichttrivial hinsichtlich der Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ zerlegbar, wenn*

$$vz(X^{(1)}, f) \leq 2^{p-1}.$$

$f \in B_n$ ist genau dann nichttrivial hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\} = \{\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_n\}\}$ zerlegbar, wenn

$$vz(X^{(1)}, f) \le 2^{p-1} \quad \textit{oder} \quad vs(X^{(1)}, f) \le 2^{(n-p)-1}.$$

Das folgende Lemma 3.1 besagt, daß es leichter ist, nichttriviale einseitige Zerlegungen hinsichtlich großer Variablenteilmengen zu finden als nichttriviale einseitige Zerlegungen hinsichtlich kleiner Variablenteilmengen.

Lemma 3.1 *Gibt es keine nichttriviale einseitige Zerlegung von $f \in B_n$ hinsichtlich der Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$, so gibt es auch keine nichttriviale einseitige Zerlegung hinsichtlich Teilmengen von $X^{(1)}$.*
bzw.:
Ist f nichttrivial zerlegbar hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$, so ist f nichttrivial zerlegbar hinsichtlich jeder Obermenge von $X^{(1)}$.

Beweis: Sei $f \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_n$ nichttrivial zerlegbar hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$, d.h. $vz(X^{(1)}, f) \le 2^{p-1}$.
Es genügt nun zu zeigen, daß f hinsichtlich $X^{(1)'} = \{x_1, \ldots, x_{p+1}\}$ nichttrivial zerlegbar ist, d.h. daß $vz(X^{(1)'}, f) \le 2^p$.
Sei $Z(X^{(1)})$ die Zerlegungsmatrix hinsichtlich $X^{(1)}$, $Z(X^{(1)'})$ die Zerlegungsmatrix hinsichtlich $X^{(1)'}$.
Sind in $Z(X^{(1)})$ die Zeilen zu $x^{(1)} = (x_1^{(1)}, \ldots, x_p^{(1)})$ und $x^{(2)} = (x_1^{(2)}, \ldots, x_p^{(2)})$ gleich, so gilt

$$f(x^{(1)}, y^{(0)}) = f(x^{(2)}, y^{(0)})$$

für alle $y^{(0)} \in \{0,1\}^{n-p}$ und somit

$$f(x^{(1)}, y_1^{(1)}, y^{(0)'}) = f(x^{(2)}, y_1^{(1)}, y^{(0)'})$$

für alle $y_1^{(1)} \in \{0,1\}$, $y^{(0)'} \in \{0,1\}^{n-p-1}$.
Also sind in $Z(X^{(1)'})$ die Zeilen zu

$$(x_1^{(1)}, \ldots, x_p^{(1)}, 0) \text{ und } (x_1^{(2)}, \ldots, x_p^{(2)}, 0)$$

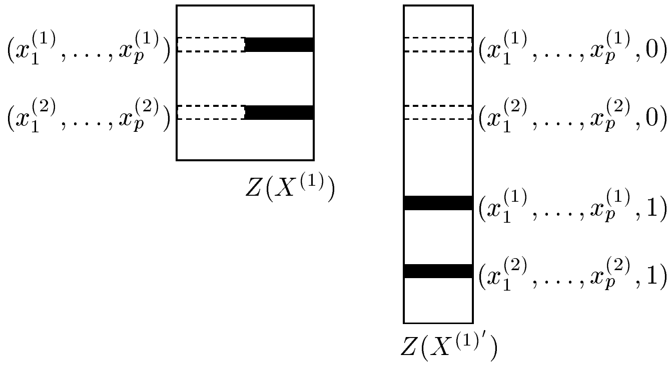
und die Zeilen zu

$$(x_1^{(1)}, \ldots, x_p^{(1)}, 1) \text{ und } (x_1^{(2)}, \ldots, x_p^{(2)}, 1)$$

gleich.
$Z(X^{(1)'})$ kann also maximal doppelt soviele verschiedene Zeilen haben wie $Z(X^{(1)})$, d.h.

$$vz(X^{(1)'}, f) \le 2vz(X^{(1)}, f) \le 2^p.$$

Der Zusammenhang zwischen den Zerlegungsmatrizen $Z(X^{(1)})$ und $Z(X^{(1)'})$ ist in folgender Abbildung dargestellt:



□

Es gilt sogar, daß jede Funktion $f \in B_n$ $(n \geq 4)$ nichttrivial zerlegbar ist, wenn man die Variablenteilmenge nur groß genug wählt:

Lemma 3.2 *Zu jeder Funktion $f \in B_n$ existiert eine nichttriviale einseitige Zerlegung*

$$f(x_1,\ldots,x_p,x_{p+1},\ldots,x_n) = g(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_r(x_1,\ldots,x_p),x_{p+1},\ldots,x_n),$$

falls $p \geq 2^{n-p}+1$.

Beweis:

$$p \geq 2^{n-p}+1 \quad \Longleftrightarrow \quad 2^{p-1} \geq 2^{2^{n-p}}$$

Die Zeilen der Zerlegungsmatrix hinsichtlich $X^{(1)} = \{x_1,\ldots,x_p\}$ haben Länge 2^{n-p}. Es gibt genau $2^{2^{n-p}}$ verschiedene Elemente in $\{0,1\}^{2^{n-p}}$.

$\Longrightarrow$ Es kann höchstens $2^{2^{n-p}}$ paarweise verschiedene Zeilen geben.

$\Longrightarrow$ $vz(X^{(1)},f) \leq 2^{2^{n-p}}$

$\Longrightarrow$ $vz(X^{(1)},f) \leq 2^{2^{n-p}} \leq 2^{p-1}$

$\Longrightarrow$ f nichttrivial zerlegbar hinsichtlich $X^{(1)}$. □

Korollar 3.3 *Ist $f \in B_n$ mit $n \geq 4$, so gibt es immer eine nichttriviale einseitige Zerlegung.*

Bemerkung 3.2 *Für Funktionen $f \in B_n$ mit $n \geq 4$ und den Eingangsvariablen $\{x_1,\ldots,x_n\}$ bildet die sogenannte* Shannon–Zerlegung *eine spezielle nichttriviale (nicht unbedingt immer kommunikationsminimale) Zerlegung hinsichtlich einer Variablenteilmenge $\{x_1,\ldots,x_{n-1}\}$ (denn es gilt offensichtlich $n-1 \geq 2^1+1$).*

Die Shannon–Zerlegung von f hinsichtlich x_n ist definiert als

$$f(x_1,\ldots,x_n) = g(\alpha_1(x_1,\ldots,x_{n-1}),\alpha_2(x_1,\ldots,x_{n-1}),x_n),$$

wobei

$$\begin{aligned}\alpha_1(x_1,\ldots,x_{n-1}) &= f(x_1,\ldots,x_{n-1},0)\\ \alpha_2(x_1,\ldots,x_{n-1}) &= f(x_1,\ldots,x_{n-1},1)\\ g(a_1,a_2,x_n) &= a_1\cdot\overline{x_n}+a_2\cdot x_n.\end{aligned}$$

Alle Booleschen Funktionen mit $n \geq 4$ Eingangsvariablen sind also nichttrivial zerlegbar (z.B. hinsichtlich einer Variablenteilmenge der Mächtigkeit $p = n-1$). Wählt man allerdings kleinere Variablenteilmengen $X^{(1)}$, hinsichtlich derer f zerlegt werden soll, so wird die nichttriviale Zerlegbarkeit einer Funktion $f \in B_n$ zu einer speziellen Struktureigenschaft der Funktion, die gewährleistet, daß die Funktion eine wesentlich geringere Komplexität hat als die durch Shannon festgestellte Mindestkomplexität $\frac{2^n}{n}$, die fast alle zufällig gewählten Funktionen überschreiten. Dies ergibt sich aus folgendem Lemma:

Lemma 3.3 *Besitzt eine Funktion $f \in B_n$ eine nichttriviale einseitige Zerlegung hinsichtlich der Variablenteilmenge $\{x_1,\ldots,x_p\}$, so hat jede Realisierung von f, die diese Zerlegung ausnutzt, eine B_2–Komplexität $< \frac{2^n}{n}$, falls n und p hinreichend groß sind und*

$$p \leq (n-1) - \log\frac{n(n-1)}{n-2} - \epsilon \text{ für } \epsilon > 0.$$

Beweis: Sei nach Voraussetzung $p \leq (n-1) - \log\frac{n(n-1)}{n-2} - \epsilon$.
Dann gilt mit genügend kleinem $c > 1$:

$$p \leq (n-1) - \log\frac{cn(n-1)}{(2-c)n-2}.$$

Diese Ungleichung läßt sich äquivalent umformen:

$$\begin{aligned}&\iff n-p \geq [\log\tfrac{cn(n-1)}{(2-c)n-2}]+1\\ &\iff 2^{n-p-1} \geq \tfrac{cn(n-1)}{(2-c)n-2}\\ &\iff cn(n-1) \leq 2^{n-p-1}(2n-2-cn)\\ &\iff c \leq \tfrac{2^{n-p}(n-1)-2^{n-p-1}cn}{n(n-1)}\\ &\iff c(1+\tfrac{2^{n-p-1}}{n-1}) \leq \tfrac{2^{n-p}}{n} \quad (\star)\end{aligned}$$

(Voraussetzung $p \leq (n-1) - \log\frac{n(n-1)}{n-2} - \epsilon$ wird an späterer Stelle in Form von Ungleichung $(\star)$ ausgenutzt.)

Sei eine nichttriviale Zerlegung der Form

$$f(x_1,\ldots,x_p,x_{p+1},\ldots,x_n) = g(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_r(x_1,\ldots,x_p),x_{p+1},\ldots,x_n)$$

gegeben.

Für die Komplexität der Zerlegungsfunktionen gilt nach dem Satz von Lupanov:

$$C_{B_2}(\alpha_i) \leq \frac{2^p}{p} + o\left(\frac{2^p}{p}\right)$$

Für die Komplexität der Zusammensetzungsfunktion gilt:

$$C_{B_2}(g) \leq \frac{2^{r+n-p}}{r+n-p} + o\left(\frac{2^{r+n-p}}{r+n-p}\right) \leq \frac{2^{n-1}}{n-1} + o\left(\frac{2^{n-1}}{n-1}\right)$$

Sind p und n hinreichend groß, so gilt:

$$C_{B_2}(\alpha_i) \leq \frac{2^p}{p} + o\left(\frac{2^p}{p}\right) \leq c\frac{2^p}{p}$$

und

$$C_{B_2}(g) \leq \frac{2^{n-1}}{n-1} + o\left(\frac{2^{n-1}}{n-1}\right) \leq c\frac{2^{n-1}}{n-1}.$$

Die Gesamtkosten eines Schaltkreises S, der die gegebene Zerlegung ausnutzt, lassen sich dann abschätzen durch

$$\begin{aligned}
C_{B_2}(S) &\leq rc\frac{2^p}{p} + c\frac{2^{n-1}}{n-1} \\
&\leq c\left[(p-1)\frac{2^p}{p} + \frac{2^{n-1}}{n-1}\right] \\
&= c2^p\left[\frac{p-1}{p} + \frac{2^{n-p-1}}{n-1}\right] \\
&< 2^p c\left[1 + \frac{2^{n-p-1}}{n-1}\right] \\
&\leq 2^p\frac{2^{n-p}}{n} \qquad \text{wegen } (\star) \\
&= \frac{2^n}{n}
\end{aligned}$$

□

Aufgrund des Shannon'schen Abzählargumentes und Lemma 3.3 ist es also unwahrscheinlich, daß eine zufällig gewählte Funktion eine solche nichttriviale Zerlegung (hinsichtlich einer kleinen Variablenteilmenge) besitzt. Wie in Abschnitt 1.2 schon angedeutet, hat man es in der Praxis aber nicht mit zufällig gewählten Funktionen zu tun (vgl. auch Kapitel 4).

Für zweiseitige Zerlegungen folgt aus Lemma 3.1, daß die Wahrscheinlichkeit dafür, daß es eine nichttriviale Zerlegung gibt, für solche Variablenaufteilungen am geringsten ist, bei denen in jeder der beiden Mengen ungefähr gleich viele Variablen vertreten sind. Solche Zerlegungen heißen gleichmächtig:

Definition 3.6 (Gleichmächtige Zerlegungen)
Eine zweiseitige Zerlegung von $f \in B_n$ hinsichtlich der Variablenaufteilung $\{\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_n\}\}$ heißt **gleichmächtig**, *falls*

$$p = \lfloor \frac{n}{2} \rfloor \;\; \textit{oder} \;\; p = \lceil \frac{n}{2} \rceil.$$

Gleichmächtige Zerlegungen sind deshalb interessant, weil die rekursive Ausnutzung gleichmächtiger Zerlegungen mit minimaler Anzahl von Zerlegungsfunktionen häufig zu Realisierungen mit geringer Tiefe führt.

Will man nun eine Logiksynthese unter Ausnutzung gleichmächtiger Zerlegungen durchführen und stellt es sich heraus, daß es keine nichttriviale gleichmächtige Zerlegung gibt, so ist es nach Lemma 3.2 durchaus sinnvoll, einen Zerlegungsschritt einzuschieben, bei dem die Aufteilung der Variablen in zwei verschiedenmächtige Mengen erfolgt. Im nächsten Rekursionsschritt können dann wieder gleichmächtige Zerlegungen möglich sein. Dies soll anhand eines Beispiels verdeutlicht werden:

Beispiel 3.2 Gegeben sei eine Funktion $f \in B_5$, die definiert ist durch

$$f(x_0, \ldots, x_4) = \overline{x_0} \cdot (x_1 \oplus x_2) \cdot (x_3 \oplus x_4) \; + \; x_0 \cdot (x_1 + x_3) \cdot (x_2 + x_4)$$

für alle $(x_0, \ldots, x_4) \in \{0,1\}^5$.

Man sieht leicht (z.B. durch Betrachtung aller Zerlegungsmatrizen zu den $\binom{5}{3}$ möglichen gleichmächtigen Variablenaufteilungen), daß es zu f keine nichttriviale gleichmächtige Zerlegung gibt.

Führt man jedoch eine Shannon–Zerlegung durch, so sind die erhaltenen Zerlegungsfunktionen nichttrivial gleichmächtig zerlegbar:

$$f(x_0, \ldots, x_4) = \overline{x_0} \cdot \alpha_1(x_1, \ldots, x_4) \; + \; x_0 \cdot \alpha_2(x_1, \ldots, x_4)$$

mit den Zerlegungsfunktionen α_1 und α_2 mit

$$\alpha_1(x_1, \ldots, x_4) = (x_1 \oplus x_2) \cdot (x_3 \oplus x_4) \text{ und}$$

$$\alpha_2(x_1, \ldots, x_4) = (x_1 + x_3) \cdot (x_2 + x_4).$$

Zu α_1 gibt es eine nichttriviale gleichmächtige Zerlegung hinsichtlich der Variablenaufteilung $\{\{x_1, x_2\}, \{x_3, x_4\}\}$ und zu α_2 gibt es eine nichttriviale gleichmäßige Zerlegung hinsichtlich der Variablenaufteilung $\{\{x_1, x_3\}, \{x_2, x_4\}\}$. Der resultierende Schaltkreis ist in Abbildung 3.2 angegeben.

Zerlegungen erweisen sich als sehr hilfreich bei der Logiksynthese (vgl. auch Kapitel 4). Auf der anderen Seite ist aber auch für Boolesche Funktionen mit *ausgezeichneten* Zerlegbarkeitseigenschaften nicht garantiert, daß die Ausnutzung von Zerlegungen notwendigerweise zu optimalen Realisierungen führen muß. Einen Beleg dafür liefert das folgende Lemma, das aus einem Satz von W. J. Paul [Pau76] hervorgeht (Beweis siehe [SM93]):

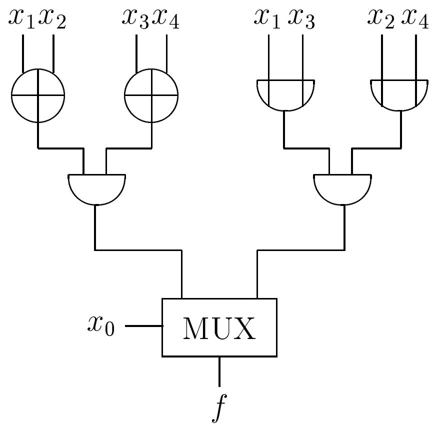


Abbildung 3.2: Schaltkreis zu Beispielfunktion f

Lemma 3.4 *Zu jedem $\epsilon > 0$ gibt es ein $n \in \mathbf{N}$ und eine Boolesche Funktion $G \in B_{2n}$ mit folgender Eigenschaft:*

Es gibt eine kommunikationsminimale gleichmächtige Zerlegung von G hinsichtlich einer Variablenaufteilung $A = \{\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_{2n}\}\}$ mit genau 2 Zerlegungsfunktionen und es gibt keine Zerlegung mit weniger Zerlegungsfunktionen.

Für die Kosten jeder Realisierung S, die eine kommunikationsminimale gleichmächtige Zerlegung hinsichtlich A ausnutzt, gilt

$$C_{B_2}(S) \geq \frac{2}{1+\epsilon} C_{B_2}(G) + 1.$$

3.1.3 Zerlegung von Funktionen, die durch ROBDDs repräsentiert sind

Funktionstabellen und Zerlegungsmatrizen haben für alle Booleschen Funktionen aus B_n exponentielle Größe. Wie in Kapitel 1 schon angesprochen wurde, gibt es Datenstrukturen, die wesentlich kompaktere Repräsentationen Boolescher Funktionen liefern — zwar nicht im Mittel über alle möglichen Booleschen Funktionen aus B_n, aber doch für sehr viele für die Praxis relevante Funktionen. ROBDDs [Bry86] stellen eine solche Datenstruktur dar, die aus diesem Grund in den letzten Jahren immer größere Verbreitung gefunden hat.

Daß es sich auf jeden Fall lohnt, sich mit Zerlegungen von Funktionen in ROBDD–Darstellung zu befassen, zeigt (im Vorgriff auf Kapitel 4) Tabelle 3.1. Es werden exemplarisch für Addierer mit verschiedenen Bitbreiten Laufzeiten des auf Grundlage dieser Arbeit implementierten Logiksynthesewerkzeuges angegeben. Die erste Spalte zeigt die Laufzeiten für eine frühere Version, die auf der Verarbeitung von Funktionstabellen und Zerlegungsmatrizen beruhte, die zweite Spalte Laufzeiten der aktuellen ROBDD–basierten Version. Die sehr viel

Schaltkreis	Laufzeit der Tabellenversion	Laufzeit der ROBDD–Version
$adder_4$	1,3 s	0,2 s
$adder_8$	18 h 38 min	1,15 s
$adder_{16}$	—	11,55 s
$adder_{32}$	—	3 min 8 s
$adder_{64}$	—	31 min 16 s

Tabelle 3.1: Laufzeiten für die Synthese von n–Bit–Addierern $adder_n$.

höheren Laufzeiten der tabellenbasierten Version lassen sich einfach dadurch erklären, daß wesentlich größere Datenmengen zu bearbeiten sind. Wie man leicht nachrechnet, würde beim 16–Bit–Addierer die Größe einer Funktionstabelle (bei 32 Eingängen, 16 Ausgängen und einem Bit pro Tabelleneintrag) schon 8 Gigabyte betragen, beim 32–Bit–Addierer ca. 69 Milliarden Gigabyte. Allein wegen der Größe der benötigten Repräsentation können Probleme dieser Größenordnung also von der tabellenbasierten Version nicht mehr bearbeitet werden.

Ziel ist es nun, ausgehend von ROBDDs Zerlegungen Boolescher Funktionen durchführen zu können, ohne den Umweg über Funktionstabellen oder Zerlegungsmatrizen gehen zu müssen.

3.1.3.1 Minimale Anzahl von Zerlegungsfunktionen

Das erste Problem, das sich hierbei stellt, ist die Bestimmung der Anzahl von benötigten Zerlegungsfunktionen bei vorgegebener Variablenteilmenge $X^{(1)}$, hinsichtlich der man eine Zerlegung durchführen will. Wie im vorangegangenen Abschnitt schon beschrieben wurde, ist dies für eine Funktion $f \in B_n$ prinzipiell mit Hilfe einer Zerlegungsmatrix $Z(X^{(1)})$ für f machbar. Allerdings kann man die Anzahl der verschiedenen Zeilen von $Z(X^{(1)})$ auch direkt anhand des ROBDD zu f ablesen.

Die entscheidende Beobachtung hierbei ist, daß bei $X^{(1)} = \{x_{index(1)}, \ldots, x_{index(p)}\}$ und

$$\epsilon_{index} = \left(\epsilon_{index(1)}, \ldots, \epsilon_{index(p)}\right) \in \{0,1\}^p$$

die Zeile von $Z(X^{(1)})$ mit Index $int(\epsilon_{index})^2$ gerade die Funktionstabelle des Kofaktors $f_{x_{index(1)}^{\epsilon_{index(1)}} \ldots x_{index(p)}^{\epsilon_{index(p)}}}$ von f darstellt[3].

[2] $int(\epsilon_{index})$ gibt den dezimalen Wert der Binärzahl ϵ_{index} an.

[3] Da ein Kofaktor von $f \in B_n$ hinsichtlich $x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}$ nicht von $x_1, \ldots, x_p$ abhängt, wird er im folgenden zumeist als Funktion aus B_{n-p} interpretiert.

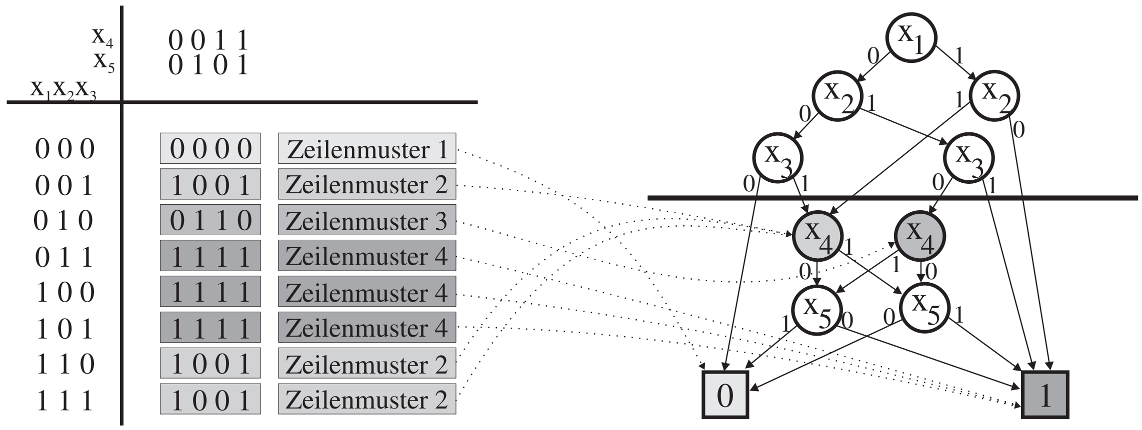


Abbildung 3.3: Zerlegungsmatrix und ROBDD einer Beispielfunktion aus B_5. Die Zerlegung erfolgt hinsichtlich der Variablenmenge $\{x_1, x_2, x_3\}$. Es gibt genau 4 verschiedene Zeilenmuster und daher auch genau 4 verschiedene „Verbindungsknoten“ (grau gefärbt) unterhalb der Schnittlinie nach den ersten 3 Variablen im ROBDD.

Die Aufgabe, die Anzahl der Zeilen von $Z(X^{(1)})$ mit verschiedenem Zeilenmuster zu berechnen, ist also gleichbedeutend mit der Aufgabe, die Anzahl der verschiedenen Kofaktoren

$$f_{x_{index(1)}^{\epsilon_{index(1)}} \dots x_{index(p)}^{\epsilon_{index(p)}}} \text{ mit } (\epsilon_{index(1)}, \dots, \epsilon_{index(p)}) \in \{0,1\}^p$$

zu bestimmen.

Wie schon in Kapitel 1 bemerkt wurde (Bemerkung 1.5, Seite 27), ist es bei einem ROBDD $F = (G, m)$ mit Variablenordnung $<_{index}$ leicht zu testen, ob die Kofaktoren

$$f_{x_{index(1)}^{\epsilon_{index(1)}} \dots x_{index(p)}^{\epsilon_{index(p)}}} \text{ und } f_{x_{index(1)}^{\delta_{index(1)}} \dots x_{index(p)}^{\delta_{index(p)}}}$$

gleich sind. Sie sind genau dann gleich, wenn die durch ϵ_{index} bzw. δ_{index} erreichbaren Knoten von G identisch sind. $vz(X^{(1)}, f)$ ist also gleich der Anzahl der verschiedenen durch p–Tupel $\epsilon_{index} \in \{0,1\}^p$ erreichbaren Knoten des ROBDD.

In Abbildung 3.3 wird der Zusammenhang illustriert zwischen verschiedenen Zeilenmustern in der Zerlegungsmatrix und ROBDD-Knoten, die Wurzeln von Teil-ROBDDs sind, die die entsprechenden Kofaktoren repräsentieren. Diese Knoten, die in eineindeutiger Weise den zugehörigen Zeilenmustern entsprechen, werden im folgenden auch als „Verbindungsknoten“ bezeichnet. Die Bezeichnung ergibt sich daraus, daß diese Knoten im ROBDD direkt unterhalb einer gedachten Schnittlinie nach den ersten p Variablen (den Variablen aus der Variablenmenge $X^{(1)}$) liegen. (Es handelt sich genau um die Knoten, die unterhalb dieser Schnittlinie liegen, aber mit mindestens einem Knoten verbunden sind, der oberhalb der Schnittlinie liegt.)

$vz(X^{(1)}, f)$ läßt sich demnach mit einem einfachen Algorithmus bestimmen:
Zu Beginn wird der Zähler vz auf 0 gesetzt. Man durchläuft dann G nach einer Tiefensuchstrategie. Abweichend von der üblichen Tiefensuche hört man nicht erst dann auf, in die Tiefe zu laufen, wenn man an einem Blatt ankommt, sondern dann, wenn man an einem Knoten v ankommt, der als Markierung eine Variable $x_{index(l)}$ mit $x_{index(p)} <_{index} x_{index(l)}$ (bzw. $l > p$) trägt oder 0 bzw. 1 als Markierung trägt. Wurde v vorher noch nicht besucht, so wird vz um 1 erhöht.

Ist dieses Verfahren beendet, so liefert der Stand des Zählers vz die Anzahl der verschiedenen durch p–Tupel $\epsilon_{index} \in \{0,1\}^p$ erreichbaren Knoten von G und damit den gesuchten Wert $vz(X^{(1)}, f)$. Man kann $vz(X^{(1)}, f)$ also mit einer Laufzeit bestimmen, die linear in der Größe von G ist.

Das gerade beschriebene Verfahren funktioniert allerdings nur dann, wenn die Variablenteilmenge $X^{(1)}$ gerade aus den p Variablen besteht, die die ersten in der Ordnung $<_{index}$ sind. Ist dies nicht der Fall, so muß man die Variablenordnung ändern[4]. Ein Verfahren, das eine geeignete Variablenteilmenge $X^{(1)}$ und eine dazugehörige Variablenordnung bestimmt, ist in Abschnitt 3.1.4 beschrieben.

3.1.3.2 Bestimmung von Zerlegungs– und Zusammensetzungsfunktionen

Hat man festgestellt, wieviele Zerlegungsfunktionen man bei einer kommunikationsminimalen Zerlegung hinsichtlich einer Variablenteilmenge $X^{(1)}$ benötigt, so bleibt das Problem, eine zugehörige Zerlegung (mit Zerlegungsfunktionen und Zusammensetzungsfunktion) zu finden.

Im Beweis zu den Satz 3.1 ist grundsätzlich angegeben, wie man entsprechende Zerlegungs- und Zusammensetzungsfunktionen (für einseitige Zerlegungen) finden kann: Geht man von der Zerlegungsmatrix $Z(X^{(1)})$ aus, so müssen die Zerlegungsfunktionen $(\alpha_1, \ldots, \alpha_r)$ so gewählt werden, daß sie den Indizes von Zeilen mit verschiedenen Zeilenmustern auch verschiedene Zerlegungsfunktionswerte zuweisen. (Ansonsten hat man bei der Wahl der Zerlegungsfunktionen im allgemeinen aber noch Freiheiten. Abschnitt 3.4 befaßt sich unter anderem mit der Ausnutzung dieser Freiheiten.) Die Zusammensetzungsfunktion g in der Zerlegung $f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$ ergibt sich dann automatisch aus der Forderung, daß für alle $\epsilon \in \{0,1\}^n$ $g(\alpha_1(\epsilon_1, \ldots, \epsilon_p), \ldots, \alpha_r(\epsilon_1, \ldots, \epsilon_p), \epsilon_{p+1}, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_n)$. (Abgesehen von Stellen $(a_1, \ldots, a_r, \epsilon_{p+1}, \ldots, \epsilon_n) \in \{0,1\}^n$, für die $(a_1, \ldots, a_r)$ nicht im Bildbereich von $(\alpha_1, \ldots, \alpha_r)$ liegt, so daß der Funktionswert von g an dieser Stelle beliebig gewählt werden kann. Dies führt dann wiederum zu Freiheitsgraden, deren Ausnutzung in den Abschnitten 3.2 und 3.5.2 betrachtet wird.)

Beispiel 3.3 Abbildung 3.4 zeigt eine Zerlegung der Funktion $f(x_1, \ldots, x_4) = \overline{x_4} x_2 x_3 + x_4(x_1 \oplus x_2)$ hinsichtlich $\{x_1, x_2, x_3\}$. Die zugehörige Zerlegungsmatrix hat genau 4 verschie-

[4]Dabei kann sich allerdings auch die Größe des ROBDD ändern.

$f(x_1, x_2, x_3, x_4) = \overline{x}_4\, x_2\, x_3 + x_4\, (x_1 \oplus x_2)$

$x_1x_2x_3$ \ x_4	0 1	
0 0 0	0 0	Zeilenmuster 1
0 0 1	0 0	Zeilenmuster 1
0 1 0	0 1	Zeilenmuster 2
0 1 1	1 1	Zeilenmuster 4
1 0 0	0 1	Zeilenmuster 2
1 0 1	0 1	Zeilenmuster 2
1 1 0	0 0	Zeilenmuster 1
1 1 1	1 0	Zeilenmuster 3

Codierung

Zerlegungsfunktionen:

$x_1x_2x_3$	α_1	α_2
0 0 0	0	0
0 0 1	0	0
0 1 0	0	1
0 1 1	1	1
1 0 0	0	1
1 0 1	0	1
1 1 0	0	0
1 1 1	1	0

Zusammensetzungsfunktion:

$a_1a_2x_4$	g
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

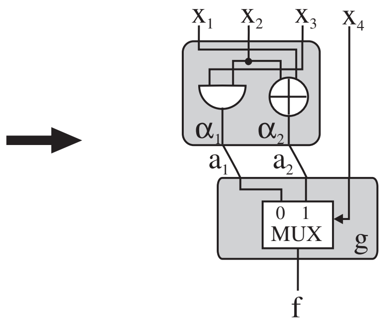


Abbildung 3.4: Zerlegung der Funktion $f(x_1, \ldots, x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$ hinsichtlich $\{x_1, x_2, x_3\}$. Es sind 2 Zerlegungsfunktionen α_1 und α_2, die Zusammensetzungsfunktion g und darunter der resultierende Schaltkreis angegeben.

dene Zeilenmuster, so daß für eine kommunikationsminimale Zerlegung 2 Zerlegungsfunktionen α_1 und α_2 benötigt werden. Der Funktionswert $(0, 0)$ von (α_1, α_2) tritt nur für die Indizes der Zerlegungsmatrixzeilen mit Zeilenmuster 1 auf, $(0, 1)$ nur für Zeilen mit Muster 2 usw. Die Zusammensetzungsfunktion ergibt sich aus der Wahl der Zerlegungsfunktionen α_1 und α_2.

Wie weiter oben schon bemerkt wurde, liefert bei der Zerlegung einer Funktion f hinsichtlich der Variablenteilmenge $X^{(1)}$ die Gleichheit auf den Zeilen der Zerlegungsmatrix $Z(X^{(1)})$ eine Äquivalenzrelation auf $\{0, 1\}^{|X^{(1)}|}$, wenn man für $x^{(1)}, x^{(2)} \in \{0, 1\}^{|X^{(1)}|}$ definiert: $x^{(1)} \equiv x^{(2)}$ genau dann, wenn die Zeilen mit den Indizes $int(x^{(1)})$ und $int(x^{(2)})$ gleich sind. Die Menge der Äquivalenzklassen $\{K_1, \ldots, K_{vz(X^{(1)},f)}\}$ dieser Relation wird im folgenden mit $\{0, 1\}^{|X^{(1)}|}/_{\equiv}$ bezeichnet. Die Bedingung aus Korollar 3.1 besagt, daß es bei der Zerlegung $f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$ kein $\epsilon \in K_i$ und $\delta \in K_j$ mit $i \neq j$ geben darf mit $(\alpha_1, \ldots, \alpha_r)(\epsilon) = (\alpha_1, \ldots, \alpha_r)(\delta)$ (*). Ist die Anzahl der verschiedenen Zeilenmuster $vz(X^{(1)}, f)$ keine Zweierpotenz, so kann es auch bei kommunikationsminimalen Zerlegungen mit $\lceil \log(vz(X^{(1)}, f)) \rceil$ Zerlegungsfunktionen durchaus vorkommen, daß es $\epsilon \neq \delta \in K_i$ gibt mit $(\alpha_1, \ldots, \alpha_r)(\epsilon) \neq (\alpha_1, \ldots, \alpha_r)(\delta)$, ohne daß Bedingung (*) verletzt wird.

Beispiel 3.4 Eine Funktion $f \in B_n$ werde hinsichtlich $X^{(1)} = \{x_1, x_2, x_3\}$ zerlegt. Es gelte $\{0,1\}^{|X^{(1)}|}/_\equiv = \{K_1, K_2, K_3\}$ Außerdem sei $K_1 = \{(000)\}$, $K_2 = \{(001), (010), (011), (100), (101), (110)\}$, $K_3 = \{(111)\}$.
Für diese Zerlegung werden dann 2 Zerlegungsfunktionen α_1 und α_2 benötigt.
Die Wahl von $\alpha = (\alpha_1, \alpha_2)$ mit $\alpha(K_1) = (0,0)$, $\alpha(\{(001), (010), (011)\} = (01)$, $\alpha(\{(100), (101), (110)\}) = (10)$ und $\alpha(K_3) = (11)$ erfüllt offensichtlich die Bedingung aus Korollar 3.1, so daß α_1 und α_2 als Zerlegungsfunktionen in einer Zerlegung von f hinsichtlich $X^{(1)}$ verwendet werden können. α weist Elementen aus *verschiedenen* Klassen K_i *verschiedene* Werte zu, aber nicht allen Elementen aus K_2 den gleichen Wert.

In Beispiel 3.3 werden dagegen allen Elementen der gleichen Äquivalenzklasse gleiche Zerlegungsfunktionswerte zugeordnet:

Beispiel 3.3 (Fortsetzung):
Es gilt $K_1 = \{(000), (001), (110)\}$, $K_2 = \{(010), (100), (101)\}$ $K_3 = \{(111)\}$ und $K_4 = \{(011)\}$. Weiter gilt $\alpha(K_1) = (0,0)$, $\alpha(K_2) = (0,1)$, $\alpha(K_3) = (1,0)$ und $\alpha(K_4) = (1,1)$.

Zerlegungen mit der Eigenschaft aus Beispiel 3.3 werden als *strikt* bezeichnet:

Definition 3.7 (Strikte Zerlegungen)
Sei $f \in B_n$. $f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$ eine Zerlegung von f hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. $\{0,1\}^p/_\equiv = \{K_1, \ldots, K_{vz(X^{(1)},f)}\}$ sei die durch die Zeilengleichheit in $Z(X^{(1)})$ induzierte Äquivalenzklasseneinteilung von $\{0,1\}^p$. Eine Zerlegungsfunktion α_i heißt strikt, *falls für alle $\epsilon, \delta \in K_j$ $(1 \leq j \leq vz(X^{(1)}, f))$ gilt: $\alpha_i(\epsilon) = \alpha_i(\delta)$. Die Zerlegung heißt* strikt, *falls alle Zerlegungsfunktionen α_i $(1 \leq i \leq r)$ strikt sind.*

Eine strikte Zerlegung läßt sich als eine Kodierung der Äquivalenzklassen $K_i \in \{0,1\}^p/_\equiv$ durch α auffassen. Jeder Äquivalenzklasse wird ein fester Code zugewiesen.

Lemma 3.5
Sei $f \in B_n$. $f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$ eine Zerlegung von f hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. $\{0,1\}^p/_\equiv = \{K_1, \ldots, K_{vz(X^{(1)},f)}\}$ sei die durch die Zeilengleichheit in $Z(X^{(1)})$ induzierte Äquivalenzklasseneinteilung von $\{0,1\}^p$. Ist $vz(X^{(1)}, f)$ eine Zweierpotenz, so ist jede kommunikationsminimale Zerlegung von f hinsichtlich $X^{(1)}$ auch eine strikte Zerlegung.

Beweis: Sei $vz(X^{(1)}, f) = 2^r$, $\{0,1\}^p/_\equiv = \{K_1, \ldots, K_{2^r}\}$.
Nach Satz 3.1 ist dann die Anzahl der Zerlegungsfunktionen in einer kommunikationsminimalen Zerlegung gleich r.

Korollar 3.1 besagt, daß es kein $\epsilon \in K_i$ und $\delta \in K_j$ mit $i \neq j$ geben darf mit $(\alpha_1, \ldots, \alpha_r)(\epsilon) = (\alpha_1, \ldots, \alpha_r)(\delta)$. Gäbe es ein K_i $(1 \leq i \leq 2^r)$ mit $\epsilon^{(1)} \neq \epsilon^{(2)} \in K_i$, $\alpha(\epsilon^{(1)}) \neq \alpha(\epsilon^{(2)})$, so wäre die Anzahl der verschiedenen Funktionswerte von α größer als 2^r. Offensichtlich kann es aber nur 2^r verschiedene Werte im Bildbereich von $\alpha = (\alpha_1, \ldots, \alpha_r)$ geben. □

Beschränkt man sich auf strikte Zerlegungen, so ist man in der Lage, ausgehend von ROBDDs in einfacher Weise auch Zerlegungs– und Zusammensetzungsfunktionen zu bestimmen, ohne den Umweg über Funktionstabellen oder Zerlegungsmatrizen gehen zu müssen. Hier nutzt man wie schon in Abschnitt 3.1.3.1 die Tatsache aus, daß zu jedem Zeilenmuster m_i in der Zerlegungsmatrix $Z(X^{(1)})$ einer Funktion f ein eindeutiger „Verbindungsknoten" n_i im ROBDD zu f existiert, so daß der Teilbaum, dessen Wurzel n_i ist, gerade die Funktion repräsentiert, deren Funktionstabelle Zeilenmuster m_i angibt. (Nehme o.B.d.A. an, daß $X^{(1)} = \{x_1, \ldots, x_p\}$ und daß die Variablenordnung des ROBDD durch $x_1 <_{index} \cdots <_{index} x_n$ gegeben ist.) Die Menge aller Zeilenindizes von $Z(X^{(1)})$ mit Zeilenmuster m_i ist durch eine Äquivalenzklasse K_i aus $\{0,1\}^p/_\equiv$ gegeben. Dann ist die Menge aller $(\epsilon_1, \ldots, \epsilon_p) \in \{0,1\}^p$, durch die der Verbindungsknoten n_i erreichbar ist, gerade die Menge K_i. Ersetzt man den Knoten n_i durch die Konstante 1 und alle anderen Verbindungsknoten durch die Konstante 0, so erhält man also einen (evtl. nicht reduzierten) OBDD für die charakteristische Funktion von K_i. [5]

Eine strikte Zerlegungsfunktion α_j ist durch die Funktionswerte w_j^i auf den Äquivalenzklassen K_i bestimmt $(\alpha_j(K_i) = \{w_j^i\})$. Um einen OBDD für α_j zu erhalten, genügt es also, im ursprünglichen ROBDD die „Verbindungsknoten" n_i jeweils durch die Konstante 0 zu ersetzen, falls $w_j^i = 0$ bzw. durch die Konstante 1, falls $w_j^i = 1$. Man erhält so einen (evtl. nicht reduzierten) OBDD $A_j = (G = (V, E), m)$ für α_j, den man in Zeit $O(|V| \cdot \log(|V|))$ zu einem ROBDD reduzieren kann. Aus den Codierungen $(w_1^i, \ldots, w_r^i)$ für die Verbindungsknoten n_i (bzw. für die Äquivalenzklassen K_i) lassen sich also in einfacher Weise ROBDDs für die Zerlegungsfunktionen $\alpha_1, \ldots, \alpha_r$ berechnen.

Auch der ROBDD für die Zusammensetzungsfunktion g läßt sich einfach bestimmen: g hängt von den neuen Variablen $a_1, \ldots, a_r$ und den Variablen $x_{p+1}, \ldots, x_n$ ab. Der ROBDD für g wird mit der Variablenordnung $a_1 <_{index} \cdots <_{index} a_r <_{index} x_{p+1} <_{index} \cdots <_{index} x_n$ konstruiert. Ist der durch $(\epsilon_1, \ldots, \epsilon_p)$ erreichbare Knoten im ursprünglichen ROBDD für f der Verbindungsknoten n_i, so liefere $\alpha = (\alpha_1, \ldots, \alpha_r)$ auf der Eingabe $(\epsilon_1, \ldots, \epsilon_p) \in K_i$ den Wert $(w_1^i, \ldots, w_r^i)$. Wegen

$$\begin{aligned} f(\epsilon_1, \ldots, \epsilon_p, \delta_{p+1}, \ldots, \delta_n) &= g(\alpha_1(\epsilon_1, \ldots, \epsilon_p), \ldots, \alpha_r(\epsilon_1, \ldots, \epsilon_p), \delta_{p+1}, \ldots, \delta_n) \\ &= g(w_1^i, \ldots, w_r^i, \delta_{p+1}, \ldots, \delta_n) \end{aligned}$$

$\forall(\delta_{p+1}, \ldots, \delta_n) \in \{0,1\}^{n-p}$ müssen die Kofaktoren $f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}$ und $g_{a_1^{w_1^i} \ldots a_r^{w_r^i}}$ von f und g identisch sein. Folglich muß man im ROBDD für g nach Verfolgen des Pfades (von der Wurzel aus) gemäß der Belegung $(w_1^i, \ldots, w_r^i)$ zu dem Teilbaum gelangen, dessen Wurzel n_i ist

[5] D.h. für die Funktion $c \in B_p$ mit $c(\epsilon_1, \ldots, \epsilon_p) = 1$, falls $(\epsilon_1, \ldots, \epsilon_p) \in K_i$, $c(\epsilon_1, \ldots, \epsilon_p) = 0$ sonst.

und somit den Kofaktor $f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}} = g_{a_1^{w_1^i}\ldots a_r^{w_r^i}}$ repräsentiert. Man erhält also einen OBDD für g, indem man den „unteren Teil" des ROBDD zu f mit den Teilgraphen, deren Wurzeln die Verbindungsknoten sind, übernimmt und den „oberen Teil" mit den Variablen $a_1, \ldots a_r$ so konstruiert, daß durch $(w_1^i, \ldots, w_r^i)$ die Verbindungsknoten n_i erreicht werden $(1 \leq i \leq vz(\{x_1, \ldots, x_p\}, f))$. Man erreicht dies, indem man einen vollständigen balancierten binären Baum der Tiefe r konstruiert und den Knoten mit Tiefe 0 mit a_1 beschriftet, die Knoten mit Tiefe 1 mit a_2 usw., die Knoten mit Tiefe $r-1$ mit a_r. Die beiden von einem Knoten ausgehenden Kanten werden durch 0 bzw. 1 beschriftet. Ein Knoten mit Tiefe r wird durch den Verbindungsknoten n_i ersetzt, falls er durch $(w_1^i, \ldots, w_r^i)$ erreichbar ist. Im Fall einer kommunikationsminimalen Zerlegung bei der die Anzahl der verschiedenen Verbindungsknoten (bzw. Zeilenmuster der Zerlegungsmatrix) eine Zweierpotenz ist (d.h. $vz(\{x_1, \ldots, x_p\}, f) = 2^r$) hat man nun schon einen reduzierten OBDD bestimmt, der g repräsentiert, da die Anzahl der Blätter des binären Baumes genau gleich der Anzahl der Verbindungsknoten ist und der ROBDD zu f reduziert war. Falls die Anzahl der Verbindungsknoten keine Zweierpotenz ist, gibt es noch Knoten mit Tiefe r, die nicht durch Verbindungsknoten ersetzt worden sind. Es handelt sich genau um die Knoten, die durch Vektoren $(\delta_1, \ldots, \delta_r)$ erreicht werden, die nicht im Bild von $\alpha = (\alpha_1, \ldots, \alpha_r)$ auftreten. Diese Knoten kann man durch beliebige Knoten ersetzen (z.B. durch die Konstante 0). Danach ist dann evtl. noch ein Reduzieren des Funktionsgraphen nötig.

Beispiel 3.5 illustriert die Bestimmung von Zerlegungs– und Zusammensetzungsfunktionen auf ROBDD–Basis:

Beispiel 3.5 Abbildung 3.5 zeigt die Zerlegung der Funktion $f(x_1, \ldots, x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$ aus Beispiel 3.3 hinsichtlich $\{x_1, x_2, x_3\}$. Es gibt 4 verschiedene Verbindungsknoten n_1 bis n_4 (von links nach rechts), so daß für eine kommunikationsminimale Zerlegung 2 Zerlegungsfunktionen α_1 und α_2 benötigt werden. Die zugehörigen Äquivalenzklassen K_1, K_2, K_3 bzw. K_4 werden durch $(0,0)$, $(0,1)$, $(1,0)$ bzw. $(1,1)$ kodiert. Also erhält man einen OBDD für α_1 durch Ersetzung von n_1 durch das erste Codebit von $(0,0)$, von n_2 durch das erste Codebit von $(0,1)$, von n_3 durch das erste Codebit von $(1,0)$ und von n_4 durch das erste Codebit von $(1,1)$. (Analog erhält man einen OBDD für α_2 durch Ersetzungen durch die zweiten Codebits.) Um einen ROBDD für α_1 bzw. α_2 zu erhalten, muß man die OBDDs noch reduzieren. Der ROBDD der Zusammensetzungsfunktion g ergibt sich, indem man den Teil oberhalb der Schnittlinie entfernt und einen binären Baum so darüberlegt, daß durch $(0,0)$ Knoten n_1 erreicht wird, durch $(0,1)$ n_2 erreicht wird usw..

Die Berechnung von Zerlegungs– und Zusammensetzungsfunktionen im Fall von zweiseitigen Zerlegungen läßt sich auf die Berechnung für einseitige Zerlegungen zurückführen. In Abschnitt 3.3 wird dies in allgemeinerer Form für mehrseitige Zerlegungen gezeigt.

3.1.3.3 Vorteil *strikter* Zerlegungen

Im vorangegangenen Abschnitt wurde deutlich, daß man bei Beschränkung auf *strikte* Zerlegungen in der Lage ist, ausgehend von ROBDDs relativ einfach Zerlegungs– und Zusam-

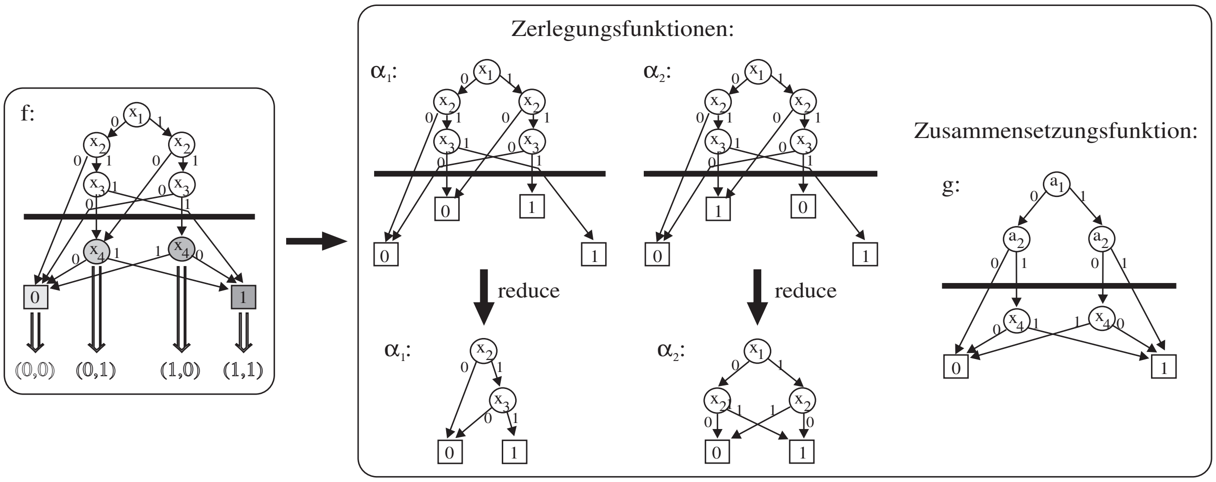


Abbildung 3.5: Zerlegung der Funktion $f(x_1, \ldots, x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$ hinsichtlich $\{x_1, x_2, x_3\}$ ausgehend von der ROBDD–Darstellung.

mensetzungsfunktionen zu bestimmen, ohne Funktionstabellen oder Zerlegungsmatrizen aufstellen zu müssen. Daß wir uns im folgenden auf strikte Zerlegungen beschränken, ist allerdings nicht nur auf diese eher „technische" Begründung zurückzuführen. Experimentelle Untersuchungen haben gezeigt, daß man bei der Beschränkung auf strikte Zerlegungen i.a. einfachere Zerlegungsfunktionen erhält als bei nicht–strikten Zerlegungen. Der Ursache dafür liegt darin, daß Boolesche Funktionen, die in der Praxis vorkommen, i.a. gewisse Struktureigenschaften aufweisen (vgl. Abschnitt 1.2), die man bei der Logiksynthese ausnutzen kann, um zu günstigen Realisierungen zu kommen. Strikte Zerlegungsfunktionen erhalten solche Struktureigenschaften in gewisser Weise. So sind z.B. bei einer Booleschen Funktion, die hinsichtlich einer Variablenteilmenge $\{x_1, \ldots, x_p\}$ zerlegt wird und unabhängig von einer Variablen $x_i \in \{x_1, \ldots, x_p\}$ ist, sämtliche strikte Zerlegungsfunktionen ebenfalls unabhängig von x_i. Falls die Funktion symmetrisch ist in einem Variablenpaar $x_i, x_j \in \{x_1, \ldots, x_p\}$, dann sind auch alle strikten Zerlegungsfunktionen symmetrisch in x_i, x_j. Allgemein gilt der folgende Zusammenhang:

Satz 3.3 *Ist $f \in B_n$ invariant unter $eb_{p,n}(G)$, $G \subseteq \mathbf{S}_{\{0,1\}^p}$, so ist jede strikte Zerlegungsfunktion einer Zerlegung von f hinsichtlich $\{x_1, \ldots, x_p\}$ invariant unter G. $eb_{p,n}(G)$ ist definiert wie in der folgenden Bezeichnung 3.3.*

Bezeichnung 3.3 *Jede Permutation τ aus $\mathbf{S}_{\{0,1\}^p}$ läßt sich in einfacher Weise in $\mathbf{S}_{\{0,1\}^n}$ $(n > p)$ einbetten, indem man die Einbettung $eb_{p,n}(\tau)$ definiert als:*

$$eb_{p,n}(\tau)(\epsilon_1, \ldots, \epsilon_n) = (\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_n)$$

$$\mathit{mit}\ (\delta_1, \ldots, \delta_p) = \tau(\epsilon_1, \ldots, \epsilon_p)\ \forall (\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n.$$

Untergruppen G von $\mathbf{S}_{\{0,1\}^p}$ lassen sich damit in $\mathbf{S}_{\{0,1\}^n}$ einbetten durch

$$eb_{p,n}(G) = \{eb_{p,n}(\tau) \mid \tau \in G\}.$$

Beweis: Sei $\tau \in G$ beliebig. Es ist zu zeigen, daß jede strikte Zerlegungsfunktion α_i einer Zerlegung von f hinsichtlich $\{x_1, \ldots, x_p\}$ invariant ist unter τ.
Sei $(\epsilon_1, \ldots, \epsilon_p) \in \{0,1\}^p$ beliebig und sei $(\delta_1, \ldots, \delta_p) = \tau(\epsilon_1, \ldots, \epsilon_p)$.
Dann gilt nach Voraussetzung für alle $(\gamma_{p+1}, \ldots, \gamma_n) \in \{0,1\}^{n-p}$

$$f(\epsilon_1, \ldots, \epsilon_p, \gamma_{p+1}, \ldots, \gamma_n) = f(\delta_1, \ldots, \delta_p, \gamma_{p+1}, \ldots, \gamma_n) = f(eb_{p,n}(\tau)(\epsilon_1, \ldots, \epsilon_p, \gamma_{p+1}, \ldots, \gamma_n)),$$

da f invariant ist unter $eb_{p,n}(\tau)$.
Folglich ist $(\epsilon_1, \ldots, \epsilon_p) \equiv (\delta_1, \ldots, \delta_p)$, wobei $\equiv$ die Äquivalenzrelation ist, die durch die Gleichheit der Zeilenmustern von $Z(\{x_1, \ldots, x_p\})$ auf $\{0,1\}^p$ erzeugt wird.
Nach der Definition strikter Zerlegungsfunktionen gilt also für jede strikte Zerlegungsfunktion α_i einer Zerlegung von f hinsichtlich $\{x_1, \ldots, x_p\}$

$$\alpha_i(\epsilon_1, \ldots, \epsilon_p) = \alpha_i(\delta_1, \ldots, \delta_p) = \alpha_i(\tau(\epsilon_1, \ldots, \epsilon_p)).$$

Da $(\epsilon_1, \ldots, \epsilon_p) \in \{0,1\}^p$ beliebig gewählt worden war, ist α_i invariant unter τ. □

Satz 3.3 wurde für allgemeine Untergruppen $G \subseteq \mathbf{S}_{\{0,1\}^p}$ bewiesen. Dann gilt er folglich auch für speziellere Untergruppen $G \subseteq \mathbf{P}_p \subseteq \mathbf{S}_{\{0,1\}^p}$, wobei (wie in Abschnitt 2.1) $\mathbf{P}_p$ die Untergruppe von $\mathbf{S}_{\{0,1\}^p}$ ist, die durch die Permutationen $\sigma_{ik}(1 \leq i < k \leq p)$ und $\nu_i(1 \leq i \leq p)$ erzeugt wird mit $\forall \alpha \in \{0,1\}^p$ $\sigma_{ik}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_k, \ldots, \alpha_p) = (\alpha_1, \ldots, \alpha_k, \ldots, \alpha_i, \ldots, \alpha_p)$ und $\nu_i(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_p) = (\alpha_1, \ldots, \overline{\alpha_i}, \ldots, \alpha_p)$.
Es gilt also folgendes Korollar:

Korollar 3.4 *Sei $f \in B_n$ $eb_{p,n}(G)$–symmetrisch für $G \subseteq \mathbf{P}_p$. Dann ist jede strikte Zerlegungsfunktion einer Zerlegung von f hinsichtlich $\{x_1, \ldots, x_p\}$ G–symmetrisch für $G \subseteq \mathbf{P}_p$. Insbesondere übertragen sich Eigenschaften wie Unabhängigkeit von Variablen $x_i \in \{x_1, \ldots, x_p\}$ oder Symmetrie in Variablenpaaren $x_i, x_j \in \{x_1, \ldots, x_p\}$ von f auf die strikten Zerlegungsfunktionen von f.*

3.1.3.4 Zusammenhang zwischen ROBDD–Größe und Zerlegbarkeit

In Abschnitt 3.1.3.1 wurde der Zusammenhang zwischen der Anzahl der „Verbindungsknoten" im ROBDD (bei Schnitt nach den Variablen $x_1, \ldots, x_p$) und der Anzahl der benötigten Zerlegungsfunktionen bei Zerlegung der Funktion hinsichtlich der Variablenteilmenge $\{x_1, \ldots, x_p\}$ angegeben. Daraus wird deutlich, daß bei großer Variablenanzahl n einer Funktion $f \in B_n$ f gut zerlegbar sein muß, falls f kompakt als ROBDD darstellbar ist. Bei hinreichend großem n wird man nur noch Funktionen f darstellen können, die ROBDD–Darstellungen besitzen, deren Größe beschränkt ist durch ein Polynom in n mit kleinem Grad k. Ist die Anzahl der ROBDD–Knoten von $f \in B_n$ beschränkt durch $p(n) = \sum_{i=0}^{k} a_i n^i$, so ist die Anzahl der Verbindungsknoten beschränkt durch $\frac{1}{2}(p(n)+1)$. (Dies ergibt sich aus der Tatsache, daß jeder mit einer Variable beschriftete Knoten im ROBDD Ausgangsgrad 2 hat. Gibt es k Verbindungsknoten, so muß es mindestens $k-1$ Knoten geben, die mit den Variablen $x_1, \ldots, x_p$ beschriftet sind.) Die Anzahl der benötigten Zerlegungsfunktionen ist dann beschränkt durch

$$\begin{aligned} \lceil \log(\frac{1}{2}(p(n)+1)) \rceil &\leq \log(p(n)) \\ &\leq \log(a_{max} \cdot \frac{n^{k+1}-1}{n-1}) \text{ mit } a_{max} = \max_{0 \leq i \leq k} a_i \\ &< \log(a_{max}) + (k+1)\log(n) - \log(n-1) \end{aligned}$$

Zerlegt man hinsichtlich der Variablenteilmenge $\{x_1, \ldots, x_p\}$ mit z.B. $p = \lceil \frac{n}{2} \rceil$, so ist bei genügend großem n (und bei genügend kleinem Grad k des Polynoms und genügend kleinen Konstanten a_i) die Obergrenze $\log(a_{max}) + (k+1)\log(n) - \log(n-1)$ für die Anzahl der Zerlegungsfunktionen sehr viel kleiner als $p = \lceil \frac{n}{2} \rceil$, so daß es sich auf jeden Fall um eine nichttriviale Zerlegung handelt.

Insofern erweist es sich auf jeden Fall als eine gute Heuristik, für die Durchführung von Zerlegungen nach möglichst kleinen ROBDD–Darstellungen zu suchen. Bei totalen Funktionen kommt hierfür eine Optimierung der Variablenordnung des ROBDDs in Betracht, bei partiellen Funktionen zusätzlich noch die Belegung von don't cares.

Allerdings läßt sich der Umkehrschluß nicht allgemein zeigen:
Es lassen sich nämlich durchaus Familien von Funktionen konstruieren, die gute Zerlegbarkeitseigenschaften haben, aber keine kompakte ROBDD–Darstellung besitzen. (Es ist zu beachten, daß „gute Zerlegbarkeitseigenschaften" sich hier aber ausschließlich auf die Funktionen selbst bezieht, nicht notwendigerweise auf Zerlegungs– und Zusammensetzungsfunktionen im Hinblick auf eine rekursive Anwendung des Zerlegungsverfahrens.)
Für die Konstruktion kann man Lemma 3.6 benutzen, das sich (ohne Ausnutzung spezieller Eigenschaften von ROBDDs) leicht ausgehend von Shannons Abzählargument (vergleiche Abschnitt 1.2) herleiten läßt:

Lemma 3.6 *Für hinreichend große n ist die* ROBDD*–Größe (d.h. die Anzahl der Knoten im* ROBDD*) für „fast alle" Funktionen $f \in B_n$ — unabhängig von der Variablenordnung des* ROBDD *— größer als $\frac{2^n}{3n}$.*

Beweis: (Skizze)
Satz 1.2 von Shannon besagt, daß für hinreichend große n fast alle Funktionen $f \in B_n$ eine B_2–Komplexität größer als $\frac{2^n}{n}$ besitzen. Ein ROBDD für eine Funktion f läßt sich als Multiplexer–Schaltkreis interpretieren, der f realisiert, indem man die mit Variablen beschrifteten Knoten durch Multiplexer ersetzt, deren Select–Eingang mit der entsprechenden Eingangsvariablen verbunden ist (siehe Abbildung 3.6). Jeder Multiplexer läßt sich mit 3 2–Input–Zellen aus B_2 realisieren. □

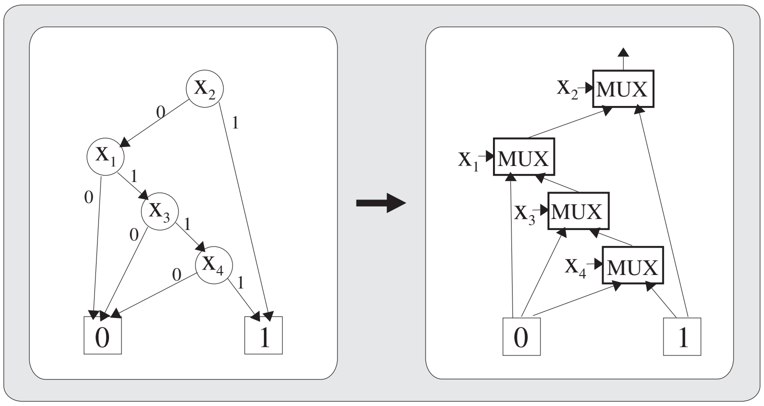


Abbildung 3.6: ROBDD für $f(x_1, \ldots, x_4) = x_2 + x_1 x_3 x_4$ und dazugehöriger Multiplexer–Schaltkreis.

Daß es Funktionen f aus B_n gibt, die gute Zerlegbarkeitseigenschaften haben, aber keine kompakte ROBDD–Darstellung besitzen, zeigt nun das folgende Lemma:

Lemma 3.7 *Für genügend große n gibt es Funktionen aus B_{2n}, deren* ROBDD*–Größe unabhängig von der Variablenordnung des* ROBDD *größer als $\frac{2^n}{3n}$ ist, die aber eine Zerlegung der Form*

$$f(x_1, \ldots, x_n, y_1, \ldots, y_n) = g(\alpha_1(x_1, \ldots, x_n), y_1, \ldots, y_n)$$

(mit nur einer Zerlegungsfunktion) besitzen.

Beweis: Sei $f_1 \in B_n$ eine Funktion, deren ROBDD unabhängig von der Variablenordnung eine Größe größer als $\frac{2^n}{3n}$ aufweist. Seien f_2 und f_3 2 unterschiedliche Funktionen aus B_n, so daß es o.B.d.A. ein $(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ gibt, so daß $f_2(\epsilon_1, \ldots, \epsilon_n) = 0$ und $f_3(\epsilon_1, \ldots, \epsilon_n) = 1$.
Die Funktion $f \in B_{2n}$, die definiert ist durch

$$f(x_1, \ldots, x_n, y_1, \ldots, y_n) = \overline{f_1(x_1, \ldots, x_n)} \cdot f_2(y_1, \ldots, y_n) + f_1(x_1, \ldots, x_n) \cdot f_3(y_1, \ldots, y_n)$$

erfüllt dann die im Lemma geforderten Eigenschaften.

1. Die ROBDD–Größe eines ROBDD für f ist unabhängig von der Variablenordnung größer als $\frac{2^n}{3n}$:
   Für den Kofaktor $f_{y_1^{\epsilon_1}, \ldots, y_n^{\epsilon_n}}$ gilt:

   $$f_{y_1^{\epsilon_1}, \ldots, y_n^{\epsilon_n}} = f_1.$$

   Der ROBDD zu f_1 hat unabhängig von der Variablenordnung mehr als $\frac{2^n}{3n}$ Knoten. Da der ROBDD eines Kofaktors einer Funktion f bei gleicher Variablenordnung nie größer sein kann als der ROBDD zu f[‖], hat auch der ROBDD zu f unabhängig von der Variablenordnung mehr als $\frac{2^n}{3n}$ Knoten.

2. Es ist klar, daß f die im Lemma geforderte Zerlegung besitzt. Wähle z.B.

   - $\alpha_1 = f_1$ und
   - $g \in B_{n+1}$ mit $g(a, y_1, \ldots, y_n) = mux(a, f_2(y_1, \ldots, y_n), f_3(y_1, \ldots, y_n))$, $mux \in B_3$,

$$mux(s, x_1, x_2) = \begin{cases} x_1, & falls\ s = 0 \\ x_2, & falls\ s = 1. \end{cases}$$

□

3.1.4 Bestimmung geeigneter Inputpartitionierungen

Wie schon anhand von Beispiel 3.1 (Seite 78) deutlich wurde, ist die Anzahl der Zerlegungsfunktionen in einer kommunikationsminimalen Zerlegung stark abhängig von der Wahl der Variablenteilmenge bzw. Variablenaufteilung, hinsichtlich der zerlegt werden soll. Ziel dieses Abschnittes ist es, eine Variablenteilmenge bzw. Variablenaufteilung zu finden, bei der die Anzahl der Zerlegungsfunktionen möglichst klein ist.

Ist bei einseitiger Zerlegung einer Funktion $f \in B_n$ die Mächtigkeit p der Variablenteilmenge vorgegeben und ist $\binom{n}{p}$ nicht zu groß, so kann man für alle $\binom{n}{p}$ p–elementigen Teilmengen $X^{(1)}$ der Variablenmenge X die Anzahl $\lceil \log(vz(X^{(1)}, f)) \rceil$ der benötigten Zerlegungsfunktionen bestimmen und dann die Teilmenge $X^{(1)}$ mit der geringsten Anzahl von Zerlegungsfunktionen auswählen.

Ist f durch eine Funktionstabelle gegeben, so stellt man dazu für alle p–elementigen Teilmengen $X^{(1)}$ die Zerlegungsmatrix $Z(X^{(1)})$ auf und bestimmt die Anzahl der verschiedenen Zeilenmuster (durch Sortieren der 2^p Zeilen der Länge 2^{n-p} und anschließendes lineares Durchmustern der Zeilen). Für eine einzelne Teilmenge $X^{(1)}$ benötigt man dazu Laufzeit $O(p2^p2^{n-p}) = O(p2^n) = O(n2^n)$. Man führt das Verfahren $\binom{n}{p} = O\left(\binom{n}{n/2}\right) = O\left(\frac{2^n}{\sqrt{n}}\right)$ Mal durch, so daß die Gesamtlaufzeit zur Bestimmung einer geeigneten Variablenteilmenge $O(\sqrt{n} \cdot 2^{2n})$ beträgt. Drückt man die Laufzeit in der Länge der Eingabe $N = 2^n$ (für die Funktionstabelle bzw. die Zerlegungsmatrix) aus, so ergibt sich eine Laufzeit der Größenordnung $O(N^2\sqrt{\log N})$.

Ist f durch einen ROBDD F gegeben, so bestimmt man $vz(X^{(1)}, f)$ für die Menge $X^{(1)}$ der ersten p Variablen in der Ordnung des ROBDD (wie in Abschnitt 3.1.3.1 angegeben)

[‖]Man erhält einen OBDD eines Kofaktors $f_{x_i^{\epsilon_i}}$ aus dem ROBDD zu f, indem man Knoten mit Markierung x_i löscht und beim Löschen eines Knotens n sämtliche eingehenden Kanten von n mit dem ϵ_i–Sohn von n verbindet. Danach muß man evtl. den OBDD noch reduzieren, um einen ROBDD zu erhalten. Auf jeden Fall erhält man einen ROBDD, der nicht größer ist als der ROBDD zu f.

durch Zählen der Verbindungsknoten unterhalb einer Schnittlinie nach den ersten p Variablen im ROBDD. (Die Verbindungsknoten kann man durch einen Tiefensuchlauf im ROBDD in Linearzeit (in der Größe des ROBDD) bestimmen.) Will man für andere p–elementige Teilmengen $Y^{(1)}$ der Variablenmenge X $vz(Y^{(1)}, f)$ bestimmen, so muß man die Variablenordnung des ROBDD so ändern, daß die Variablen in $Y^{(1)}$ in der Variablenordnung vor allen anderen Variablen stehen. Um $vz(Y^{(1)}, f)$ für alle p–elementigen Teilmengen $Y^{(1)}$ von X zu bestimmen, erzeugt man ausgehend von $F_1 = F$ durch jeweils eine Vertauschung eines Variablenpaares in der Ordnung des aktuellen ROBDDs F_i eine Folge von $\binom{n}{p}$ verschiedenen ROBDDs F_1 bis $F_{\binom{n}{p}}$, bei denen die Mengen $X_i^{(1)}$ der ersten p Variablen in der Ordnung von F_i alle paarweise verschieden sind. Die gesuchte Variablenteilmenge ist dann die Menge $X_i^{(1)}$, bei der $vz(X_i^{(1)}, f)$ am kleinsten ist. Um in der Ordnung von F_i ein Paar von Variablen $x_{index(k)}$ und $x_{index(j)}$ zu vertauschen, gibt es verschiedene Möglichkeiten:

- Man bestimmt unter Anwendung von ROBDD–Operationen den ROBDD F_i' zu der Funktion

$$\begin{aligned} f' \;=\; & \overline{x_{index(k)}}\,\overline{x_{index(j)}} \cdot f_{\overline{x_{index(k)}}\,\overline{x_{index(j)}}} \\ & + \overline{x_{index(k)}} x_{index(j)} \cdot f_{x_{index(k)}\overline{x_{index(j)}}} + x_{index(k)}\overline{x_{index(j)}} \cdot f_{\overline{x_{index(k)}} x_{index(j)}} \\ & + x_{index(k)} x_{index(j)} \cdot f_{x_{index(k)} x_{index(j)}}. \end{aligned}$$

  Danach beschriftet man alle Knoten, die mit $x_{index(k)}$ beschriftet waren mit $x_{index(j)}$ und umgekehrt. So erhält man einen ROBDD F_{i+1} für f mit der gewünschten Variablenordnung.

- Man führt die Variablenvertauschung auf wiederholte Vertauschung benachbarter Variablen zurück. Das Vertauschen benachbarter Variablen $x_{index(k)}$ und $x_{index(k+1)}$ ist eine lokale Operation (d.h. betrifft nur die ROBDD–Knoten, die mit $x_{index(k)}$ und $x_{index(k+1)}$ beschriftet sind) und kann einfach durchgeführt werden [Rud93].

- Man führt die Variablenvertauschung auf eine Verschiebung der Variablen $x_{index(k)}$ und eine Verschiebung der Variablen $x_{index(j)}$ zurück. Die beiden Variablenverschiebungen können mit Hilfe der von Bollig/Wegener [BLM95] eingeführten *jump*–Operationen durchgeführt werden.

Sind n bzw. p sehr groß, so ist allerdings nicht zu erwarten, daß man in der Lage ist, alle $\binom{n}{p}$ benötigten ROBDDs zu erzeugen, um eine geeignete Variablenteilmenge zu bestimmen. Man wird sich mit einer heuristischen Bestimmung einer guten Variablenteilmenge zufrieden geben müssen. Obwohl es nach Abschnitt 3.1.3.4 durchaus Funktionen gibt, die gute Zerlegbarkeitseigenschaften haben, aber keine kompakte ROBDD–Darstellung besitzen, scheint es aber doch eine gute Heuristik zu sein, bei der Suche nach guten Zerlegungen einer Booleschen Funktion f zunächst nach kleinen ROBDD–Darstellungen von f zu suchen. Dazu können bekannte Variablenordnungsheuristiken, z.B. *sifting* von Rudell [Rud93] angewendet werden (vgl. auch Kapitel 1). Man erhält dann einen ROBDD F' für f

und eine p–elementige Variablenteilmenge $X_1^{(1)}$, die aus den ersten p Variablen in der Ordnung von F' besteht. Bei einer Zerlegung hinsichtlich $X_1^{(1)}$ benötigt man $\lceil \log(vz(X_1^{(1)}, f)) \rceil$ verschiedene Zerlegungsfunktionen. Man versucht nun, mit Hilfe eines greedy–Verfahrens eine p–elementige Variablenteilmenge zu finden, bei der man mit weniger Zerlegungsfunktionen auskommt. Dazu sucht man das Variablenpaar $x_i \in X_1^{(1)}$ und $x_j \in X \setminus X_1^{(1)}$ für das die Anzahl der Zerlegungsfunktionen bei Zerlegung hinsichtlich $X_2^{(1)} = (X_1^{(1)} \setminus \{x_i\}) \cup \{x_j\}$ minimal ist und vertauscht x_i und x_j in der Variablenordnung. Das Verfahren wird so lange fortgesetzt, bis sich in einem Schritt $\log(vz(X_i^{(1)}, f))$ nicht mehr verbessert oder bis eine vorgegebene Anzahl von Verschlechterungen von $\log(vz(X_{i-1}^{(1)}, f))$ auf $\log(vz(X_i^{(1)}, f))$ festgestellt wurde. Ausgewählt wird dann die Variablenteilmenge $X_j^{(1)}$, bei der $\log(vz(X_j^{(1)}, f))$ am kleinsten war.

Im vorliegenden Abschnitt wurde bisher immer vorausgesetzt, daß die Anzahl p der Variablen schon vorgegeben ist. Dies kann z.B. dann der Fall sein, wenn man sich auf gleichmächtige Zerlegungen mit $p = \frac{n}{2}$ festlegt oder wenn man sich auf eine bestimmte Technologie festgelegt hat, die eine Wahl von p nahelegt[7]. Es kann aber dann vorkommen, daß Zerlegungen mit anderen Mächtigkeiten p der Variablenteilmenge, hinsichtlich der zerlegt wird, zu wesentlich besseren Realisierungen führen oder daß es sogar keine Variablenteilmenge der Mächtigkeit p gibt, die zu einer nichttrivialen Zerlegung führt. Da wir uns auf nichttriviale Zerlegungen beschränken wollten[8], muß in diesem Fall p geändert werden. Gemäß Korollar 3.3 wäre (bei $n \geq 4$) $p = n - 1$ auf jeden Fall geeignet. Eine andere Möglichkeit besteht darin, p nicht fest vorzugeben, sondern im Verlauf des Verfahrens zu bestimmen: Ist f durch einen ROBDD F mit Variablenordnung $<_{index}$ gegeben, so läßt sich $vz(f, \{x_{index(1)}, \ldots, x_{index(p)}\})$ durch Tiefensuche im ROBDD (Zählen der Verbindungsknoten unterhalb einer Schnittlinie nach der Variablen $x_{index(p)}$) bestimmen. Es ist klar, daß man in einem Tiefensuchlauf auch parallel die Anzahl der Verbindungsknoten unterhalb *verschiedener* Schnittlinien (nach Variable $x_{index(1)}$, nach $x_{index(2)}$ usw.) bestimmen kann. Somit ist es möglich, in einem einzigen Tiefensuchlauf $vz(f, \{x_{index(1)}, \ldots, x_{index(p)}\})$ für $p = 1, \ldots, n-1$ zu bestimmen.
Will man eine gute Wahl von p erhalten, dann genügt es nicht mehr, die Realisierung mit der geringsten Anzahl von Zerlegungsfunktionen $\lceil \log(vz(f, \{x_{index(1)}, \ldots, x_{index(p)}\})) \rceil$ zu wählen. Stattdessen werden nun die Kosten einer Realisierung bei Zerlegung hinsichtlich $\{x_{index(1)}, \ldots, x_{index(p)}\}$ grob abgeschätzt, um eine Festlegung auf einen bestimmten Wert von p zu erzielen. Ist $r_p = \lceil \log(vz(f, \{x_{index(1)}, \ldots, x_{index(p)}\})) \rceil$ die minimale Anzahl von Zerlegungsfunktionen bei Zerlegung hinsichtlich $\{x_{index(1)}, \ldots, x_{index(p)}\}$, so sind bei der Zerlegung also r_p verschiedene Zerlegungsfunktionen aus B_p zu realisieren und eine Zusammensetzungsfunktion aus B_{n-p+r_p}. Bezeichnet man mit $cost(in)$ die geschätzten Kosten einer Booleschen Funktion mit in Eingängen und einem Ausgang, so betragen die

[7]Benutzt man z.B. lookup table–basierte FPGA's (**F**ield **P**rogrammable **G**ate **A**rrays), so kann man Boolesche Funktionen bis zu einer vorgegebenen Konstante b von Eingängen in einer einzigen funktionalen Einheit (LUT = lookup table) realisieren. Es bietet sich dann an, die Wahl $p = b$ zu bevorzugen.

[8]Ansonsten bricht das Verfahren bei rekursiver Anwendung auf Zerlegungs– und Zusammensetzungsfunktionen möglicherweise nicht ab.

geschätzten Kosten bei Zerlegung hinsichtlich $\{x_{index(1)}, \ldots, x_{index(p)}\}$

$$est\_cost_p(<_{index}) = \begin{cases} sharing(r_p) \cdot cost(p) + cost(n-p+r_p), & \text{falls } r_p < p, \\ \infty, & \text{falls } r_p = p, \end{cases}$$

wobei $sharing(r_p)$ ein Faktor ist, der berücksichtigt, daß bei der Realisierung von r_p verschiedenen Booleschen Funktionen i.a. Logik mehrfach verwendet werden kann (vgl. Abschnitt 3.4). *sharing* muß infolgedessen eine streng monoton wachsende Funktion sein, für die auf jeden Fall $sharing(r_p) \le r_p$ gilt (z.B. $sharing(r_p) = (r_p)^c$ mit $0 < c \le 1$). [9] Berücksichtigt man noch, daß bei Verwendung von strikten Zerlegungsfunktionen in der Zerlegung einer Funktion f einzelne Zerlegungsfunktionen nie von Variablen abhängen können, von denen f *nicht* abhängt, so kann man die Kostenabschätzung (für nichttriviale Zerlegungen) noch verfeinern zu

$$\begin{aligned} est\_cost_p(<_{index}) &= sharing(r_p) \cdot cost(ess\_var(\{x_{index(1)}, \ldots, x_{index(p)}\})) \\ &\quad + cost(ess\_var(\{x_{index(p+1)}, \ldots, x_{index(n)}\}) + r_p), \end{aligned}$$

wobei $ess\_var(Y)$ für eine Variablenteilmenge der Funktion f angibt, von wievielen Variablen aus Y f echt abhängt. Bei einer eher pessimistischen Kostenabschätzung kann man die Funktion *cost* z.B. als $cost(n) = \frac{2^n}{n}$ wählen (siehe Shannon–Effekt, Kapitel 1); auf jeden Fall sollte *cost* aber streng monoton wachsend gewählt werden (z.B. $cost(n) = n$, $cost(n) = n^2$ oder $cost(n) = n^3$). [10] Unter den sinnvollen Werten $p = 2$ bis $p = n-1$ wählt man nun den Wert für p aus, bei dem $est\_cost_p(<_{index})$ am geringsten ist. Im folgenden wird diese hinsichtlich der Variablenordnung $<_{index}$ beste Wahl von p als $p_{opt}(<_{index})$ bezeichnet, die dazugehörige Kostenabschätzung als $min\_cost(<_{index}) = \min_{2 \le p \le n-1} est\_cost_p(<_{index})$. Zur Bestimmung einer geeigneten Variablenteilmenge zur Zerlegung bietet sich ein greedy–Verfahren analog zum Verfahren für festes p an:

1. Wende ausgehend von einem ROBDD F für $f \in B_n$ eine Variablenordnungsheuristik zur Minimierung der ROBDD–Größe an. Man erhält einen ROBDD F_1 mit der Variablenordnung $<_{index_1}$. Bestimme $p_{opt}(<_{index_1})$ und $min\_cost(<_{index_1})$.
   $i = 1$, $fehlversuch = 0$.

2. Bestimme das Variablenpaar $x_{index_i(k)}$, $x_{index_i(j)}$ $(k \neq j)$, so daß für die nach Austausch von $x_{index_i(k)}$ und $x_{index_i(j)}$ resultierende Variablenordnung $<_{index_{i+1}}$ $min\_cost(<_{index_{i+1}})$ minimal ist[11]. Der aus dem Variablentausch resultierende ROBDD wird mit F_{i+1} bezeichnet.

3. Falls $min\_cost(<_{index_i}) \le min\_cost(<_{index_{i+1}})$: $fehlversuch = fehlversuch + 1$.

[9] Die Experimente aus Kapitel 4 wurden mit $sharing(r_p) = (r_p)^{0.9}$ durchgeführt.

[10] Die Experimente aus Kapitel 4 wurden mit $cost(n) = 0.3n^2$ für $n \ge 3$ durchgeführt.

[11] Betrachte dabei nur solche Variablenpaare, so daß die resultierende Variablenordnung bisher noch nicht in $<_{index_1}$ bis $<_{index_i}$ vorgekommen ist.

4. Falls $fehlversuch$ eine vorgegebene Schranke überschreitet: Breche das Verfahren ab, sonst:
   $i = i + 1$, fahre fort mit Punkt 2.

Nach Ende dieses Verfahrens wählt man die Variablenordnung $<_{index_i}$ aus, bei der $min\_cost(<_{index_i})$ minimal war. Die gefundene Variablenteilmenge ist $\{x_{index_i(1)}, \ldots, x_{index_i(p_{opt}(<_{index_i}))}\}$.

Die beschriebenen Ansätze lassen sich ohne weiteres auf zweiseitige Zerlegungen übertragen. Bei der Verwendung von Zerlegungsmatrizen muß man zusätzlich zur Anzahl der verschiedenen Zeilenmuster $vz(X^{(1)}, f)$ auch die Anzahl der verschiedenen Spaltenmuster $vs(X^{(1)}, f)$ bestimmen.
Ist f durch einen ROBDD F mit Variablenordnung $<_{index}$ gegeben, so benötigt man zur Bestimmung von $vs(X^{(1)}, f)$ ($|X^{(1)}| = p$) zusätzlich noch einen ROBDD F_{rev} mit umgekehrter Variablenordnung $<_{index_{rev}}$ mit $index_{rev}(i) = index(n + 1 - i)$. Wegen $vs(X^{(1)}, f) = vz(X \setminus X^{(1)}, f)$ kann man $vs(X^{(1)}, f)$ durch Zählen der Verbindungsknoten unterhalb der Schnittlinie nach den ersten $n - p$ Variablen in F_{rev} bestimmen. Für variables p wird die Kostenabschätzung für zweiseitige Zerlegungen (bei Variablenordnung $<_{index}$) folgendermaßen angepaßt:

$$\begin{aligned} est\_cost_p(<_{index}) \;=\; & sharing(r_p) \cdot cost(ess\_var(\{x_{index(1)}, \ldots, x_{index(p)}\}) \\ & +sharing(s_p) \cdot cost(ess\_var(\{x_{index(p+1)}, \ldots, x_{index(n)}\}) \\ & +cost(r_p + s_p) \end{aligned} \qquad (\star)$$

Dabei ist

$$r_p = \lceil \log(vz(f, \{x_{index(1)}, \ldots, x_{index(p)}\})) \rceil$$

und

$$s_p = \lceil \log(vs(f, \{x_{index(1)}, \ldots, x_{index(p)}\})) \rceil = \lceil \log(vz(f, \{x_{index(p+1)}, \ldots, x_{index(n)}\})) \rceil.$$

Anhand von Formel ($\star$) erkennt man schon, daß bei pessimistischer Kostenabschätzung (z.B. Wahl einer exponentiellen Funktion *cost*) von der Tendenz her eher gleichmächtige Zerlegungen bevorzugt werden.

Eine Modifikation des Verfahrens unter Ausnutzung von Symmetrien Boolescher Funktionen wird in Abschnitt 3.5 beschrieben.

3.2 Zerlegung partieller Funktionen

Bei praktischen Problemen treten häufig partielle Funktionen auf (d.h. Funktionen, deren don't care–Menge nicht leer ist), da die Schaltkreise, die diese partiellen Funktionen realisieren sollen, beispielsweise in ein größeres System eingebettet werden und man daher gewisse Eingabevektoren von vornherein ausschließen kann oder da im praktischen Problem bei einer Funktion mit mehreren Ausgängen für bestimmte Eingaben der Funktionswert nur an einem Teil der Ausgänge interessant ist.

Auch bei der rekursiven Ausnutzung von Zerlegungen bei der Logiksynthese können partielle Funktionen auftreten. Selbst wenn die ursprünglich zu realisierende Funktion total ist, können bei der rekursiven Behandlung der Zerlegungsfunktionen bzw. der Zusammensetzungsfunktion partielle Funktionen vorkommen. Ist bei einer einseitigen Zerlegung hinsichtlich einer Variablenteilmenge z.B. die Anzahl der Zeilen der zugehörigen Zerlegungsmatrix keine Zweierpotenz, so müssen die Zerlegungsfunktionen nicht alle möglichen Ausgangskombinationen annehmen (auch wenn die Anzahl der Zerlegungsfunktionen minimal gewählt wird). Die Zusammensetzungsfunktion kann dann partiell sein (bzw. ist dann bei Verwendung strikter Zerlegungsfunktionen auf jeden Fall partiell).
Ist die zu realisierende Funktion schon partiell, so können darüber hinaus auch die Zerlegungsfunktionen partiell sein.

Ein Ω–Schaltkreis, der eine partielle Funktion f realisiert, definiert eine totale Funktion, die eine Erweiterung von f darstellt. Wie in dieser totalen Funktion die bisher undefinierten Funktionswerte von f belegt werden, ist unerheblich. Allerdings haben im allgemeinen verschiedene totale Erweiterungen von f verschiedene Komplexität. Auch die minimale Anzahl von Zerlegungsfunktionen einer Zerlegung von f kann bei verschiedenen Erweiterungen von f verschieden sein.

Wie man unter Berücksichtigung der Resultate von Shannon [Sha49] (vgl. Abschnitt 1.2) leicht sieht, wird man bei partiellen Booleschen Funktionen mit sehr großen don't care–Mengen durch eine ungeschickte Belegung der don't cares mit hoher Wahrscheinlichkeit zu Booleschen Funktionen mit sehr hoher Komplexität gelangen. In Verbindung mit den Überlegungen aus Abschnitt 3.1.2 sieht man ebenfalls leicht, daß sich dann auch mit hoher Wahrscheinlichkeit Boolesche Funktionen mit schlechten Zerlegungseigenschaften ergeben.

Es ist daher sinnvoll, bei der Logiksynthese die undefinierten Funktionswerte einer partiellen Funktion nicht beliebig festzulegen, sondern nach günstigen Erweiterungen der partiellen Funktion zu suchen.

Der vorliegende Abschnitt beschäftigt sich mit der Belegung von don't cares im Hinblick auf das Zerlegungsverfahren. Die don't cares sollen so belegt werden, daß die Anzahl der benötigten Zerlegungsfunktionen bei der Zerlegung minimiert wird.

Zunächst werden die entsprechenden Probleme EKM und ZKM für einseitige und zweiseitige Zerlegungen formuliert und ihre Komplexität untersucht. Danach werden Lösungen für EKM und ZKM angegeben.

Im darauffolgenden Teilabschnitt wird beschrieben, wie sich die Lösungen für EKM und ZKM auf ROBDD–Darstellungen übertragen. Speziell für ROBDDs wurde parallel zu dieser Arbeit auch von Chang/Marek-Sadowska [CCM94] im Zusammenhang mit der Minimierung von ROBDD–Größen eine Lösung für EKM angegeben.

Neben den in diesem Abschnitt behandelten Möglichkeiten zur don't care–Belegung durch ein „lokales“ Verfahren zur Minimierung der Anzahl der Zerlegungsfunktionen im aktuellen Zerlegungsschritt[12] wurde in Kapitel 2.2 schon ein eher „globales" Verfahren zur Erzeugung starker Symmetrien entwickelt. Im späteren Abschnitt 3.5.2 ist es gelungen, die Verträglichkeit dieser beiden Ansätze zu beweisen: Unter Beachtung der in Abschnitt 3.5.2 angegebenen Voraussetzungen werden durch das Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen keine starken Symmetrien zerstört.

Da don't cares Freiheiten darstellen, die man bei der Berechnung guter Realisierungen Boolescher Funktionen ausnutzen kann, ist es natürlich erstrebenswert, im Rahmen des rekursiven Zerlegungsverfahrens möglichst viele don't cares zu identifizieren. Daher werden im letzten Punkt des Abschnitts don't care–Mengen für Zerlegungs– und Zusammensetzungsfunktionen (bei Vorgabe einer totalen oder partiellen Funktion) hergeleitet. Es konnte bewiesen werden, daß hier hergeleiteten don't care–Mengen auch wirklich maximal sind.

3.2.1 Die Probleme EKM und ZKM

Zunächst wird die Behandlung partieller Funktionen bei einseitigen Zerlegungen betrachtet.

Analog zur Gleichheit von Zerlegungsmatrixzeilen bei totalen Funktionen wird hier die Kompatibilität von Zerlegungsmatrixzeilen definiert.

Definition 3.8 (Kompatible Zeilen einer Zerlegungsmatrix)
Sei $f \in S(D)$ ($D \subseteq \{0,1\}^n$) eine Funktion mit den Eingangsvariablen $x_1, \ldots, x_n$. Sei $Z(X^{(1)})$ die Zerlegungsmatrix von f hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. Sind

$$\epsilon^{(1)} = (\epsilon_1^{(1)}, \ldots, \epsilon_p^{(1)}) \text{ und } \epsilon^{(2)} = (\epsilon_1^{(2)}, \ldots, \epsilon_p^{(2)}) \in \{0,1\}^p$$

und $i = int(\epsilon^{(1)})$, $j = int(\epsilon^{(2)})$, so heißen die Zeilen von $Z(X^{(1)})$ mit den Indizes i und j kompatibel (in Zeichen: $\epsilon^{(1)} \sim \epsilon^{(2)}$), wenn es keinen Spaltenindex $0 \leq k \leq 2^{n-p} - 1$ gibt mit

$$\begin{array}{ll} (Z(X^{(1)})_{ik} = 0 \text{ und } Z(X^{(1)})_{jk} = 1) & \text{oder} \\ (Z(X^{(1)})_{ik} = 1 \text{ und } Z(X^{(1)})_{jk} = 0). & \end{array}$$

(Mit anderen Worten:

$$\epsilon^{(1)} \sim \epsilon^{(2)} \iff$$
$$\nexists \delta \in \{0,1\}^{n-p} \text{ mit } (\epsilon^{(1)}, \delta), (\epsilon^{(2)}, \delta) \in D \text{ und } f(\epsilon^{(1)}, \delta) \neq f(\epsilon^{(2)}, \delta).)$$

Bemerkung 3.3 *„$\sim$" ist* keine *Äquivalenzrelation auf $\{0,1\}^p$.*

Dies zeigt das folgende kleine Beispiel einer Zerlegungsmatrix:

[12]Dieses Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen ist deshalb als „lokal" zu bezeichnen, weil es durchgeführt wird für eine fest vorgegebene Variablenaufteilung bei der Zerlegung bzw. für eine fest vorgegebene Variablenordnung eines ROBDD.

$$\begin{array}{c|cc|}
\cline{2-3}
0 & * & 1 \\
1 & 0 & * \\
2 & 0 & 0 \\
3 & 0 & 0 \\
\cline{2-3}
\end{array}$$

Zeilen 0 und 1 sind kompatibel, Zeilen 1 und 2 sind kompatibel, aber Zeilen 0 und 2 sind *nicht* kompatibel.

Analog zur Anzahl verschiedener Zeilen einer Zerlegungsmatrix bei totalen Funktionen wird hier folgende Bezeichnung eingeführt:

Bezeichnung 3.4 *Sei $f \in S(D)$ ($D \subseteq \{0,1\}^n$) mit den Eingangsvariablen $x_1, \ldots, x_n$. Sei $Z(X^{(1)})$ die Zerlegungsmatrix von f hinsichtlich der Variablenteilmenge $X^{(1)} := \{x_1, \ldots, x_p\}$.*

Dann wird die minimale Anzahl von Mengen, in die $\{0,1\}^p$ partitioniert werden kann, so daß je 2 Elemente der gleichen Menge (bzgl. $\sim$) kompatibel sind, mit $vkk(X^{(1)}, f)$ bezeichnet.

Es ist nun wie bei totalen Funktionen leicht, ein Kriterium dafür anzugeben, daß zu einer vorgegebenen Variablenteilmenge eine Zerlegung mit einer festen Anzahl von Zerlegungsfunktionen existiert. (Ein entsprechender Satz wurde schon von Karp in [Kar63] bewiesen.)

Satz 3.4 *Sei $f \in S(D)$ ($D \subseteq \{0,1\}^n$) eine Funktion mit den Eingangsvariablen $x_1, \ldots, x_n$. Dann gibt es genau dann eine einseitige Zerlegung von f hinsichtlich der Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$, so daß für alle $(x_1, \ldots, x_n)$ aus D*

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$$

gilt, wenn

$$r \geq \log(vkk(X^{(1)}, f)).$$

Beweis: Der Beweis erfolgt genau analog zum Beweis von Satz 3.1. Es ist lediglich jeweils die Gleichheit von Zeilen durch „Kompatibilität von Zeilen“ zu ersetzen. □

Angenommen die Elemente von $\{0,1\}^p$ sind so in verschiedene Mengen $PK_1, \ldots, PK_{vkk(X^{(1)},f)}$ partitioniert, daß für alle Paare $\epsilon^{(1)}$ und $\epsilon^{(2)} \in PK_i$ ($1 \leq i \leq vkk(X^{(1)}, f)$) $\epsilon^{(1)} \sim \epsilon^{(2)}$ gilt. Betrachtet man also 2 Zeilen der Zerlegungsmatrix $Z(X^{(1)})$, deren Indizes aus PK_i sind, so kann es nicht vorkommen, daß in der einen Zeile in einer bestimmten Spalte eine 1 steht, in der anderen Zeile in dieser Spalte eine 0. Dann kann man die don't care–Stellen von f („*“ in der Zerlegungsmatrix) so belegen, daß alle Zeilen, die zu Elementen einer Menge PK_i ($1 \leq i \leq vkk(X^{(1)}, f)$) gehören, gleich werden. Man erhält so eine totale Erweiterung f' von f, bei der die Äquivalenzklassen der Äquivalenzrelation $\equiv$ auf $\{0,1\}^p$, die durch die Gleichheit von Zeilen der Zerlegungsmatrix hinsichtlich $X^{(1)}$ induziert wird, genau mit PK_1 bis $PK_{vkk(X^{(1)},f)}$ übereinstimmen. Es gilt also $vkk(X^{(1)}, f) = vz(X^{(1)}, f')$.

Um für eine partielle Funktion f bei vorgegebener Variablenteilmenge $X^{(1)}$ die minimale Anzahl von Zerlegungsfunktionen in einer Zerlegung von f hinsichtlich $X^{(1)}$ zu bestimmen, muß man (statt $vz(X^{(1)}, f)$ bei totalen Funktionen) $vkk(X^{(1)}, f)$ berechnen.

Das Problem, das dabei auftritt, besteht darin, daß die Aufgabe, bei einer gegebenen Zerlegungsmatrix die minimale Anzahl von Kompatibilitätsklassen $vkk(X^{(1)}, f)$ zu bestimmen, *wesentlich* schwieriger ist als die Bestimmung der Anzahl verschiedener Zeilen der Matrix. Es stellt sich heraus, daß dieses Problem sogar NP–hart ist. Es ist also sehr unwahrscheinlich, daß man einen Algorithmus finden kann, der $vkk(X^{(1)}, f)$ berechnet und dessen Laufzeit polynomiell in der Größe der Zerlegungsmatrix ist.

Folgendes Problem ist zu lösen:

Problem EKM (Einseitige Kommunikationsminimierung)

Gegeben: Partielle Funktion $f \in S(D)$ $(D \subseteq \{0,1\}^n)$ in den Eingangsvariablen $x_1, \ldots, x_n$ mit der Zerlegungsmatrix $Z(X^{(1)})$ hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$.

Gesucht: Anzahl von Zerlegungsfunktionen bei einer einseitigen kommunikationsminimalen Zerlegung von f hinsichtlich $X^{(1)}$, d.h. $\lceil \log(vkk(X^{(1)}, f)) \rceil$.

Es gilt (siehe auch [SM93]):

Satz 3.5 *Das Problem* **EKM** *ist NP–hart.*

Satz 3.5 wird bewiesen durch Angabe einer Polynomzeittransformation vom NP–vollständigen Problem „Partition into Cliques“ (PC) nach EKM.

Die Definition von Problem PC findet man in [GJ79], S. 193 bzw. in Abschnitt 2.2.1.2 auf Seite 52.

Das Problem PC bleibt NP–vollständig, wenn man die Grenze K auf Zweierpotenzen beschränkt, d.h. auch folgendes Problem ist NP–vollständig:

Problem PC'

Gegeben: Ein Graph $G = (V, E)$ und eine natürliche Zahl $K = 2^m$ für $m \in \mathbf{N}$, $K \leq |V|$.

Gesucht: Kann V partitioniert werden in K disjunkte Mengen $V_1, \ldots, V_K$, so daß für $1 \leq i \leq K$ der Teilgraph, der alle Knoten aus V_i umfaßt, ein vollständiger Graph ist?

Beweisskizze: Mit Hilfe einer einfachen Polynomzeittransformation von PC nach PC' sieht man, daß PC' NP–hart ist: Ist bei einer Instanz des Problems PC die natürliche Zahl K keine Zweierpotenz, so wähle die nächsthöhere Zweierpotenz 2^m und füge zum Graphen

G $2^m - K$ zusätzliche Knoten hinzu, die weder Quelle noch Ziel einer Kante sind. Der resultierende Graph hat genau dann eine Partition in 2^m vollständige Teilgraphen, wenn der ursprüngliche Graph eine Partition in K vollständige Teilgraphen hat. □

Nun kann Satz 3.5 bewiesen werden. Die Idee des Beweises besteht darin, daß man die Kompatibilitätsrelation $\sim$ auf Zerlegungsmatrixzeilen als einen Graphen betrachtet. Das Problem, die minimale Anzahl von Kompatibilitätsklassen auf den Zerlegungsmatrixzeilen zu bestimmen, ist äquivalent zum Problem, eine Partition des Graphen in vollständige Teilgraphen zu bestimmen. Die Einzelheiten sind in folgendem Beweis zu finden:

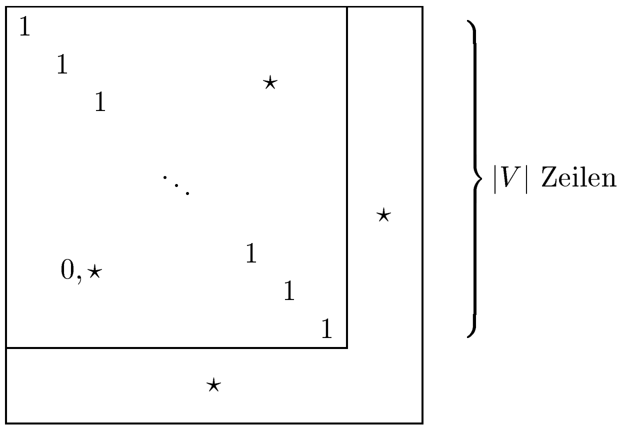
Beweis: Gegeben sei eine Instanz eines PC'–Problems bestehend aus einem Graphen $G = (V, E)$ und einer natürlichen Zahl $K \leq |V|$ mit $K = 2^m$ für $m \in \mathbf{N}$.
Zu G kann in Polynomzeit eine partielle Funktion f bestimmt werden, so daß bei einseitiger Zerlegung von f hinsichtlich einer Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ $vkk(X^{(1)}, f)$ gleich der minimalen Anzahl von vollständigen disjunkten Teilgraphen von G ist.
Zur Konstruktion von f:
Sei p minimal mit $2^p \geq |V|$. Definiere $f : \{0,1\}^{2p} \rightsquigarrow \{0,1\}$ durch Angabe einer Zerlegungsmatrix $Z(X^{(1)})$ hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. Nehme dazu der Einfachheit halber o.B.d.A. an, daß $V = \{0, \ldots, |V| - 1\}$.
Es gelte

$$Z(X^{(1)})_{ij} = \star \text{ für } i \geq |V| \text{ oder } j \geq |V|.$$

Für alle $0 \leq i, j \leq |V| - 1$ wird definiert:

$$Z(X^{(1)})_{ij} = \begin{cases} \star, & \text{falls } \; i < j \\ 1, & \text{falls } \; i = j \\ 0, & \text{falls } \; i > j \text{ und } \{i, j\} \notin E \\ \star, & \text{falls } \; i > j \text{ und } \{i, j\} \in E \end{cases}$$

$Z(X^{(1)})$ hat also folgende Gestalt:



Seien $i, j \in \{0, \ldots, |V|-1\}$ und sei o.B.d.A. $i > j$.
Für alle Spalten $k < j$ gilt:

$$Z(X^{(1)})_{jk}, Z(X^{(1)})_{ik} \in \{0, \star\}.$$

Für alle Spalten $k > j$ gilt:

$$Z(X^{(1)})_{jk} = \star.$$

Für Spalte j gilt:

$$Z(X^{(1)})_{jj} = 1 \text{ und } Z(X^{(1)})_{ij} = 0 \overset{\text{Def. von } f}{\Longleftrightarrow} \{i, j\} \notin E.$$

Also sind Zeilen i und j genau dann kompatibel (bzw. $bin_p(i) \sim bin_p(j)$), wenn $\{i, j\} \in E$. Also bilden die Knoten $i_1, \ldots i_m$ genau dann einen vollständigen Teilgraphen, wenn die Zeilen mit den Indizes $i_1, \ldots i_m$ paarweise kompatibel sind.
Zeilen mit Indizes $i \geq |V|$ sind zu allen anderen Zeilen kompatibel. Also gilt:

- Falls es eine Partition von $V = \{0, \ldots, |V|-1\}$ in K disjunkte Mengen $V_1, \ldots, V_K$ gibt, so daß die durch die Knoten aus V_i $(1 \leq i \leq K)$ induzierten Teilgraphen vollständige Graphen sind, dann ist

  $$\{V_1, \ldots, V_{K-1}, V_K \cup \{|V|, \ldots, 2^p - 1\}\}$$

  eine Partition der Zeilenindizes, bei der die einzelnen Teilmengen nur Indizes kompatibler Zeilen enthalten. Es gilt dann: $vkk(X^{(1)}, f) \leq K = 2^m$ und es gibt nach Satz 3.4 eine Zerlegung hinsichtlich $X^{(1)}$ mit m Zerlegungsfunktionen.

- Gibt es eine Zerlegung von f hinsichtlich $X^{(1)}$ mit m Zerlegungsfunktionen, so gilt nach Satz 3.4:

  $$vkk(X^{(1)}, f) \leq 2^m = K.$$

  Es gibt also eine Partition $\{PK_1, \ldots, PK_K\}$ der Zeilenindizes $\{0, \ldots, 2^p - 1\}$, so daß die Zeilen mit Indizes aus den Mengen PK_i $(1 \leq i \leq K)$ paarweise kompatibel sind. Da Zeilen mit Indizes $i \geq |V|$ zu allen anderen Zeilen kompatibel sind und $K = 2^m \leq |V|$, kann man annehmen, daß für alle $1 \leq i \leq K$ $PK_i \cap V \neq \emptyset$. Dann gibt es auch eine Partition von V in Mengen

  $$V_1 = PK_1 \cap V, \ldots, V_K = PK_K \cap V,$$

  so daß die durch die Knoten aus V_i induzierten Teilgraphen vollständige Graphen sind.

Insgesamt gilt: Die in Polynomzeit konstruierbare Funktion f hat genau dann eine Zerlegung hinsichtlich $X^{(1)}$ mit m Zerlegungsfunktionen, wenn sich die Knoten von G so partitionieren lassen, daß sich $K = 2^m$ vollständige Teilgraphen ergeben[13].

□

Gemäß Satz 3.2 (Seite 81) kann man bei totalen Funktionen die minimale Anzahl von Zerlegungsfunktionen anhand der Anzahl der verschiedenen Zeilen und verschiedenen Spalten einer Zerlegungsmatrix bestimmen. Definiert man wie auf den Zeilen einer Zerlegungsmatrix einer partiellen Funktion auch auf den Spalten eine Kompatibilitätsrelation, so gilt die analoge Aussage zu Satz 3.2 allerdings nicht. Bei einer Zerlegung von f hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$ ist die minimale Anzahl der Zerlegungsfunktionen *nicht* durch $\lceil \log(vkk(X^{(1)}, f)) \rceil + \lceil \log(vkk(X^{(2)}, f)) \rceil$ gegeben. Das folgende kleine Beispiel zeigt den Grund dafür:

Beispiel 3.6 $\boxed{\begin{array}{cc} * & 1 \\ 0 & * \end{array}}$

Die beiden Zeilen und die beiden Spalten der Matrix sind jeweils kompatibel. Es gibt aber keine Ersetzung der don't care–Stellen durch Elemente aus $\{0,1\}$, so daß *sowohl* die Zeilen gleich werden *als auch* die Spalten gleich werden.

Das Beispiel zeigt, daß durch den Ausdruck $\lceil \log(vkk(X^{(1)}, f)) \rceil + \lceil \log(vkk(X^{(2)}, f)) \rceil$ die Anzahl der benötigten Zerlegungsfunktionen einer kommunikationsminimalen Zerlegung unterschätzt werden kann.

Das zu EKM analoge Problem für zweiseitige Zerlegungen lautet folgendermaßen:

ZKM (Problem der zweiseitigen Kommunikationsminimierung)

Gegeben: Partielle Funktion $f \in S(D)$ $(D \subseteq \{0,1\}^n)$ in den Eingangsvariablen $x_1, \ldots, x_n$ mit der Zerlegungsmatrix $Z(X^{(1)})$ hinsichtlich der Variablenaufteilung $\{X^{(1)}, X^{(2)}\} = \{\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_n\}\}$.

Gesucht: Minimale Anzahl von Zerlegungsfunktionen bei einer zweiseitigen Zerlegung von f hinsichtlich $\{X^{(1)}, X^{(2)}\}$.

Beispiel 3.6 zeigt, daß die exakte Lösung von **ZKM** nicht einfach auf eine zweimalige Lösung von **EKM** zurückgeführt werden kann.

Auch für **ZKM** gilt:

Satz 3.6 *Das Problem* **ZKM** *ist NP–hart.*

In [SM93] wird Satz 3.6 bewiesen durch Angabe einer Polynomzeittransformation von einem NP–vollständigen Problem **BIP** (**B**inary **I**nteger **P**rogramming, [LP81]) nach **ZKM**. Im Gegensatz zum Beweis zu Satz 3.5 ist die Konstruktion wesentlich komplexer und technischer und legt auch keinen Algorithmus zur Lösung von ZKM nahe.

[13] Wenn man in Polynomzeit die minimale Anzahl von Zerlegungsfunktionen bei Zerlegung von f hinsichtlich $X^{(1)}$ berechnen könnte, so könnte man natürlich auch berechnen, ob es eine Zerlegung mit m Zerlegungsfunktionen gibt.

3.2.2 Lösungen für die Probleme EKM und ZKM

Anhand des Beweises zu Satz 3.5 wird auch ein Verfahren deutlich, wie man für eine Zerlegung von f hinsichtlich $X^{(1)}$ $vkk(X^{(1)}, f)$ berechnen bzw. approximieren kann:

- Bestimme die zugehörige Zerlegungsmatrix $Z(X^{(1)})$.
- Bestimme die Kompatibilitätsrelation $\sim$ auf $\{0,1\}^p$.
- Interpretiere $\sim$ als Kantenmenge eines ungerichteten Graphen G mit Knoten aus $\{0,1\}^p$. (Es gibt genau dann eine Kante $\{\epsilon_1, \epsilon_2\}$ für $\epsilon_1, \epsilon_2 \in \{0,1\}^p$, wenn $\epsilon_1 \sim \epsilon_2$.)
- Bestimme $vkk(X^{(1)}, f)$ als die minimale Zahl K, so daß $\{0,1\}^p$ in Mengen $V_1, \ldots, V_K$ partitioniert werden kann, wobei die Teilgraphen aller Knoten in V_i jeweils vollständige Graphen sind. Dies kann durch Anwendung einer Heuristik zur Lösung des bekannten Problems „Partition into Cliques“ auf G geschehen.

Will man zu einer gegebenen partiellen Funktion f und einer Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$ eine totale Erweiterung f' finden, so daß die Anzahl der Zerlegungsfunktionen, die bei einer Zerlegung hinsichtlich $\{X^{(1)}, X^{(2)}\}$ nötig sind, minimal wird (Problem **ZKM**), so kann man unter Ausnutzung der Heuristik für **EKM** folgendes Näherungsverfahren angegeben:

1. Bestimme bei der Zerlegungsmatrix $Z := Z(X^{(1)})$ eine Einteilung der Zeilen in Kompatibilitätsklassen $PK_1, \ldots, PK_z$ hinsichtlich der Kompatibilitätsrelation $\sim$, wobei z möglichst minimal ist. (Dies entspricht einer Lösung zu Problem **EKM** und erfolgt durch Anwendung einer Heuristik zum Problem „Partition into Cliques“.)
2. Gibt es in einer Kompatibilitätsklasse PK_i eine Zeile, die in Spalte j eine 1 (bzw. eine 0) hat, so belege bei allen Zeilen aus PK_i die j–te Spalte mit 1 (bzw. 0). (Wegen der Kompatibilität muß man nur don't care–Stellen (*) ändern. Gibt es bei den Zeilen aus PK_i in Spalte j keine 0 bzw. keine 1, so bleiben die entsprechenden don't care–Stellen (*) erhalten. Man erhält so aus Z eine Matrix Z' (mit z verschiedenen Zeilenmustern).
3. Bestimme nun bei Z' eine Einteilung der Spalten in Klassen $PI_1, \ldots, PI_s$ kompatibler Spalten, wobei s möglichst minimal ist.
4. Belege nun bei allen Spalten aus einer Kompatibilitätsklasse PI_i die don't cares so, daß die Spalten gleich werden. Man erhält dadurch aus Z' eine Matrix Z''.

Die resultierende Matrix hat z verschiedene Zeilen und s verschiedene Spalten. Für die Korrektheit dieser Aussage ist es wesentlich, festzustellen, daß die Belegung von don't cares in Schritt 4. die Gleichheit der Zeilen mit Indizes in PK_i $(1 \leq i \leq z)$ nicht zerstört:

Lemma 3.8 *Sind beim obigen Algorithmus 2 Zeilenindizes z_1 und z_2 von Matrix Z in einer Kompatibilitätsklasse PK_i, d.h. sind die entsprechenden Zeilen in Z' gleich, so sind auch die Zeilen mit Indizes z_1 und z_2 in Matrix Z'' gleich.*

Beweis:
Annahme: Zeilen z_1 und z_2 von Z'' sind nicht gleich, obwohl z_1 und z_2 in *einer* Kompatibilitätsklasse PK_i liegen.
Dann muß es o.B.d.A. s_1 geben, so daß $Z''_{z_1 s_1} = c \in \{0,1\}$ und $Z''_{z_2 s_1} \neq c$.
Außerdem muß gelten: $Z'_{z_1 s_1} = *$ und $Z'_{z_2 s_1} = *$.
(Wegen Definition von Z' in 2.) und da Zeilen z_1 und z_2 von Z kompatibel sind.)
Sei Spalte s_1 in Kompatibilitätsklasse PI_j. Dann muß es in PI_j eine Spalte s_2 geben mit $Z'_{z_1 s_2} = c$. Es ergibt sich folgendes Bild von Z':

$$
\begin{array}{ccccc|c}
 & \vdots & & \vdots & & \\
\cdots & * & \cdots & c & \cdots & z_1 \\
 & \vdots & & \vdots & & \\
\cdots & * & \cdots & \epsilon & \cdots & z_2 \\
 & \vdots & & \vdots & & \\
 & s_1 & & s_2 & &
\end{array}
$$

- Wäre $\epsilon = c$, so würde aus $s_1, s_2 \in PI_j$ $Z''_{z_2 s_1} = c$ folgen, im Widerspruch zur Annahme.
- Wäre $\epsilon = \overline{c}$, so wären Zeilen z_1 und z_2 nicht in *einer* Kompatibilitätsklasse PK_i.
- $\epsilon = *$ ist nicht möglich, denn wegen c in Zeile z_1 wäre $*$ in Schritt 2.) des Algorithmus durch c ersetzt worden.

Die dargestellte Situation kann also nicht auftreten. Es ergibt sich ein Widerspruch zur Annahme. □

Das folgende kleine Beispiel zeigt, daß die Durchführung von Schritt 2.) des Algorithmus entscheidend dafür ist, daß man die Zusicherung von Lemma 3.8 machen kann:

Beispiel 3.7 Gegeben ist folgende Zerlegungsmatrix einer Funktion $f : \{0,1\}^3 \rightsquigarrow \{0,1\}$:

x_1 \ x_2, x_3	0 0	0 1	1 0	1 1
0	1	$*$	0	0
1	$*$	0	0	0

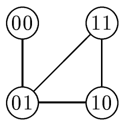
Der Algorithmus würde zunächst feststellen, daß die beiden Zeilen der Zerlegungsmatrix kompatibel sind. Wird dann Schritt 2.) ausgelassen, so kann man eine Partition der Spalten in 2 Mengen paarweise kompatibler Spalten finden. Angenommen Schritt 3.) faßt z.B. die ersten beiden und die letzten beiden Spalten zusammen. Es ergibt sich dann nach Schritt 4.) folgende Matrix Z'', bei der die beiden Zeilen *nicht* kompatibel sind:

$$
\begin{array}{c|cccc|}
x_2 & 0 & 0 & 1 & 1 \\
x_3 & 0 & 1 & 0 & 1 \\
x_1 & & & & \\
\hline
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
$$

Ebenso sieht man an diesem Beispiel, daß man mit dem Algorithmus, der das Verfahren für einseitige Zerlegungen nacheinander auf die Zeilen und die Spalten anwendet, nur *Näherungslösungen* für ZKM erhält:

$$
\begin{array}{c|cc|}
x_1 & 0 & 1 \\
x_2x_3 & & \\
\hline
0\ 0 & 1 & * \\
0\ 1 & * & 0 \\
1\ 0 & 0 & 0 \\
1\ 1 & 0 & 0 \\
\hline
\end{array}
$$

Die Kompatibilitätsrelation $\sim$ auf den Zeilenindizes aus $\{0,1\}^2$ hat folgendes Aussehen:



Es gibt also verschiedene Möglichkeiten, die Zeilen in 2 Kompatibilitätsklassen zu partitionieren:

- Wählt man $\{\{00, 01\}, \{10, 11\}\}$, dann erhält man folgende Matrix Z'

$$
\begin{array}{c|cc|}
x_1 & 0 & 1 \\
x_2x_3 & & \\
\hline
0\ 0 & 1 & 0 \\
0\ 1 & 1 & 0 \\
1\ 0 & 0 & 0 \\
1\ 1 & 0 & 0 \\
\hline
\end{array}
$$

  und man erhält 2 verschiedene Spalten.

- Wählt man jedoch $\{\{00\}, \{01, 10, 11\}\}$, dann erhält man folgende Matrix Z'

x_4	0	0	0	0	1	1	1	1
x_5	0	0	1	1	0	0	1	1
x_6	0	1	0	1	0	1	0	1
$x_1x_2x_3$								
0 0 0	$*$	1	1	$*$	$*$	0	0	$*$
0 0 1	1	0	0	1	$*$	$*$	$*$	$*$
0 1 0	1	0	0	1	$*$	$*$	$*$	$*$
0 1 1	$*$	1	1	$*$	$*$	0	0	$*$
1 0 0	$*$	$*$	$*$	$*$	0	1	1	0
1 0 1	0	$*$	$*$	0	1	$*$	$*$	1
1 1 0	0	$*$	$*$	0	1	$*$	$*$	1
1 1 1	$*$	$*$	$*$	$*$	0	1	1	0

Abbildung 3.7: Zerlegungsmatrix $Z(\{x_1, \ldots, x_3\})$ einer Beispielsfunktion isp

x_1	0	1
x_2x_3		
0 0	1	$*$
0 1	0	0
1 0	0	0
1 1	0	0

und die beiden Spalten sind kompatibel.

3.2.3 Lösungen für EKM und ZKM bei ROBDD–Darstellungen

In diesem Abschnitt wird dargestellt, wie sich bei ROBDD–Darstellungen die Probleme EKM und ZKM lösen lassen.

Parallel zu dieser Arbeit wurde in Zusammenhang mit der Minimierung von ROBDD–Größen für partielle Boolesche Funktionen die Lösung von EKM bei ROBDD–Darstellungen auch von Chang/Marek–Sadowska [CCM94] betrachtet. Es wird hier mit einer Darstellung partieller Boolescher Funktionen gearbeitet, die in [CCM94] eingeführt wurde:
Außer den Eingangsvariablen $\{x_1, \ldots, x_n\}$ einer partiellen Funktion f wird noch eine zusätzliche Variable z eingeführt und f wird durch einen ROBDD für die totale Funktion $ext(f) = \overline{z} \cdot f_{off} + z \cdot f_{on}$ aus B_{n+1} repräsentiert. Bevor auf ein Verfahren zur Lösung von EKM und ZKM für ROBDD–Darstellungen partieller Funktionen eingegangen wird, soll zunächst das Vorgehen bei Zerlegungsmatrizen noch einmal genauer betrachtet werden.
Will man hinsichtlich einer Variablenteilmenge $\{x_1, \ldots, x_p\}$ zerlegen, so wird die Zerlegungsmatrix $Z(\{x_1, \ldots, x_p\})$ aufgestellt (im Beispiel von Abbildung 3.7 $Z(\{x_1, \ldots, x_3\})$).

Wie bei totalen Funktionen kann man auch Kofaktoren partieller Funktionen definieren:

Bezeichnung 3.5 *Sei $f \in S(D)$, $D \subseteq \{0,1\}^n$. Der* **Kofaktor** *von f hinsichtlich $x_1^{\epsilon_1} \dots x_p^{\epsilon_p}$ ist eine Funktion $f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ aus $S(D')$, wobei*

$$D' = \{(y_1, \dots, y_{n-p}) \in \{0,1\}^{n-p} | (\epsilon_1, \dots, \epsilon_p, y_1, \dots, y_{n-p}) \in D\}$$

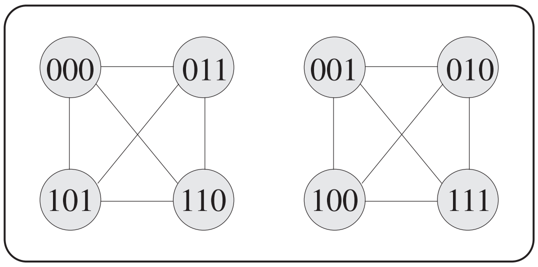
und

$$f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}(y_1, \dots, y_{n-p}) = f(\epsilon_1, \dots, \epsilon_p, y_1, \dots, y_{n-p}) \ \forall (y_1, \dots, y_{n-p}) \in D'.$$

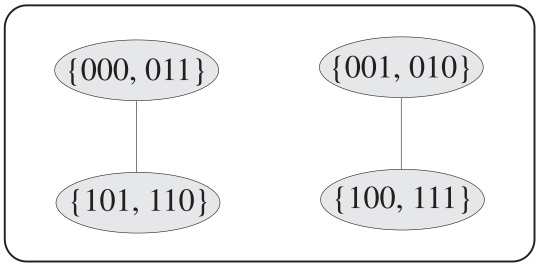
Analog zu totalen Funktionen kann man auch hier die Zeilen der Zerlegungsmatrix $Z(\{x_1, \dots, x_p\})$ einer partiellen Funktion f als Funktionstabellen von Kofaktoren $f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ interpretieren.

Bei der Bestimmung einer minimalen Partition $\{PK_1, \dots, PK_{vkk(\{x_1,\dots,x_p\},f)}\}$ von $\{0,1\}^p$, bei der die Elemente einer Klasse PK_i paarweise kompatibel sind, kann man den Suchraum einschränken: $\{0,1\}^p/_\equiv = \{K_1, \dots, K_{vz(\{x_1,\dots,x_p\},f)}\}$ sei die durch *Zeilengleichheit* in $Z(\{x_1, \dots, x_p\})$ induzierte Äquivalenzklasseneinteilung von $\{0,1\}^p$. Es ist klar, daß man $\{PK_1, \dots, PK_{vkk(\{x_1,\dots,x_p\},f)}\}$ so wählen kann, daß $PK_i = \bigcup_{j=1}^{l} K_{i_j}$, also daß alle Indizes gleicher Zeilen von $Z(\{x_1, \dots, x_p\})$ in der gleichen Kompatibilitätsklasse sind.

Für die Funktion *isp* aus Abbildung 3.7 sieht der Graph zur Kompatibilitätsrelation auf den Zeilen von $Z(\{x_1, \dots, x_3\})$ folgendermaßen aus:



Beschränkt man sich auf eine Relation auf $\{K_1, \dots, K_{vz(\{x_1,\dots,x_p\},f)}\}$, so kommt man zu folgendem Graph:



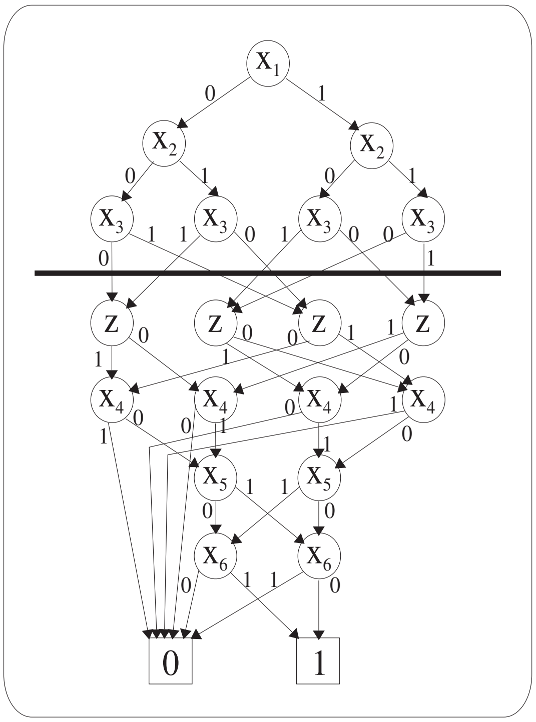


Abbildung 3.8: ROBDD zu $ext(isp) = \overline{z} \cdot isp_{off} + z \cdot isp_{on}$.

z	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
x_4	0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
x_5	0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
x_6	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
$x_1 x_2 x_3$		
0 0 0	0 0 0 0 0 1 1 0	0 1 1 0 0 0 0 0
0 0 1	0 1 1 0 0 0 0 0	1 0 0 1 0 0 0 0
0 1 0	0 1 1 0 0 0 0 0	1 0 0 1 0 0 0 0
0 1 1	0 0 0 0 0 1 1 0	0 1 1 0 0 0 0 0
1 0 0	0 0 0 0 1 0 0 1	0 0 0 0 0 1 1 0
1 0 1	1 0 0 1 0 0 0 0	0 0 0 0 1 0 0 1
1 1 0	1 0 0 1 0 0 0 0	0 0 0 0 1 0 0 1
1 1 1	0 0 0 0 1 0 0 1	0 0 0 0 0 1 1 0

Abbildung 3.9: Zerlegungsmatrix $Z(\{x_1, \ldots, x_3\})$ zu $ext(isp) = \overline{z} \cdot isp_{off} + z \cdot isp_{on}$.

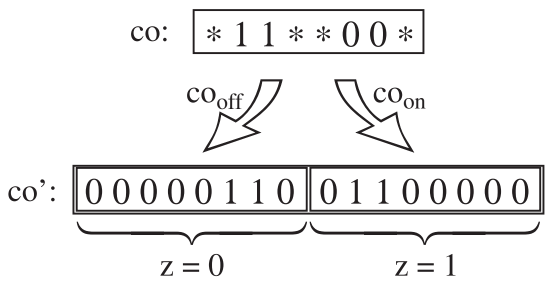


Abbildung 3.10: Bestimmung einer Zerlegungsmatrixzeile von $ext(f)$ aus der entsprechenden Zeile der Matrix zu f.

Ein ROBDD, der die Funktion $ext(isp) = \overline{z} \cdot isp_{off} + z \cdot isp_{on}$ repräsentiert, ist in Abbildung 3.8 angegeben. Die zugehörige Zerlegungsmatrix findet man in Abbildung 3.9. Die Zerlegungsmatrix zu $ext(isp)$ in Abbildung 3.9 geht aus der Zerlegungsmatrix zu isp (Abbildung 3.7) in einfacher Weise hervor:
Aus einer Zeile der Matrix zu isp (d.h. der Funktionstabelle zu einem Kofaktor $isp_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$) erhält man die entsprechende Zeile von $ext(isp)$, d.h. die Funktionstabelle zu

$$\begin{aligned} & ext(isp)_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}} = \\ = \; & (\overline{z} \cdot isp_{off} + z \cdot isp_{on})_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}} \\ = \; & (\overline{z} \cdot isp_{off\, x_1^{\epsilon_1} \dots x_p^{\epsilon_p}} + z \cdot isp_{on\, x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}), \end{aligned}$$

durch Nebeneinandersetzen der Funktionstabellen von $isp_{off\, x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ und $isp_{on\, x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ (siehe Abbildung 3.10).

Da ein Kofaktor $f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ einer (partiellen) Funktion f eineindeutig durch $(f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}})_{off}$ und $(f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}})_{on}$ beschrieben werden kann, gibt es eine eineindeutige Entsprechung zwischen Kofaktoren $f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ von f und Kofaktoren $ext(f)_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ von $ext(f)$. Da $ext(f)$ eine totale Funktion ist, gibt es wiederum (wie schon in Abschnitt 3.1.3.1 festgestellt wurde) eine eineindeutige Entsprechung zwischen den Kofaktoren $ext(f)_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ von $ext(f)$ und den „Verbindungsknoten“ (unterhalb einer Schnittlinie nach $x_1, \dots, x_p$) in einem ROBDD zu $ext(f)$, bei dem die Variablen $x_1, \dots, x_p$ in der Ordnung vor allen anderen Variablen stehen.

Beispiel 3.8 Im Beispiel der Funktion isp aus Abbildung 3.8 gibt es genau 4 „Verbindungsknoten“ und entsprechend genau 4 verschiedene Zeilenmuster in der Zerlegungsmatrix von $ext(isp)$ (Abbildung 3.9) und 4 verschiedene Zeilenmuster in der Zerlegungsmatrix von isp (Abbildung 3.7). Es ergeben sich also 4 Klassen der durch Zeilengleichheit in $Z(\{x_1, \dots, x_3\})$ induzierten Äquivalenzklasseneinteilung von $\{0,1\}^p$, nämlich $K_1 = \{(000), (011)\}$, $K_2 = \{(101), (110)\}$, $K_3 = \{(001), (010)\}$ und $K_4 = \{(100), (111)\}$.

Nach Bestimmung der $vz(\{x_1, \dots, x_p\}, f)$ verschiedenen Verbindungsknoten wird der „Kompatibilitätsgraph“ der Relation „$\sim$“ auf den Äquivalenzklassen $K_1, \dots, K_{vz(\{x_1, \dots, x_p\}, f)}$ aufgestellt. ($K_i \sim K_j$ gilt, wenn für $\epsilon \in K_i$ und $\delta \in K_j$ in der Zerlegungsmatrix zu f die Zeile mit dem Index ϵ in keiner Spalte eine 1 aufweist, in der die Zeile mit Index δ eine 0 aufweist oder umgekehrt.) Es gilt

$$\begin{array}{lll} & K_i \sim K_j & \\ \Longleftrightarrow & \text{für } \epsilon \in K_i, \delta \in K_j: & (f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}})_{on} \wedge (f_{x_1^{\delta_1}\ldots x_p^{\delta_p}})_{off} = 0 \textit{ und} \\ & & (f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}})_{off} \wedge (f_{x_1^{\delta_1}\ldots x_p^{\delta_p}})_{on} = 0. \\ \Longleftrightarrow & \text{für } \epsilon \in K_i, \delta \in K_j: & ext(f)_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p} z} \wedge ext(f)_{x_1^{\delta_1}\ldots x_p^{\delta_p}\overline{z}} = 0 \textit{ und} \\ & & ext(f)_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}\overline{z}} \wedge ext(f)_{x_1^{\delta_1}\ldots x_p^{\delta_p} z} = 0. \end{array}$$

Man erkennt, daß sich die Kompatibilitätsrelation direkt auf der Grundlage des ROBDD zu $ext(f)$ aufstellen läßt. Dieses Vorgehen hat gegenüber dem Aufstellen einer Relation auf $\{0,1\}^p$ den Vorteil, daß man bei einer kompakten Darstellung der partiellen Funktion f durch einen ROBDD zu $ext(f)$ auch eine kompakte Kompatibilitätsrelation erhält.
EKM wird dann gelöst durch Bestimmung einer Lösung für das Problem „Partition into Cliques“ auf diesem Kompatibilitätsgraphen. Man erhält eine Partition $\{PK_1, \ldots, PK_{vkk(\{x_1,\ldots,x_p\},f)}\}$ von $\{0,1\}^p$, wobei $\forall 1 \leq i \leq vkk(\{x_1,\ldots,x_p\},f)$ $PK_i = \bigcup_{j=1}^{l_i} K_{i_j}$*.

Beispiel 3.8 (Fortsetzung):
Es gilt $PK_1 = K_1 \cup K_2 = \{(000),(011),(101),(110)\}$, $PK_2 = K_3 \cup K_4 = \{(001),(010),(100),(111)\}$, $vkk(\{x_1,\ldots,x_3\},isp) = 2$.

Es bleibt noch das Problem, nach Lösung der Instanz von „Partition into Cliques“ die don't cares von f geeignet zu belegen. Die don't cares sind so zu belegen, daß bei der resultierenden Erweiterung f' von f für beliebige Elemente $\epsilon^{(i)}$ der Klassen K_i die Kofaktoren $f'_{x_1^{\epsilon_1^{(i_1)}}\ldots x_p^{\epsilon_p^{(i_1)}}}, \ldots, f'_{x_1^{\epsilon_1^{(i_{l_i})}}\ldots x_p^{\epsilon_p^{(i_{l_i})}}}$ alle gleich sind ($PK_i = \bigcup_{j=1}^{l_i} K_{i_j}$), so daß schließlich $vz(\{x_1,\ldots,x_p\},f') = vkk(\{x_1,\ldots,x_p\},f)$.

Beispiel 3.8 (Fortsetzung):
Bei Beispiel 3.8 sind in der Zerlegungsmatrix aus Abbildung 3.7 sämtliche Zeilen der Form

$$\boxed{\star\ 1\ 1\ \star\ \star\ 0\ 0\ \star}$$

und

$$\boxed{0\ \star\ \star\ 0\ 1\ \star\ \star\ 1}$$

durch ihre gemeinsame Erweiterung

$$\boxed{0\ 1\ 1\ 0\ 1\ 0\ 0\ 1}$$

zu ersetzen. Sämtliche Zeilen der Form

*Verwendet man eine *Heuristik* zur Lösung von „Partition into Cliques“, so kommt man natürlich evtl. zu mehr als $vkk(\{x_1,\ldots,x_p\},f)$ verschiedenen Klassen.

$$\boxed{1\ 0\ 0\ 1 \star\star\star\star}$$

und

$$\boxed{\star\star\star\star\ 0\ 1\ 1\ 0}$$

sind durch

$$\boxed{1\ 0\ 0\ 1\ 0\ 1\ 1\ 0}$$

zu ersetzen.

Bezeichnung 3.6 *Seien $co_1 \in S(D_1)$, ..., $co_l \in S(D_l)$ mit $D_1, \ldots, D_l \subseteq \{0,1\}^{n-p}$ mit der Eigenschaft, daß es kein Paar $i, j \in \{1, \ldots, l\}$ und $\epsilon \in D_i \cap D_j$ gibt, so daß $co_i(\epsilon) \neq co_j(\epsilon)$. Dann heißt die Funktion*

$$gem\_erw(\{co_1, \ldots, co_l\}) : \cup_{i=1}^{l} D_i \rightarrow \{0,1\}, gem\_erw(\{co_1, \ldots, co_l\})(\epsilon) = co_j(\epsilon)$$

für ein beliebiges co_j mit $\epsilon \in D_j$ gemeinsame Erweiterung *von $co_1, \ldots, co_l$.*

Bemerkung 3.4 *Seien $co_1 \in S(D_1)$, ..., $co_l \in S(D_l)$ mit $D_1, \ldots, D_l \subseteq \{0,1\}^{n-p}$ mit der Eigenschaft, daß es kein Paar $i, j \in \{1, \ldots, l\}$ und $\epsilon \in D_i \cap D_j$ gibt, so daß $co_i(\epsilon) \neq co_j(\epsilon)$. Sei $ext(co_i) = \overline{z} \cdot (co_i)_{off} + z \cdot (co_i)_{on}$ ($1 \leq i \leq l$). Dann ist*

$$ext(gem\_erw(\{co_1, \ldots, co_l\})) = \overline{z} \cdot \bigvee_{i=1}^{l} (co_i)_{off} + z \cdot \bigvee_{i=1}^{l} (co_i)_{on}.$$

Seien $\epsilon^{(j)}$ wiederum beliebige Elemente der Klassen K_j, $PK_i = \bigcup_{j=1}^{l_i} K_{i_j}$. Im ROBDD zu $ext(f)$ führt man die don't care–Belegung folgendermaßen durch: Für alle $1 \leq i \leq vkk(\{x_1, \ldots, x_p\}, f)$ werden die Verbindungsknoten, die durch $\epsilon^{(i_1)}, \ldots \epsilon^{(i_{l_i})}$ erreichbar sind, zusammen mit den Graphen deren Wurzel sie darstellen, ersetzt durch den ROBDD zu

$$ext(gem\_erw(\{f_{x_1^{\epsilon_1^{(i_1)}} \ldots x_p^{\epsilon_p^{(i_1)}}}, \ldots, f_{x_1^{\epsilon_1^{(i_{l_i})}} \ldots x_p^{\epsilon_p^{(i_{l_i})}}}\})).$$

Diesen ROBDD erhält man wie in der vorangegangen Bemerkung angegeben aus $ext(f)_{x_1^{\epsilon_1^{(i_1)}} \ldots x_p^{\epsilon_p^{(i_1)}}}, \ldots, ext(f)_{x_1^{\epsilon_1^{(i_{l_i})}} \ldots x_p^{\epsilon_p^{(i_{l_i})}}}$.

Beispiel 3.8 (Fortsetzung):
Bei Beispiel 3.8 sind im ROBDD (siehe Abbildung 3.8) der erste und der zweite „Verbindungsknoten von links (mit den dazugehörigen Untergraphen) zu ersetzen durch den ROBDD zu

$$ext(gem\_erw(\{isp_{x_1^0, x_2^0, x_3^0}, isp_{x_1^1, x_2^0, x_3^1}\})) = \overline{z} \cdot \overline{exor(x_1, x_2, x_3)} + z \cdot exor(x_1, x_2, x_3)$$

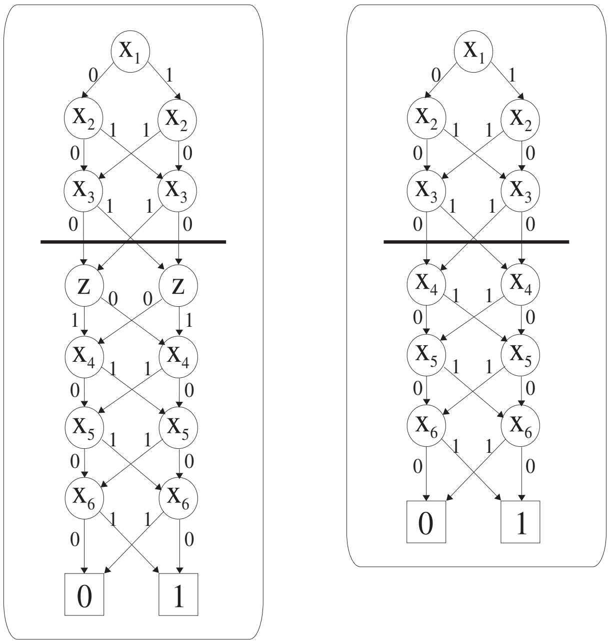


Abbildung 3.11: Auf der linken Seite ist der ROBDD zu der Funktion $ext(isp')$ nach don't care–Belegung dargestellt, auf der rechten Seite der ROBDD zu isp'_{on}.

und der dritte und vierte Verbindungsknoten (mit den dazugehörigen Untergraphen) zu ersetzen durch den ROBDD zu

$$ext(gem\_erw(\{isp_{x_1^0,x_2^0,x_3^1}, isp_{x_1^1,x_2^0,x_3^0}\})) = \overline{z} \cdot exor(x_1, x_2, x_3) + z \cdot \overline{exor(x_1, x_2, x_3)}.$$

Der ROBDD zu der resultierenden Funktion $ext(isp')$ ist in Abbildung 3.11 auf der linken Seite dargestellt. Der ROBDD auf der rechten Seite repräsentiert $isp'_{on} = exor(x_1, \ldots, x_6)$.

Nach der don't care–Belegung erhält man dann einen ROBDD zu einer Erweiterung $ext(f')$ mit $vz(\{x_1, \ldots, x_p\}, f') = vkk(\{x_1, \ldots, x_p\}, f)$ Verbindungsknoten. Da die eingesetzten gemeinsamen Erweiterungen nicht notwendigerweise total sind, kann das Ergebnis $ext(f')$ auch eine *partielle* Funktion darstellen. Benötigt man eine *totale* Erweiterung von f mit $vkk(\{x_1, \ldots, x_p\}, f)$ Verbindungsknoten, so kann man die verbleibenden don't cares beispielsweise alle mit dem gleichen Wert belegen.

Eine Näherungslösung für ZKM kann auch hier (wie im vorhergehenden Abschnitt angegeben) durch zweimaliges Lösen von EKM gewonnen werden.

3.2.4 Anwendung auf ROBDD–Minimierung

Das im vorangegangenen Abschnitt angegebene Verfahren minimiert die Anzahl der Verbindungsknoten unterhalb einer Schnittlinie nach den (in der Variablenordnung) ersten

p Variablen $x_1, \ldots, x_p$. Diese Tatsache wird in [CCM94] ausgenutzt, um bei vorgegebener Variablenordnung Erweiterungen partieller Funktionen zu finden, deren ROBDD–Größe möglichst gering ist. Wie bereits erwähnt erhält man nach don't care–Belegung im allgemeinen wieder eine partielle Funktion. Es bietet sich dann an, die verbleibenden don't cares zu nutzen, um die Anzahl der Verbindungsknoten auch unterhalb *anderer* Schnittlinien zu minimieren. Chang/Marek–Sadowska benutzen das beschriebene Verfahren, um der Reihe nach für $i = 1, \ldots, n-1$ die Anzahl der Verbindungsknoten bei Schnitt nach den Variablen $x_1, \ldots, x_i$ zu minimieren (hierbei wird o.B.d.A angenommen, daß für die Variablenordnung des ROBDD $x_1 <_{index} \ldots <_{index} x_n$ gilt). Ein Nachteil des Verfahrens liegt allerdings darin, daß es nur basierend auf einer fest vorgegebenen Variablenordnung arbeitet. In Abschnitt 3.5.2 wird eine Methode angegeben, die dieses Verfahren (unter Ausnutzung von Symmetrien partieller Funktionen) mit der Bestimmung guter Variablenordnungen kombiniert.

Unter anderem auch im Hinblick auf die rekursive Anwendung des Zerlegungsverfahrens ist es auch bei der Durchführung von Zerlegungen sinnvoll, nicht nur an *einer* Schnittlinie die ROBDD–Größe zu minimieren, sondern insgesamt eine Erweiterung der zu realisierenden Funktion mit einem möglichst kleinen ROBDD zu finden, um zu möglichst einfachen, gut zerlegbaren Funktionen zu kommen. Auch die Ausnutzung von don't cares bei der Suche nach Erweiterungen mit möglichst vielen Symmetrieeigenschaften (siehe Abschnitt 3.5) spielt für das Verfahren eine große Rolle.

3.2.5 Berechnung *partieller* Zerlegungs– und Zusammensetzungsfunktionen

Nachdem man die minimale Anzahl der benötigten Zerlegungsfunktionen bei Zerlegung hinsichtlich einer Variablenteilmenge $\{x_1, \ldots, x_p\}$ bestimmt hat, bleibt die Aufgabe, die konkrete Wahl der Zerlegungs– und Zusammensetzungsfunktionen zu treffen. Hierbei soll versucht werden, die don't care–Menge dieser Funktionen möglichst groß zu wählen, da don't cares bei der Logiksynthese Freiheiten liefern, die man ausnutzen kann, um möglichst gute Realisierungen zu erhalten.

Nach einer Herleitung von don't care–Mengen für Zerlegungs– und Zusammensetzungsfunktionen wird schließlich bewiesen, daß die hergeleiteten don't care–Mengen tatsächlich maximal sind, d.h. daß sie nicht mehr erweitert werden können.

Bei der Bestimmung von Zerlegungs– und Zusammensetzungsfunktionen bei der Zerlegung einer partiellen Funktion $f \in S(D_f)$ ($D_f \subseteq \{0,1\}^n$) hinsichtlich einer Variablenteilmenge $\{x_1, \ldots, x_p\}$ geht man aus von einer Partitionierung $PK = \{PK_1, \ldots, PK_v\}$ von $\{0,1\}^p$. Für die Mengen PK_i ($1 \leq i \leq v$) gilt

$$\epsilon \sim \delta \text{ für alle } \epsilon, \delta \in PK_i, \qquad (\star)$$

wobei

$$\begin{aligned} \epsilon \sim \delta \iff & \nexists (y_1, \ldots, y_{n-p}) \text{ mit} \\ & (\epsilon_1, \ldots, \epsilon_p, y_1, \ldots, y_{n-p}),\ (\delta_1, \ldots, \delta_p, y_1, \ldots, y_{n-p}) \in D_f, \\ & f(\epsilon_1, \ldots, \epsilon_p, y_1, \ldots, y_{n-p}) \neq f(\delta_1, \ldots, \delta_p, y_1, \ldots, y_{n-p}). \end{aligned}$$

Die Partitionierung $PK = \{PK_1, \ldots, PK_v\}$ muß (bei Anwendung einer Heuristik für „Partition into Cliques") nicht unbedingt die minimale Größe $v = vkk(\{x_1, \ldots, x_p\}, f)$ haben. Im folgenden wird jedoch angenommen, daß sie zumindest die folgende „lokale Optimalitätseigenschaft" hat: Für alle $i \neq j \in \{1, \ldots, v\}$ gilt: $\exists \epsilon \in PK_i, \delta \in PK_j$ mit $\epsilon \not\sim \delta$, d.h. man kann zwei Klassen PK_i und PK_j *nicht* vereinigen, ohne Eigenschaft $(\star)$ zu verletzen.

Die ROBDDs bdd_i zu den Funktionen $ext(gem\_erw(\{f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}} | (\epsilon_1, \ldots, \epsilon_p) \in PK_i\}))$ $(1 \leq i \leq v)$ werden auf Grundlage des ROBDD zu $ext(f)$ bestimmt wie im vorhergehenden Abschnitt beschrieben.

Analog zum Vorgehen bei totalen Funktionen wird nun zu jeder Menge PK_i (bzw. zu jedem bdd_i) ein eineindeutiger Code $(a_1^{(i)}, \ldots, a_r^{(i)})$ aus $\{0,1\}^r$ mit $r = \lceil \log(v) \rceil$ zugeordnet. Die Zusammensetzungsfunktion g erhält man im wesentlichen dadurch, daß man wie bei totalen Funktionen einen „Codebaum" über die ROBDDs bdd_1 bis bdd_v baut. Falls $v < 2^r$, dann sind einige Codes aus $\{0,1\}^r$ nicht vergeben. Ist $(\epsilon_1, \ldots, \epsilon_r)$ nicht als Code vergeben, dann ist für alle $(y_1, \ldots, y_{n-p})$ aus $\{0,1\}^{n-p}$ $(\epsilon_1, \ldots, \epsilon_r, y_1, \ldots, y_{n-p})$ in der don't care–Menge von g. Beim ROBDD zu $ext(g) = \overline{z} \cdot g_{off} + z \cdot g_{on}$ muß dann also der durch $(\epsilon_1, \ldots, \epsilon_r)$ erreichbare Knoten der $\boxed{0}$–Knoten sein.

Die ROBDDs der Zerlegungsfunktionen $\alpha_1, \ldots, \alpha_r$ ergeben sich ebenfalls ähnlich wie bei totalen Funktionen. Die Klassen PK_i $(1 \leq i \leq v)$ ergeben sich durch Vereinigung von Klassen $K_1, \ldots, K_{vz(\{x_1, \ldots, x_p\}, f)}$, wobei die Klassen K_j die Äquivalenzklassen sind, die durch Zeilengleichheit in der Zerlegungsmatrix zu $ext(f)$ hinsichtlich $\{x_1, \ldots, x_p\}$ induziert werden. Eine Zuordnung von Codes zu den Klassen PK_i ordnet auch den Klassen K_j eindeutige Werte zu. (Allerdings ist dann im allgemeinen verschiedenen (kompatiblen) Klassen K_j der gleiche Wert zugeordnet.) Es existiert eine eineindeutige Entsprechung zwischen den „Verbindungsknoten" bei Schnitt des ROBDD zu $ext(f)$ nach den Variablen $x_1, \ldots, x_p$ und diesen Klassen K_j. Zunächst erhält man analog zu den totalen Funktionen den ROBDD zu einer Zerlegungsfunktion α_i' durch Ersetzung der Verbindungsknoten im ROBDD zu $ext(f)$ durch die i-ten Bits der zugehörigen Codes und anschließendes Reduzieren des ROBDD.

Der einzige Unterschied bei partiellen Funktionen besteht darin, daß man nun gegebenenfalls aufgrund von don't cares in f eine *partielle* Zerlegungsfunktion $\alpha = (\alpha_1, \ldots, \alpha_r)$ bestimmen kann, deren Erweiterung $(\alpha_1', \ldots, \alpha_r')$ ist. Einige Elemente von $\{0,1\}^p$ können evtl. in die don't care–Menge von α aufgenommen werden, ohne daß sich die Beziehung

$$f(\epsilon_1, \ldots, \epsilon_n) = g(\alpha(\epsilon_1, \ldots, \epsilon_p), \epsilon_{p+1}, \ldots, \epsilon_n) \text{ für alle } (\epsilon_1, \ldots, \epsilon_n) \in D_f \qquad (\star)$$

ändert. Bei der Bestimmung einer don't care–Menge für α werden die beiden folgenden Fälle unterschieden:

1. Fall: $v < 2^r$

Betrachte ein $\epsilon \in \{0,1\}^p$, wobei für die Definitionsmenge $D(f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}) = \emptyset$ gilt (d.h. die Zerlegungsmatrixzeile von $Z(\{x_1,\ldots,x_p\})$ mit Index ϵ enthält nur $\star$). Es ist klar, daß man $\alpha(\epsilon)$ beliebig wählen kann, da für alle $(y_1,\ldots,y_{n-p}) \in \{0,1\}^{n-p}$ $(\epsilon_1,\ldots,\epsilon_p,y_1,\ldots,y_{n-p})$ ohnehin nicht in der Definitionsmenge D_f von f enthalten ist und Beziehung $(\star)$ ja nur für Elemente der Definitionsmenge von f gelten muß.

Alle $\epsilon \in \{0,1\}^p$ mit $D(f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}) = \emptyset$ (bzw. $ext(f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}) = 0$) sind in einer einzigen Klasse K_i enthalten. Falls es also ein solches ϵ gibt, kann man die zugehörige Klasse $dc\_set := K_i$ als don't care–Menge von α verwenden.

2. Fall: $v = 2^r$

In diesem Fall ist die Wahl der $\epsilon \in \{0,1\}^p$, die in die don't care–Menge von α aufgenommen werden können, etwas weniger eingeschränkt:

Betrachte für $1 \le i \le v$ die Funktionen $cof_{PK_i} := gem\_erw(\{f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}} | (\epsilon_1,\ldots,\epsilon_p) \in PK_i\})$. Für $\epsilon \in \{0,1\}^p$ gelte: Für alle $1 \le i \le v$ ist cof_{PK_i} eine Erweiterung von $f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}$. Die Zerlegungsmatrixzeile von $Z(\{x_1,\ldots,x_p\})$ mit Index ϵ hat also folgende Eigenschaft: Falls die Zeile in einer bestimmten Spalte eine 1 (0) enthält, so enthält die Matrix, die aus $Z(\{x_1,\ldots,x_p\})$ durch Bestimmung der gemeinsamen Erweiterungen hervorgeht (vgl. Seite 121), in dieser Spalte nur 1 (0).

Dann kann man ϵ in die don't care–Menge von α aufnehmen, da für alle $(y_1,\ldots,y_{n-p})$ mit $(\epsilon_1,\ldots,\epsilon_p,y_1,\ldots,y_{n-p}) \in D_f$ und *jedes* $(a_1,\ldots,a_r) \in \{0,1\}^r$ gilt

1.
$$(a_1,\ldots,a_r,y_1,\ldots,y_{n-p}) \in D(g),$$
da $v = 2^r$ und es daher für alle $(a_1,\ldots,a_r) \in \{0,1\}^r$ ein PK_i gibt mit $\alpha'(PK_i) = (a_1,\ldots,a_r)$. Da cof_{PK_i} Erweiterung von $f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}$ ist, folgt aus $(\epsilon_1,\ldots,\epsilon_p,y_1,\ldots,y_{n-p}) \in D_f$ dann $(y_1,\ldots,y_{n-p}) \in D(cof_{PK_i})$ und damit $(a_1,\ldots,a_r,y_1,\ldots,y_{n-p}) \in D(g)$.

2.
$$\begin{aligned} & g(a_1,\ldots,a_r,y_1,\ldots,y_{n-p}) = \\ &= cof_{PK_i}(y_1,\ldots,y_{n-p}) \quad (\alpha'(PK_i) = (a_1,\ldots,a_r)) \\ &= f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}}(y_1,\ldots,y_{n-p}), \qquad \text{da } cof_{PK_i} \text{ Erweiterung von } f_{x_1^{\epsilon_1}\ldots x_p^{\epsilon_p}} \\ &= f(\epsilon_1,\ldots,\epsilon_p,y_1,\ldots,y_{n-p}). \end{aligned}$$

Für $(\epsilon_1,\ldots,\epsilon_p,y_1,\ldots,y_{n-p}) \in D_f$ ist also der Wert $\alpha(\epsilon_1,\ldots,\epsilon_p)$ unerheblich für das Ergebnis von $g(\alpha(\epsilon_1,\ldots,\epsilon_p),y_1,\ldots,y_{n-p})$ und ϵ kann folglich in die don't care–Menge von α aufgenommen werden.

Die don't care–Menge von α wird als die Menge aller $\epsilon \in \{0,1\}^p$ mit der obigen Eigenschaft bestimmt. Sei $\{K_1,\ldots,K_{vz(\{x_1,\ldots,x_p\},f)}\}$ die durch Zeilengleichheit in

$Z(\{x_1, \ldots, x_p\})$ induzierte Äquivalenzklasseneinteilung. Die don't care–Menge *dc_set* für α erhält man also durch Vereinigen aller Klassen K_i, bei denen *für alle* $1 \leq j \leq v$ gilt: cof_{PK_j} ist Erweiterung von $f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}$ für ein beliebiges Element $\epsilon \in K_i$ (und damit für alle Elemente $\epsilon \in K_i$). Man erhält diese Klassen, indem man bei allen Klassen K_i $(1 \leq i \leq vz(\{x_1, \ldots, x_p\}, f))$ testet, ob für alle $1 \leq j \leq v$ gilt:

$$(f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}})_{on} \cdot \overline{(cof_{PK_j})_{on}} = 0$$

und

$$(f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}})_{off} \cdot \overline{(cof_{PK_j})_{off}} = 0.$$

für ein beliebiges Element $\epsilon \in K_i$.

Bei der Logiksynthese ist es von Vorteil, Funktionen mit möglichst vielen don't cares zu haben. Es stellt sich nun die Frage, ob die obige Definition für Zerlegungs– und Zusammensetzungsfunktionen zu Funktionen mit einer maximalen Anzahl von don't cares führt. Der folgende Satz besagt, daß die Anzahl der don't cares in den oben definierten Zerlegungs– und Zusammensetzungsfunktionen in folgendem Sinn maximal ist: Erweitert man die definierte don't care–Menge der Zerlegungsfunktion α oder der Zusammensetzungsfunktion g um ein beliebiges Element, so gibt es Erweiterungen α'' bzw. g'' von α bzw. g, so daß

$$g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \epsilon_{p+1}, \ldots, \epsilon_n) \neq f(\epsilon_1, \ldots, \epsilon_n)$$

für ein $(\epsilon_1, \ldots, \epsilon_n) \in D_f$.
Man darf die don't care–Mengen also *nicht* noch weiter vergrößern.

In den Voraussetzungen des Satzes sind zunächst die oben angegebenen Definitionen der partiellen Zerlegungs– und Zusammensetzungsfunktion nochmals präzise festgehalten, um im Beweis auf die gegebenen Bezeichnungen zurückgreifen zu können.

Satz 3.7 *Sei $f \in S(D_f)$ $(D_f \subseteq \{0,1\}^n)$. f wird hinsichtlich $\{x_1, \ldots, x_p\}$ zerlegt. Sei $PK = \{PK_1, \ldots, PK_v\}$ eine Partition von $\{0,1\}^p$, wobei für die Mengen PK_i $(1 \leq i \leq v)$ gilt $\epsilon \sim \delta$ für alle $\epsilon, \delta \in PK_i$. $\{PK_1, \ldots, PK_v\}$ hat die folgende „lokale Optimalitätseigenschaft": Für alle $i \neq j \in \{1, \ldots, v\}$ gilt: $\exists \epsilon \in PK_i, \delta \in PK_j$ mit $\epsilon \not\sim \delta$. Für alle $1 \leq i \leq v$ sei $cof_{PK_i} = gem\_erw(\{f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}} | (\epsilon_1, \ldots, \epsilon_p) \in PK_i\})$. Sei $r = \lceil \log(v) \rceil$.
Bei $v = 2^r$ sei*

$$dc\_set = \{\epsilon \in \{0,1\}^p \mid \forall 1 \leq i \leq v \textit{ ist } cof_{PK_i} \textit{ Erweiterung von } f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}\}$$

und bei $v < 2^r$ sei

$$dc\_set = \{\epsilon \in \{0,1\}^p \mid (f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}})_{on} = (f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}})_{off} = 0.\}.$$

Sei $D_\alpha = \{0,1\}^p \setminus dc\_set$.[2]
Sei die Zerlegungsfunktion $\alpha : D_\alpha \to \{0,1\}^r$ *definiert durch*

$$\alpha(PK_i \setminus dc\_set) = (a_1^{(i)}, \ldots, a_r^{(i)})\; (1 \leq i \leq v)$$

$$mit\ (a_1^{(i)}, \ldots, a_r^{(i)}) \neq (a_1^{(j)}, \ldots, a_r^{(j)}) \forall 1 \leq i, j \leq v\ mit\ i \neq j.$$

Sei $D_g = \{(a_1^{(i)}, \ldots, a_r^{(i)}, y_1, \ldots, y_{n-p}) \mid (a_1^{(i)}, \ldots, a_r^{(i)}) \in Im(\alpha)$ *und* $(y_1, \ldots, y_{n-p}) \in D(cof_{PK_i})\}$.
Die Zusammensetzungsfunktion g *ist definiert durch*

$$g : D_g \to \{0,1\},\ \ g(a_1^{(i)}, \ldots, a_r^{(i)}, y_1, \ldots, y_{n-p}) = cof_{PK_i}(y_1, \ldots, y_{n-p}).$$

Dann sind D_α *und* D_g *in folgendem Sinn minimal:*
Für jede echte Teilmenge $D'_\alpha \subset D_\alpha$ *bzw.* $D'_g \subset D_g$ *gilt:*
Zu $\alpha' : D'_\alpha \to \{0,1\}^r, \alpha|_{D'_\alpha} = \alpha'$ *gibt es eine totale Erweiterung* $\alpha'' : \{0,1\}^p \to \{0,1\}^r$ *von* α', *eine totale Erweiterung* $g'' : \{0,1\}^{r+n-p} \to \{0,1\}$ *von* g *und mindestens ein* $\epsilon \in D_f$ *mit*

$$f(\epsilon_1, \ldots, \epsilon_n) \neq g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \epsilon_{p+1}, \ldots, \epsilon_n).$$

Zu $g' : D'_g \to \{0,1\}, g|_{D'_g} = g'$ *gibt es eine totale Erweiterung* $g'' : \{0,1\}^{r+n-p} \to \{0,1\}$ *von* g', *eine totale Erweiterung* $\alpha'' : \{0,1\}^p \to \{0,1\}^r$ *von* α *und mindestens ein* $\epsilon \in D_f$ *mit*

$$f(\epsilon_1, \ldots, \epsilon_n) \neq g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \epsilon_{p+1}, \ldots, \epsilon_n).$$

Satz 3.7 besagt also, daß sich bei den definierten Zerlegungs– und Zusammensetzungsfunktionen α und g die don't care–Mengen *nicht* mehr erweitern lassen. Die don't care–Mengen sind maximal gewählt.

Beweis:

Zusammensetzungsfunktion: Wähle $\epsilon \in D_g$ beliebig. (Falls $D_g = \emptyset$, ist die Aussage ohnehin bewiesen.)
Z.z.: Es gibt $g'' : \{0,1\}^{n-p+r} \to \{0,1\}$ mit $g''|_{D_g \setminus \{\epsilon\}} = g|_{D_g \setminus \{\epsilon\}}$, $\alpha'' : \{0,1\}^p \to \{0,1\}^r$ mit $\alpha''|_{D_\alpha} = \alpha$, und $(\delta_1, \ldots, \delta_p) \in \{0,1\}^p$, so daß $(\delta_1, \ldots, \delta_p, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) \in D_f$ und $g''(\epsilon_1, \ldots, \epsilon_r, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) = g''(\alpha''(\delta_1, \ldots, \delta_p), \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) \neq f(\delta_1, \ldots, \delta_p, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r})$.

[2]Ist $v > 1$, so ist für alle PK_i wegen der „lokalen Optimalitätseigenschaft" von PK $PK_i \setminus dc\_set \neq \emptyset$. Es gilt nämlich für alle $\epsilon \in dc\_set$ $\epsilon \sim \delta$ für alle $\delta \in \{0,1\}^p$. Wäre $PK_i \setminus dc\_set = \emptyset$, so wäre $PK_i \subseteq dc\_set$ und für beliebiges $1 \leq j \leq v$ mit $j \neq i$ wäre $\epsilon \sim \delta$ für alle $\epsilon \in PK_i$ und $\delta \in PK_j$. Folglich wäre die lokale Optimalitätseigenschaft von PK verletzt.

$v > 1$: Da $\epsilon \in D_g$, gibt es $PK_i \in \{PK_1, \ldots, PK_v\}$ mit $\alpha^{-1}(\epsilon_1, \ldots, \epsilon_r) = PK_i \backslash dc\_set$. Außerdem muß es ein $\delta \in PK_i$ geben, so daß $(\delta_1, \ldots, \delta_p, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) \in D_f$, denn sonst wäre $(\epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) \notin D(cof_{PK_i})$ und somit nicht $\epsilon \in D_g$. Falls $\delta \in dc\_set$, wähle $\alpha''(\delta) = (\epsilon_1, \ldots, \epsilon_r)$. Wählt man $g''(\epsilon_1, \ldots, \epsilon_r, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) = \overline{f(\delta_1, \ldots, \delta_p, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r})}$, dann gilt

$$\begin{aligned} g''(\alpha''(\delta_1, \ldots, \delta_p), \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) &= g''(\epsilon_1, \ldots, \epsilon_r, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r}) \\ &= \overline{f(\delta_1, \ldots, \delta_p, \epsilon_{r+1}, \ldots, \epsilon_{n-p+r})}. \end{aligned}$$

$v = 1$: Es gilt $r = 0$ und $f(x_1, \ldots, x_p, y_1, \ldots, y_{n-p}) = g(y_1, \ldots, y_{n-p})$ für $(x_1, \ldots, x_p, y_1, \ldots, y_{n-p}) \in D_f$.
Es muß $(\delta_1, \ldots, \delta_p)$ geben mit $(\delta_1, \ldots, \delta_p, \epsilon_1, \ldots, \epsilon_{n-p}) \in D_f$, da sonst $(\epsilon_1, \ldots, \epsilon_{n-p}) \notin D_g$. Wählt man $g''(\epsilon_1, \ldots, \epsilon_{n-p}) = \overline{f(\delta_1, \ldots, \delta_p, \epsilon_1, \ldots, \epsilon_{n-p})}$, so gilt offensichtlich $g''(\epsilon_1, \ldots, \epsilon_{n-p}) \neq f(\delta_1, \ldots, \delta_p, \epsilon_1, \ldots, \epsilon_{n-p})$.

Zerlegungsfunktion: Wähle $\epsilon \in D_\alpha$ beliebig. (Falls $D_\alpha = \emptyset$, ist die Aussage ohnehin bewiesen.)
Z.z.: Es gibt $\alpha'' : \{0,1\}^p \to \{0,1\}^r$ mit $\alpha''|_{D_\alpha \backslash \{\epsilon\}} = \alpha|_{D_\alpha \backslash \{\epsilon\}}$, $g'' : \{0,1\}^{r+n-p} \to \{0,1\}$ mit $g''|_{D_g} = g$ und $(\delta_1, \ldots, \delta_{n-p}) \in \{0,1\}^{n-p}$, so daß $(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p}) \in D_f$ und $g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \delta_1, \ldots, \delta_{n-p}) \neq f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})$.

$v < 2^r$: Es muß $(\delta_1, \ldots, \delta_{n-p})$ geben mit $(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p}) \in D_f$, da sonst $(\epsilon_1, \ldots, \epsilon_p) \notin D_\alpha$.
Sei $(a_1, \ldots, a_r) \notin Im(\alpha)$.
Wähle $\alpha''(\epsilon) = (a_1, \ldots, a_r)$.
$(a_1, \ldots, a_r, \delta_1, \ldots, \delta_{n-p}) \notin D_g$.
Wähle $g''(a_1, \ldots, a_r, \delta_1, \ldots, \delta_{n-p}) = \overline{f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})}$.

$$\begin{aligned} \Longrightarrow \quad g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \delta_1, \ldots, \delta_{n-p}) &= g''(a_1, \ldots, a_r, \delta_1, \ldots, \delta_{n-p}) \\ &= \overline{f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})}. \end{aligned}$$

$v = 2^r$: Wegen $\epsilon \in D_\alpha$ gibt es ein $PK_i \in PK$, so daß cof_{PK_i} keine Erweiterung von $f_{x_1^{\epsilon_1}, \ldots, x_p^{\epsilon_p}}$ ist. Es gibt also ein $\delta \in \{0,1\}^{n-p}$, so daß $(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p}) \in D_f$ und $\delta \notin D(cof_{PK_i})$ oder $cof_{PK_i}(\delta_1, \ldots, \delta_{n-p}) \neq f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})$.
Sei $\alpha(PK_i \setminus dc\_set) = (a_1^{(i)}, \ldots, a_r^{(i)})$.
Falls $\delta \notin D(cof_{PK_i})$ definiere

$$g''(a_1^{(i)}, \ldots, a_r^{(i)}, \delta_1, \ldots, \delta_{n-p}) = \overline{f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})},$$

ansonsten gilt ebenfalls

$$\begin{aligned} g''(a_1^{(i)}, \ldots, a_r^{(i)}, \delta_1, \ldots, \delta_{n-p}) &= g(a_1^{(i)}, \ldots, a_r^{(i)}, \delta_1, \ldots, \delta_{n-p}) \\ &= cof_{PK_i}(\delta_1, \ldots, \delta_{n-p}) \\ &= \overline{f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p})}. \end{aligned}$$

Wähle dann $\alpha''(\epsilon_1, \ldots, \epsilon_p) = (a_1^{(i)}, \ldots, a_r^{(i)})$, so daß

$$\begin{aligned} g''(\alpha''(\epsilon_1, \ldots, \epsilon_p), \delta_1, \ldots, \delta_{n-p}) &= \underline{g''(a_1^{(i)}, \ldots, a_r^{(i)}, \delta_1, \ldots, \delta_{n-p})} \\ &= f(\epsilon_1, \ldots, \epsilon_p, \delta_1, \ldots, \delta_{n-p}). \end{aligned}$$

$\square$

Bemerkung 3.5 *Satz 3.7 besagt, daß man bei der angegebenen Definition von Zerlegungs– und Zusammensetzungsfunktionen die don't care–Menge nicht mehr erweitern darf. Es ist allerdings zu beachten, daß in Satz 3.7 α als Funktion in mehreren Ausgängen definiert wurde, d.h. für alle α_i ($1 \leq i \leq r$) die gleiche Definitionsmenge $D_{\alpha_i} = D_\alpha$ vorausgesetzt wurde. Geht man von dieser Voraussetzung ab, so können die don't care–Mengen von α_i gegebenenfalls noch erweitert werden.*

Beispiel 3.9 Abbildung 3.12 zeigt eine Zerlegungsmatrix hinsichtlich $\{x_1, x_2, x_3\}$. Eine mögliche Lösung von „Partition into Cliques“ lautet

$$\{\underbrace{\{(000), (001)\}}_{PK_1}, \underbrace{\{(010), (011)\}}_{PK_2}, \underbrace{\{(100), (101)\}}_{PK_3}, \underbrace{\{(110), (111)\}}_{PK_4}\}.$$

Die Kodierung der Klassen PK_i ($1 \leq i \leq 4$) wird so gewählt, daß $\alpha(PK_1) = (00)$, $\alpha(PK_2) = (01)$, $\alpha(PK_3) = (10)$ und $\alpha(PK_4) = (11)$. Nach Satz 3.7 darf man bei $\alpha = (\alpha_1, \alpha_2)$ keine don't care–Stellen hinzufügen. Wie man leicht sieht, ist es jedoch möglich, sowohl für α_1 als auch für α_2 $D(\alpha_i) = \{0,1\}^3 \setminus \{(000)\}$ ($i = 1, 2$) zu definieren. Es ist aber *nicht* möglich, *gleichzeitig* für α_1 *und* α_2 (000) als don't care–Stelle festzulegen, da $f(00011) \neq f(11111)$.

3.3 k–seitige Zerlegungen

In diesem Abschnitt wird gezeigt, wie man zweiseitige Zerlegungen (oder allgemein k–seitige Zerlegungen mit $k \geq 2$) auf eine Folge von einseitigen Zerlegungen zurückführen kann.

k–seitige Zerlegungen mit $k > 2$ spielen eine Rolle bei der Ausnutzung von Symmetrien Boolescher Funktionen (vergleiche auch Abschnitt 3.5). Ist eine Boolesche Funktion f symmetrisch in einer Partition $\{\mu_1, \ldots, \mu_k\}$ der Menge $X = \{x_1, \ldots, x_n\}$ ihrer Eingangsvariablen und sind die Mengen μ_i ausreichend groß, so wird im implementierten Logiksyntheseverfahren f k–seitig hinsichtlich der Variablenpartition $\{\mu_1, \ldots, \mu_k\}$ zerlegt. Es gibt k Zerlegungsfunktionen (mit mehreren Ausgängen), die Zwischenergebnisse auf den Mengen μ_1 bis μ_k berechnen. In Kapitel 4 wird beispielsweise ein partieller Multiplizierer vorgestellt, bei dem k–seitige Zerlegungen hinsichtlich einer Partition in Mengen symmetrischer Variablen ausgenutzt werden.

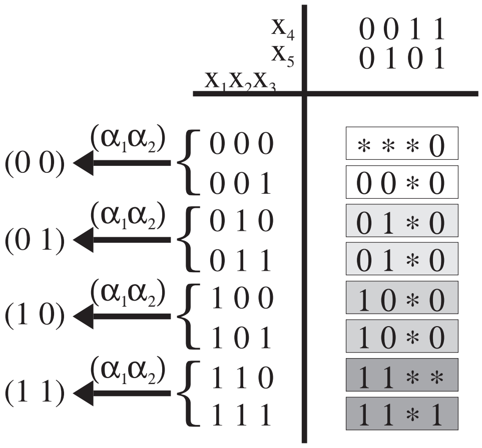


Abbildung 3.12: Zerlegungsmatrix zu Beispiel 3.9.

Führt man zweiseitige Zerlegungen *partieller* Boolescher Funktionen hinsichtlich einer Variablenaufteilung $\{X^{(1)}, X^{(2)}\}$ zurück auf eine Folge zweier einseitiger (kommunikationsminimaler) Zerlegungen, so ist die Anzahl der Zerlegungsfunktionen, die von $X^{(1)}$ bzw. $X^{(2)}$ abhängen, davon abhängig, ob man zuerst hinsichtlich $X^{(1)}$ oder zuerst hinsichtlich $X^{(2)}$ zerlegt. Dies wurde schon anhand von Beispielen zur (heuristischen) Lösung des Problems ZKM für zweiseitige Zerlegungen durch eine Folge zweier Lösungen des Problems EKM für einseitige Zerlegungen in Abschnitt 3.2.2 deutlich.

Auch wenn die ursprüngliche Funktion f *total* ist, kann bei der Zerlegung hinsichtlich einer Variablenteilmenge $X^{(1)}$ die Zusammensetzungsfunktion partiell sein (wenn $vz(X^{(1)}, f)$ keine Zweierpotenz ist). Aus diesem Grund stellt sich die Frage, ob sich zweiseitige Zerlegungen (oder allgemein k–seitige Zerlegungen ($k \geq 2$)) totaler Funktionen ohne Probleme auf eine Folge von einseitigen Zerlegungen zurückführen lassen bzw. ob die Anzahl der benötigten Zerlegungsfunktionen auf einer bestimmten Variablenteilmenge dabei unabhängig ist von der Reihenfolge der einseitigen Zerlegungen. Aus Satz 3.9 geht hervor, daß dies tatsächlich der Fall ist.

Definition 3.9 (k–seitige Zerlegung)
*Eine k–**seitige (disjunkte) Zerlegung** ($k \geq 2$) einer Booleschen Funktion $f \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_n$ hinsichtlich einer Partition $\{X^{(1)}, \ldots, X^{(k)}\}$ der Menge der Eingangsvariablen ist eine Darstellung von f in der Form*

$$f(x_1, \ldots, x_n) =$$
$$g(\alpha_1^{(1)}(x_{1_1}, \ldots, x_{1_{p_1}}), \ldots, \alpha_{r_1}^{(1)}(x_{1_1}, \ldots, x_{1_{p_1}}), \ldots, \alpha_1^{(k)}(x_{k_1}, \ldots, x_{k_{p_k}}), \ldots, \alpha_{r_k}^{(k)}(x_{k_1}, \ldots, x_{k_{p_k}})),$$

wobei $X^{(i)} = \{x_{i_1}, \ldots, x_{i_{p_i}}\}$, $p_i \geq 1$ für $1 \leq i \leq k$.

Satz 3.8 *Sei f eine Funktion aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$. Dann gibt es genau dann eine k–seitige Zerlegung von f hinsichtlich der Partition $\{X^{(1)}, \ldots, X^{(k)}\}$ der Menge der Eingangsvariablen von der Form*

$$f(x_1, \ldots, x_n) =$$
$$g(\alpha_1^{(1)}(x_{1_1}, \ldots, x_{1_{p_1}}), \ldots, \alpha_{r_1}^{(1)}(x_{1_1}, \ldots, x_{1_{p_1}}), \ldots, \alpha_1^{(k)}(x_{k_1}, \ldots, x_{k_{p_k}}), \ldots, \alpha_{r_k}^{(k)}(x_{k_1}, \ldots, x_{k_{p_k}})),$$

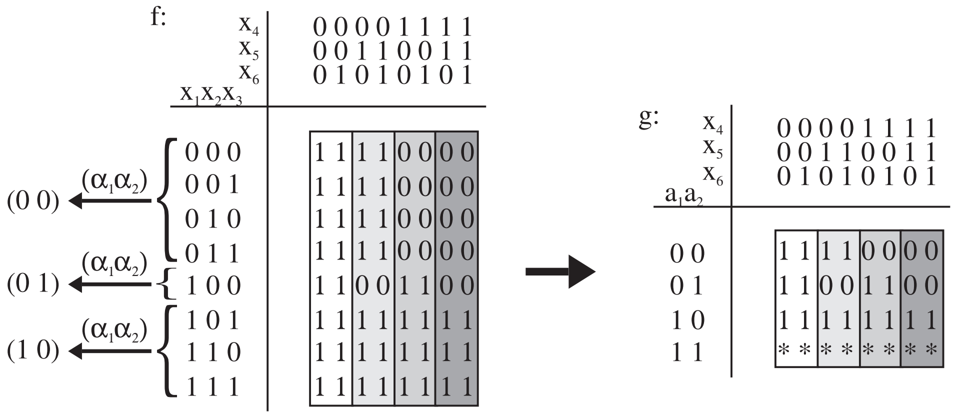
wenn

$$r_i \geq \log(vz(X^{(i)}, f)) \; \forall 1 \leq i \leq k.$$

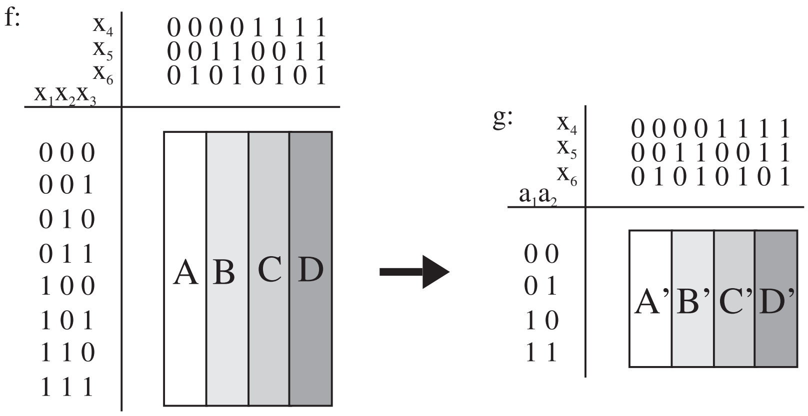
Satz 3.8 ergibt sich aus Satz 3.9, aus dem darüber hinaus folgt, daß man k–seitige Zerlegungen auf eine Folge von k einseitigen Zerlegungen zurückführen kann, so daß die Reihenfolge der einseitigen Zerlegungen für das Ergebnis *nicht* relevant ist. Zurückzuführen ist dies auf die spezielle Struktur der sich ergebenden don't care–Mengen bei den einzelnen Zusammensetzungsfunktionen.

Im nun folgenden Satz 3.9 ist zunächst in den Voraussetzungen nochmals genau festgehalten, wie bei einer Folge von k einseitigen Zerlegungen die Zerlegungen auf den einzelnen Stufen (nach dem Verfahren aus Satz 3.7) auseinander hervorgehen. Der Beweis des Satzes erfolgt induktiv und ist für den allgemeinen Fall von k–seitigen Zerlegungen relativ technisch. Daher soll vorher die Idee, die spezielle Struktur der resultierenden don't care–Mengen auszunutzen, anhand eines Beispiels für eine dreiseitige Zerlegung verdeutlicht werden:

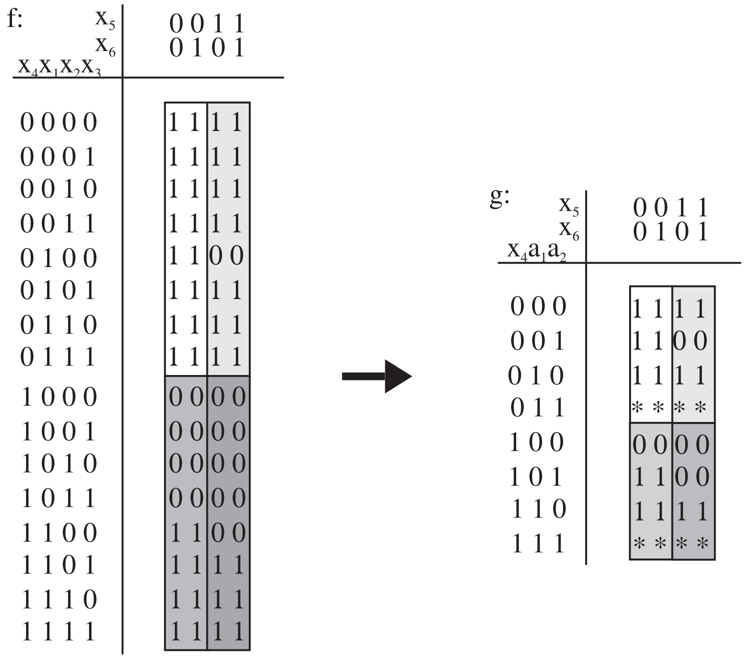
Beispiel 3.10 Eine Funktion $f \in B_6$ soll hinsichtlich der Partition $\{\{x_1, x_2, x_3\}, \{x_5, x_6\}, \{x_4\}\}$ zerlegt werden. f ist durch folgende Zerlegungsmatrix hinsichtlich $\{x_1, x_2, x_3\}$ definiert:



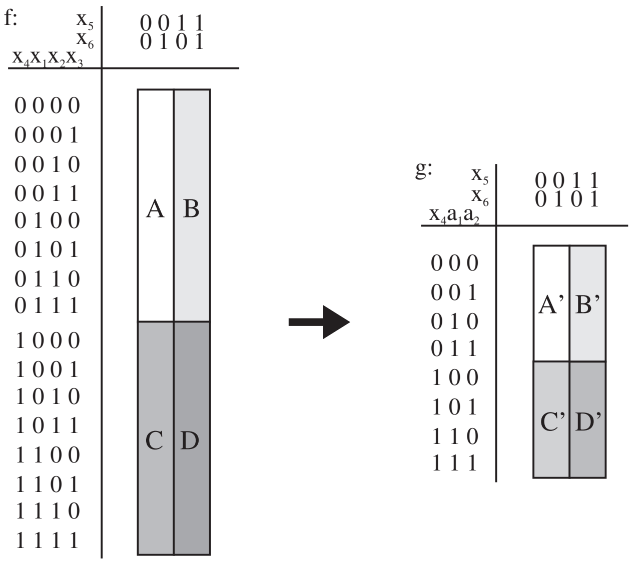
Die Zerlegung soll auf eine Folge von einseitigen Zerlegungen zurückgeführt werden. Im ersten Schritt wird hinsichtlich $\{x_1, x_2, x_3\}$ zerlegt. Die entsprechende Zerlegungsmatrix der Zusammensetzungsfunktion g ist in der obigen Abbildung neben der Zerlegungsmatrix für f dargestellt. Es ist klar, daß die Anzahlen verschiedener Spalten in den Zerlegungsmatrizen von f und g gleich sein müssen. Die Zerlegungsmatrizen lassen sich schematisch wie folgt darstellen (es gibt jeweils 4 Blöcke von 2 gleichen Spalten):



Die Zusammensetzungsfunktion wird nun weiter hinsichtlich $\{x_5, x_6\}$ zerlegt. Zerlegungsmatrizen für die Zerlegung sowohl von f als auch von g hinsichtlich $\{x_5, x_6\}$ (hier mit x_5 und x_6 als Variablen der Spalten) sind in der folgenden Abbildung angegeben:



Die Anzahl der verschiedenen Spalten bei der Zerlegung von g hinsichtlich $\{x_5, x_6\}$ stimmt mit der Anzahl der verschiedenen Spalten bei der Zerlegung von f hinsichtlich $\{x_5, x_6\}$ überein. Dies ergibt sich aus der folgenden schematischen Darstellung der beiden Zerlegungsmatrizen:



Da bei der Zerlegungsmatrix für g *in allen Spalten die don't cares an den gleichen Stellen* stehen, sind zwei Spalten genau dann kompatibel, wenn sie gleich sind. Verschiedene Spalten können also nicht durch Belegung von don't cares zur Übereinstimmung gebracht werden. Bei dem bereits beschriebenen Zerlegungsverfahren werden daher auch *keine don't cares bei der Bestimmung gemeinsamer Erweiterungen belegt.* Die Anzahl der Zerlegungsfunktionen bei Zerlegung von g hinsichtlich $\{x_5, x_6\}$ stimmt folglich mit der Anzahl der Zerlegungsfunktionen bei der entsprechenden Zerlegung von f überein.
Diese Eigenschaften gelten allgemein für beliebige k–seitige Zerlegungen.

Satz 3.9 *Sei f eine Funktion aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$. Sei $\{X^{(1)}, \ldots, X^{(k)}\}$ eine Partition der Menge $\{x_1, \ldots, x_n\}$ der Eingangsvariablen. $X^{(i)} = \{x_{i_1}, \ldots, x_{i_{p_i}}\}$. Sei o.B.d.A. $x_{i_j} = x_{\sum_{l=1}^{i-1} p_l + j}$ für alle $1 \leq i \leq k$, $1 \leq j \leq p_i$. Sei $g^{(0)} := f$ und für $1 \leq i \leq k$ gehe $g^{(i)}$ aus $g^{(i-1)}$ durch folgendes Verfahren (vgl. Satz 3.7) hervor:*
Sei $PK^{(i)} = \{PK_1^{(i)}, \ldots, PK_{v_i}^{(i)}\}$ eine Partition von $\{0,1\}^{p_i}$, wobei für die Mengen $PK_j^{(i)}$ $(1 \leq j \leq v_i)$ gilt $\epsilon \sim_i \delta$ für alle $\epsilon, \delta \in PK_j^{(i)}$. ($\epsilon \sim_i \delta$ gilt genau dann, wenn in der Zerlegungsmatrix $Z^{(i)}(X^{(i)})$ von $g^{(i-1)}$ die Zeile mit dem Index ϵ in keiner Spalte eine 1 *aufweist, in der die Zeile mit Index δ eine* 0 *aufweist oder umgekehrt.) Für alle $1 \leq j \leq v_i$ sei $cof_{PK_j^{(i)}} = gem\_erw(\{g^{(i-1)}_{x_{i_1}^{\epsilon_1} \ldots x_{i_{p_i}}^{\epsilon_{p_i}}} | (\epsilon_1, \ldots, \epsilon_{p_i}) \in PK_j^{(i)}\})$. Sei $r_i = \lceil \log(v_i) \rceil$.*
Bei $v_i = 2^{r_i}$ sei

$$dc\_set = \{\epsilon \in \{0,1\}^{p_i} \mid cof_{PK_j^{(i)}} \text{ Erweiterung von } g^{(i-1)}_{x_{i_1}^{\epsilon_1} \ldots x_{i_{p_i}}^{\epsilon_{p_i}}} \forall 1 \leq j \leq v_i\}$$

und bei $v < 2^{r_i}$ sei

$$dc\_set = \{\epsilon \in \{0,1\}^{p_i} \mid (g^{(i-1)}_{x_{i_1}^{\epsilon_1} \ldots x_{i_{p_i}}^{\epsilon_{p_i}}})_{on} = (g^{(i-1)}_{x_{i_1}^{\epsilon_1} \ldots x_{i_{p_i}}^{\epsilon_{p_i}}})_{off} = 0\}.$$

Sei $D_{\alpha^{(i)}} = \{0,1\}^{p_i} \setminus dc\_set$.
Sei die Zerlegungsfunktion $\alpha^{(i)} : D_{\alpha^{(i)}} \rightarrow \{0,1\}^{r_i}$ *definiert durch*

$$\alpha^{(i)}(PK_j^{(i)} \setminus dc\_set) = (a_1^{(j)}, \ldots, a_{r_i}^{(j)})\ (1 \leq j \leq v_i)$$

$$mit\ (a_1^{(l)}, \ldots, a_{r_i}^{(l)}) \neq (a_1^{(m)}, \ldots, a_{r_i}^{(m)})\ \forall 1 \leq l, m \leq v_i\ mit\ l \neq m.$$

Sei $D_{g^{(i)}} = \{(y_1, \ldots, y_{\sum_{j=1}^{i-1} r_j}, a_1^{(j)}, \ldots, a_{r_i}^{(j)}, z_1, \ldots, z_{\sum_{j=i+1}^{k} p_j}) \mid (a_1^{(j)}, \ldots, a_{r_i}^{(j)}) \in Im(\alpha^{(i)})$ *und* $(y_1, \ldots, y_{\sum_{j=1}^{i-1} r_j}, z_1, \ldots, z_{\sum_{j=i+1}^{k} p_j}) \in D(cof_{PK_j^{(i)}})\}$.
Die Zusammensetzungsfunktion $g^{(i)}$ *sei definiert durch*
$g^{(i)} : D_{g^{(i)}} \rightarrow \{0,1\}$,
$g^{(i)}(y_1, \ldots, y_{\sum_{j=1}^{i-1} r_j}, a_1^{(j)}, \ldots, a_{r_i}^{(j)}, z_1, \ldots, z_{\sum_{j=i+1}^{k} p_j}) =$
$= cof_{PK_j^{(i)}}(y_1, \ldots, y_{\sum_{j=1}^{i-1} r_j}, z_1, \ldots, z_{\sum_{j=i+1}^{k} p_j})$.

Dann gilt:
$D_{g^{(i)}} = \{(\epsilon_1^{(1)}, \ldots, \epsilon_{r_1}^{(1)}, \ldots, \epsilon_1^{(i)}, \ldots, \epsilon_{r_i}^{(i)}, \epsilon_1, \ldots, \epsilon_{\sum_{j=i+1}^{k} p_j}) \mid$
$(\epsilon_1^{(j)}, \ldots, \epsilon_{r_j}^{(j)}) \in Im(\alpha^{(j)})\ (\forall 1 \leq j \leq i), (\epsilon_1, \ldots, \epsilon_{\sum_{j=i+1}^{k} p_j}) \in \{0,1\}^{\sum_{j=i+1}^{k} p_j}\}$.
Für $k \geq j > i \geq 1$*:*

$$g^{(i)}_{x_{j_1}^{\epsilon_1} \ldots x_{j_{p_j}}^{\epsilon_{p_j}}} = g^{(i)}_{x_{j_1}^{\delta_1} \ldots x_{j_{p_j}}^{\delta_{p_j}}} \iff g^{(i-1)}_{x_{j_1}^{\epsilon_1} \ldots x_{j_{p_j}}^{\epsilon_{p_j}}} = g^{(i-1)}_{x_{j_1}^{\delta_1} \ldots x_{j_{p_j}}^{\delta_{p_j}}} \iff f_{x_{j_1}^{\epsilon_1} \ldots x_{j_{p_j}}^{\epsilon_{p_j}}} = f_{x_{j_1}^{\delta_1} \ldots x_{j_{p_j}}^{\delta_{p_j}}}$$

und für $k \geq j > i \geq 0$*:*

$$vkk(X^{(j)}, g^{(i)}) = vz(X^{(j)}, f).$$

Beweis: Beweis durch Induktion über i:

$i = 0$:

$$D_{g^{(0)}} = D_f = \{0,1\}^n.$$

$$vkk(X^{(j)}, g^{(0)}) = vkk(X^{(j)}, f) = vz(X^{(j)}, f)\ \forall k \geq j > 0,$$

da f eine totale Funktion ist.

$i - 1 \rightarrow i$:

- Nach Induktionsvoraussetzung gilt:
  $D_{g^{(i-1)}} = \{(\epsilon_1^{(1)}, \ldots, \epsilon_{r_1}^{(1)}, \ldots, \epsilon_1^{(i-1)}, \ldots, \epsilon_{r_{i-1}}^{(i-1)}, \epsilon_1, \ldots, \epsilon_{\sum_{j=i}^{k} p_j}) \mid$
  $(\epsilon_1^{(j)}, \ldots, \epsilon_{r_j}^{(j)}) \in Im(\alpha^{(j)})\ (\forall 1 \leq j \leq i-1), (\epsilon_1, \ldots, \epsilon_{\sum_{j=i}^{k} p_j}) \in \{0,1\}^{\sum_{j=i}^{k} p_j}\}$

Nach Induktionsvoraussetzung ist also für alle Kofaktoren $g^{(i-1)}_{x^{\epsilon_1}_{i_1}\ldots x^{\epsilon_{p_i}}_{i_{p_i}}}$ die Definitionsmenge die gleiche. Infolgedessen gilt für $1 \le j \le v_i$

$$|\{g^{(i-1)}_{x^{\epsilon_1}_{i_1}\ldots x^{\epsilon_{p_i}}_{i_{p_i}}} | (\epsilon_1, \ldots, \epsilon_{p_i}) \in PK^{(i)}_j\}| = 1,$$

und

$$\begin{aligned} cof_{PK^{(i)}_j} &= gem\_erw(\{g^{(i-1)}_{x^{\epsilon_1}_{i_1}\ldots x^{\epsilon_{p_i}}_{i_{p_i}}} | (\epsilon_1, \ldots, \epsilon_{p_i}) \in PK^{(i)}_j\}) \\ &= g^{(i-1)}_{x^{\epsilon_1}_{i_1}\ldots x^{\epsilon_{p_i}}_{i_{p_i}}} \text{ für beliebiges } (\epsilon_1, \ldots, \epsilon_{p_i}) \in PK^{(i)}_j. \end{aligned}$$

Daher ist für alle $1 \le j \le v_i$
$D(cof_{PK^{(i)}_j}) = \{(\epsilon^{(1)}_1, \ldots, \epsilon^{(1)}_{r_1}, \ldots, \epsilon^{(i-1)}_1, \ldots, \epsilon^{(i-1)}_{r_{i-1}}, \epsilon_1, \ldots, \epsilon_{\sum_{j=i+1}^k p_j}) \mid$
$(\epsilon^{(j)}_1, \ldots, \epsilon^{(j)}_{r_j}) \in Im(\alpha^{(j)})\ (\forall 1 \le j \le i-1), (\epsilon_1, \ldots, \epsilon_{\sum_{j=i+1}^k p_j}) \in \{0,1\}^{\sum_{j=i+1}^k p_j}\}$.

Mit der Definition von $D_{g^{(i)}}$ (siehe Satz 3.9) ist die Behauptung bzgl. $D_{g^{(i)}}$ bewiesen.

- Z.z.: $g^{(i)}_{x^{\epsilon_1}_{j_1}\ldots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i)}_{x^{\delta_1}_{j_1}\ldots x^{\delta_{p_j}}_{j_{p_j}}} \iff g^{(i-1)}_{x^{\epsilon_1}_{j_1}\ldots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i-1)}_{x^{\delta_1}_{j_1}\ldots x^{\delta_{p_j}}_{j_{p_j}}}$ für $j > i$.

$\Longrightarrow$: Für alle $\epsilon^{(l)} \in Im(\alpha^{(l)})$ $(1 \le l \le i-1)$, $\epsilon^{(l)} \in \{0,1\}^{p_l}$ $(i \le l \le k, l \ne j)$ gilt:

$$\begin{aligned} &g^{(i-1)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \epsilon^{(i)}, \ldots, \epsilon^{(j-1)}, \epsilon, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) = \\ &\quad= g^{(i)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \alpha^{(i)}(\epsilon^{(i)}), \ldots, \epsilon^{(j-1)}, \epsilon, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) \\ &\quad\overset{Vor\underline{a}uss.}{=} g^{(i)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \alpha^{(i)}(\epsilon^{(i)}), \ldots, \epsilon^{(j-1)}, \delta, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) \\ &\quad= g^{(i-1)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \epsilon^{(i)}, \ldots, \epsilon^{(j-1)}, \delta, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}). \end{aligned}$$

Aus dem vorhergehenden Punkt des Beweises folgt, daß

$$D(g^{(i-1)}_{x^{\epsilon_1}_{j_1}\ldots x^{\epsilon_{p_j}}_{j_{p_j}}}) = D(g^{(i-1)}_{x^{\delta_1}_{j_1}\ldots x^{\delta_{p_j}}_{j_{p_j}}})$$

und damit

$$g^{(i-1)}_{x^{\epsilon_1}_{j_1}\ldots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i-1)}_{x^{\delta_1}_{j_1}\ldots x^{\delta_{p_j}}_{j_{p_j}}}.$$

$\Longleftarrow$: Für alle $\epsilon^{(l)} \in Im(\alpha^{(l)})$ $(1 \le l \le i)$, $\epsilon^{(l)} \in \{0,1\}^{p_l}$ $(i+1 \le l \le k, l \ne j)$ gilt:

$$\begin{aligned} &g^{(i)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \epsilon^{(i)}, \ldots, \epsilon^{(j-1)}, \epsilon, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) = \\ &\quad= g^{(i-1)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, (\alpha^{(i)})^{-1}(\epsilon^{(i)}), \ldots, \epsilon^{(j-1)}, \epsilon, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) \\ &\quad\overset{Vor\underline{a}uss.}{=} g^{(i-1)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, (\alpha^{(i)})^{-1}(\epsilon^{(i)}), \ldots, \epsilon^{(j-1)}, \delta, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}) \\ &\quad= g^{(i)}(\epsilon^{(1)}, \ldots, \epsilon^{(i-1)}, \epsilon^{(i)}, \ldots, \epsilon^{(j-1)}, \delta, \epsilon^{(j+1)}, \ldots, \epsilon^{(k)}). \end{aligned}$$

Aus Teil 1 des Beweises folgt

$$D(g^{(i)}_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}}) = D(g^{(i)}_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}})$$

und damit

$$g^{(i)}_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i)}_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}}.$$

Nach Induktionsvoraussetzung gilt demzufolge:

$$g^{(i)}_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i)}_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}} \iff f_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}} = f_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}}.$$

Für beliebige $\epsilon, \delta \in \{0,1\}^{p_j}$ ist $D(g^{(i)}_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}}) = D(g^{(i)}_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}})$ (s.o.).

Folglich gilt $\epsilon \sim_i \delta$ genau dann, wenn $g^{(i)}_{x^{\epsilon_1}_{j_1} \dots x^{\epsilon_{p_j}}_{j_{p_j}}} = g^{(i)}_{x^{\delta_1}_{j_1} \dots x^{\delta_{p_j}}_{j_{p_j}}}$ und somit ist

$$vkk(X^{(j)}, g^{(i)}) = vz(X^{(j)}, f) \; \forall k \geq j > i \geq 0$$

bewiesen.

□

Aufgrund des gerade bewiesenen Satzes kann man k–seitige Zerlegungen totaler Funktionen durch eine Folge von k einseitigen Zerlegungen durchführen. Das Ergebnis ändert sich nicht durch Änderung der Reihenfolge der einseitigen Zerlegungen.

Auch für die k–seitige Zerlegung *partieller* Funktionen kann man das angegebene Resultat nutzen: Setzt man voraus, daß die vorhandenen don't cares schon vor der Zerlegung (entweder zur Herstellung starker Symmetrien oder zur Kommunikationsminimierung) belegt worden sind, so kann man für die erhaltene totale Erweiterung eine k–seitige Zerlegung durch eine beliebige Reihenfolge von k einseitigen Zerlegungen durchführen (ohne daß die Reihenfolge einen Einfluß auf das Ergebnis hat). Belegt man allerdings die don't cares noch nicht alle vor Durchführung der k–seitigen Zerlegung, so hat das Verfahren den Charakter einer Heuristik: Durch unterschiedliche Belegung der verbleibenden don't cares bei unterschiedlicher Reihenfolge der k einseitigen Zerlegungen kann man zu unterschiedlichen Resultaten gelangen (vergleiche Beispiel 3.7, Seite 115).

3.4 Zerlegungen von Funktionen mit mehreren Ausgängen

Die Effizienz guter Realisierungen Boolescher Funktionen beruht häufig gerade auf der Tatsache, daß gleiche Teilschaltungen „mehrfach verwendet“ werden. Die Identifizierung von

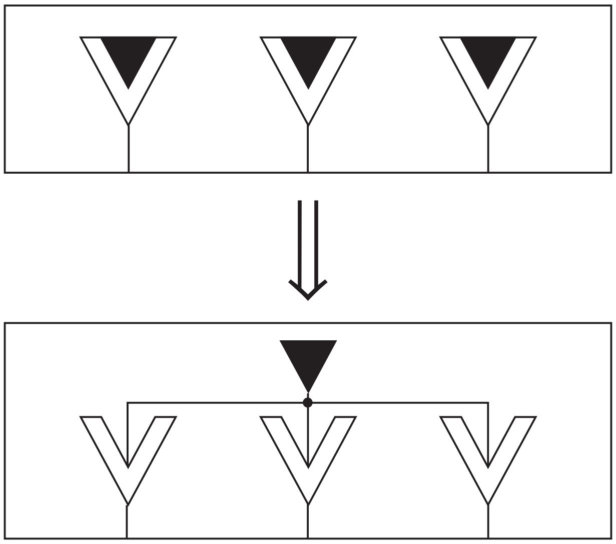


Abbildung 3.13: Identifizierung mehrfachverwendbarer Schaltungsteile.

Teillogik, die in mehreren Schaltungsteilen mit Vorteil verwendet werden kann, ist eine fundamentale Aufgabe der mehrstufigen Logiksynthese (siehe Abbildung 3.13). Bei Funktionen mit mehreren Ausgängen werden die einzelnen Ausgangsfunktionen nicht getrennt realisiert, sondern Ergebnisse, die von Teilschaltungen der Realisierung *einer* Ausgangsfunktion berechnet werden, werden bei der Realisierung *anderer* Ausgangsfunktionen mitverwendet.

So ist es bei dem Zerlegungsverfahren ebenfalls nicht ratsam, bei Funktionen $f \in B_{n,m}$ $(m > 1)$ mit mehreren Ausgängen die einzelnen Ausgänge getrennt als Funktionen $f_1, \ldots, f_m$ in einem Ausgang zu realisieren. Um bei Zerlegungen Boolescher Funktionen *gleiche* Teilschaltungen bei der Realisierung *mehrerer* Ausgangsfunktionen zu benutzen, werden in der vorliegenden Arbeit Zerlegungsfunktionen berechnet, die gleichzeitig bei der Zerlegung mehrerer Ausgangsfunktionen verwendet werden können (vgl. Abbildung 3.14).

Die Identifizierung mehrfachverwendbarer Teillogik ist bei der mehrstufigen Logiksynthese von enormer Wichtigkeit. Der vorliegende Abschnitt behandelt Lösungen für dieses Problem im Zusammenhang mit Zerlegungen Boolescher Funktionen. Aus diesem Grund sind die in diesem Abschnitt vorgestellten Resultate auch als eine der wesentlichen Leistungen dieser Arbeit zu betrachten.

Für die praktische Anwendbarkeit der Ergebnisse ist es von großer Bedeutung, daß die Verfahren nicht wie in einer früheren Version aus [MS94] nur für Zerlegungsmatrizen bzw. Funktionstabellen formuliert werden konnten, sondern auf ROBDDs als Datenstruktur arbeiten (vgl. [SM94] und [SM95b]).

Der Abschnitt beginnt mit einer kurzen Vorstellung der hier gewählten Vorgehensweise

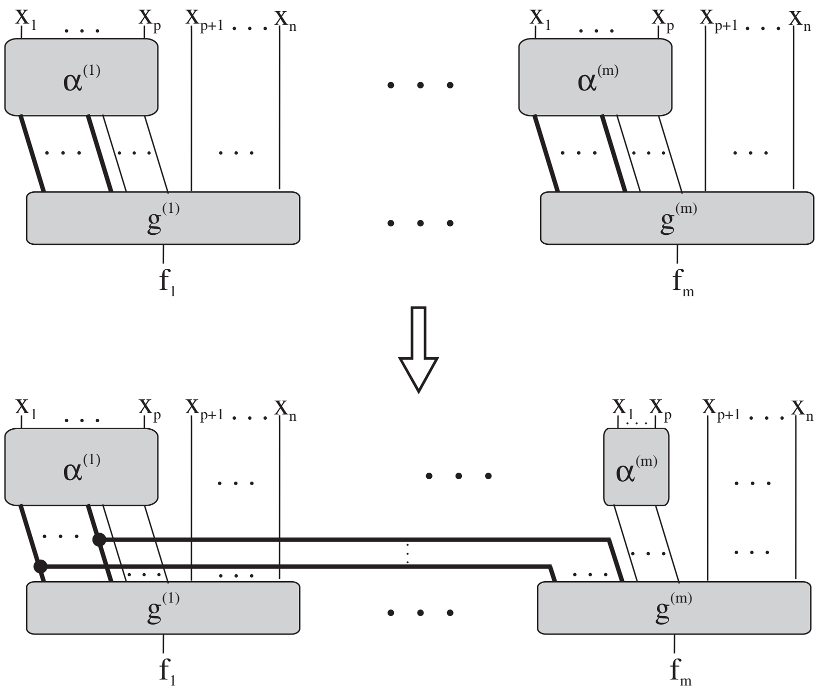


Abbildung 3.14: Zerlegung von Funktionen mit mehreren Ausgängen.

anhand eines kleinen Beispiels. Dabei wird verdeutlicht, wie bei der Zerlegung von Funktionen mit mehreren Ausgängen Freiheiten bei der Wahl der Zerlegungsfunktionen ausgenutzt werden können, um mehrfachverwendbare Teillogik (in Form von gemeinsamen Zerlegungsfunktionen mehrerer Ausgangsfunktionen) zu erzeugen.

Danach wird das Verfahren zur Bestimmung geeigneter Variablenaufteilungen bzw. Variablenteilmengen zur Zerlegung aus Abschnitt 3.1.4 verallgemeinert auf Funktionen mit mehreren Ausgängen.

Der nächste Teilabschnitt stellt den Kern des vorliegenden Abschnitts dar. Nach einer Untersuchung der Komplexität des Problems, gemeinsame Zerlegungsfunktionen mehrerer Ausgangsfunktionen zu bestimmen, wird schließlich ein Algorithmus zur Berechnung gemeinsamer Zerlegungsfunktionen hergeleitet. Hierbei stellt sich heraus, daß die Beschränkung auf gemeinsame, *strikte* Zerlegungsfunktionen nicht nur der Erhaltung von Struktureigenschaften der ursprünglichen Funktionen dient (vergleiche Abschnitt 3.1.3.3), sondern auch erheblich zur Beschleunigung der Berechnung beitragen kann.

Der Abschnitt wird abgeschlossen mit der Entwicklung eines Verfahrens zur Belegung von don't cares in Hinblick auf die Berechnung gemeinsamer Zerlegungsfunktionen. Zu diesem Zweck wird das Verfahren zur Minimierung der Anzahl der benötigten Zerlegungsfunktionen aus den Abschnitten 3.2.2 und 3.2.3 zur Anwendung auf Funktionen mit mehreren Ausgängen modifiziert. Schließlich konnte für das resultierende Verfahren gezeigt werden, daß es mit dem ursprüglichen Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen verträglich ist: Seine Ergebnisse werden nicht zerstört, falls danach noch das Verfahren aus den Abschnitten 3.2.2 bzw. 3.2.3 angewendet wird.

Die grundsätzliche Vorgehensweise in diesem Abschnitt wird durch Beispiel 3.11 illustriert:

Beispiel 3.11 Gegeben sei eine Boolesche Funktion f aus $B_{4,3}$ mit $f_1(x_1, \ldots, x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$, $f_2(x_1, \ldots, x_4) = \overline{x_4}(x_1 \oplus x_2) + x_4(x_2 + x_3)$ und $f_3(x_1, \ldots, x_4) = \overline{x_4}(x_2 + x_3) + x_1x_4$. f_1, f_2 und f_3 werden jeweils hinsichtlich der Variablenteilmenge $\{x_1, x_2, x_3\}$ zerlegt (vgl. Abbildung 3.15). Es treten jeweils 4 „Verbindungsknoten" auf, so daß man je 2 Zerlegungsfunktionen benötigt. Bei der Kodierung der Verbindungsknoten bzw. der zugehörigen Klassen aus $\{0, 1\}^3$ hat man allerdings noch Freiheiten: In Abbildung 3.15 ist eine Kodierung angegeben, die zu der ebenfalls in Abbildung 3.15 gezeigten Realisierung führt. Man erkennt, daß bei dieser Kodierung alle Zerlegungsfunktionen $\alpha_1^{(1)}$, $\alpha_2^{(1)}$, $\alpha_1^{(2)}$, $\alpha_2^{(2)}$, $\alpha_1^{(3)}$ und $\alpha_2^{(3)}$ paarweise verschieden sind.
Ziel ist es nun, die Kodierung so zu wählen, daß möglichst viele Zerlegungsfunktionen von den Ausgangsfunktionen f_1, f_2 und f_3 *gemeinsam* verwendet werden können. Abbildung 3.16 zeigt eine solche Kodierung. Bei dieser Kodierung sind die Zerlegungsfunktionen $\alpha_2^{(1)}$ und $\alpha_1^{(2)}$ bzw. $\alpha_2^{(2)}$ und $\alpha_1^{(3)}$ paarweise identisch und müssen daher nur einmal realisiert werden.

In dem Verfahren zur *Berechnung* gemeinsamer Zerlegungsfunktionen, das in Abschnitt 3.4.2.1 vorgestellt wird, werden im Gegensatz zu anderen Lösungen wie z.B. in [HOI89] nicht Teilfunktionen miteinander verglichen in der Hoffnung, daß einige dieser Teilfunktionen zufällig gleich sind, sondern es wird darauf hingearbeitet, gleiche Teilfunktionen zu erhalten. Kodierungen wie die im obigen Beispiel werden *berechnet* mit dem Ziel, möglichst viele Zerlegungsfunktionen in der Zerlegung mehrerer Ausgangsfunktionen *gemeinsam* verwenden zu können.

Das Vorgehen ist nicht nur bei der Bearbeitung von Booleschen Funktionen mit *mehreren* Ausgängen von Bedeutung. Auch wenn eine Funktion ursprünglich nur einen Ausgang hat, so treten bei der rekursiven Anwendung des Zerlegungsverfahrens auf ihre Zerlegungsfunktionen im allgemeinen Funktionen mit mehreren Ausgängen auf.

Anstatt die einzelnen Ausgangsfunktionen kommunikationsminimal zu zerlegen und dabei darauf zu achten, daß möglichst viele Zerlegungsfunktionen gemeinsam verwendet werden können, könnte man auch daran denken, die Funktion mit mehreren Ausgängen als Ganzes kommunikationsminimal zu zerlegen (vergleiche auch [LPV94]), d.h. zu $f \in B_{n,m}$ eine Zerlegungsfunktion $\alpha \in B_{p,r}$ und eine Zusammensetzungsfunktion $g \in B_{n-p+r,m}$ zu suchen, so daß

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n). \quad (\star)$$

Die Anzahl der benötigten Zerlegungsfunktionen ergibt sich auch hier anhand der Zerlegungsmatrix von f. Die Einträge in der Matrix sind hierbei nicht Elemente aus $\{0, 1\}$,

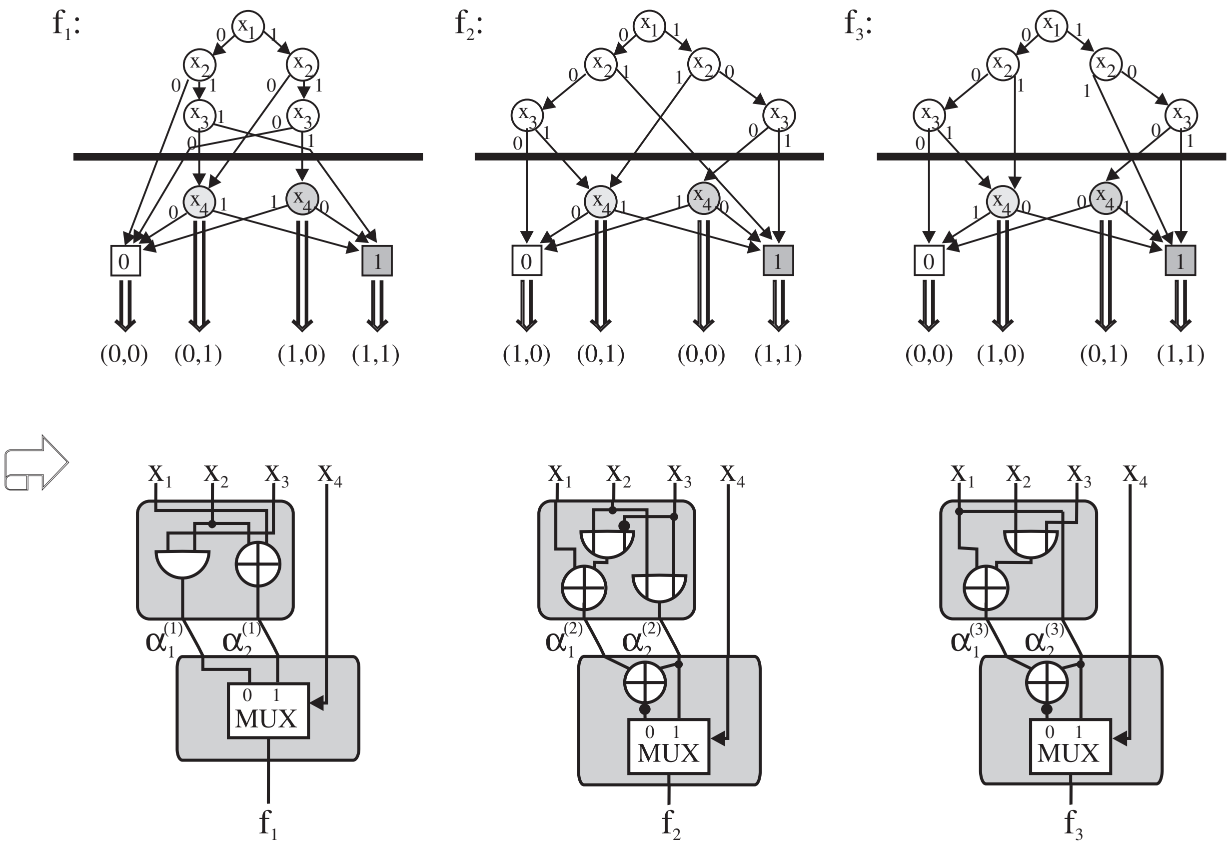


Abbildung 3.15: Zerlegung der Beispielsfunktion aus Beispiel 3.11. $f_1(x_1,\ldots,x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$, $f_2(x_1,\ldots,x_4) = \overline{x_4}(x_1 \oplus x_2) + x_4(x_2 + x_3)$ und $f_3(x_1,\ldots,x_4) = \overline{x_4}(x_2 + x_3) + x_1x_4$.

sondern aus $\{0,1\}^m$. Für die Funktion aus Beispiel 3.11 sieht dann die Zerlegungsmatrix hinsichtlich $\{x_1, x_2, x_3\}$ folgendermaßen aus:

$x_1x_2x_3$ \ x_4	0	1
0 0 0	(000)	(000)
0 0 1	(001)	(010)
0 1 0	(011)	(110)
0 1 1	(111)	(110)
1 0 0	(010)	(101)
1 0 1	(011)	(111)
1 1 0	(001)	(011)
1 1 1	(101)	(011)

Analog zum Beweis von Satz 3.1 sieht man, daß sich die minimale Anzahl von Zerlegungs-

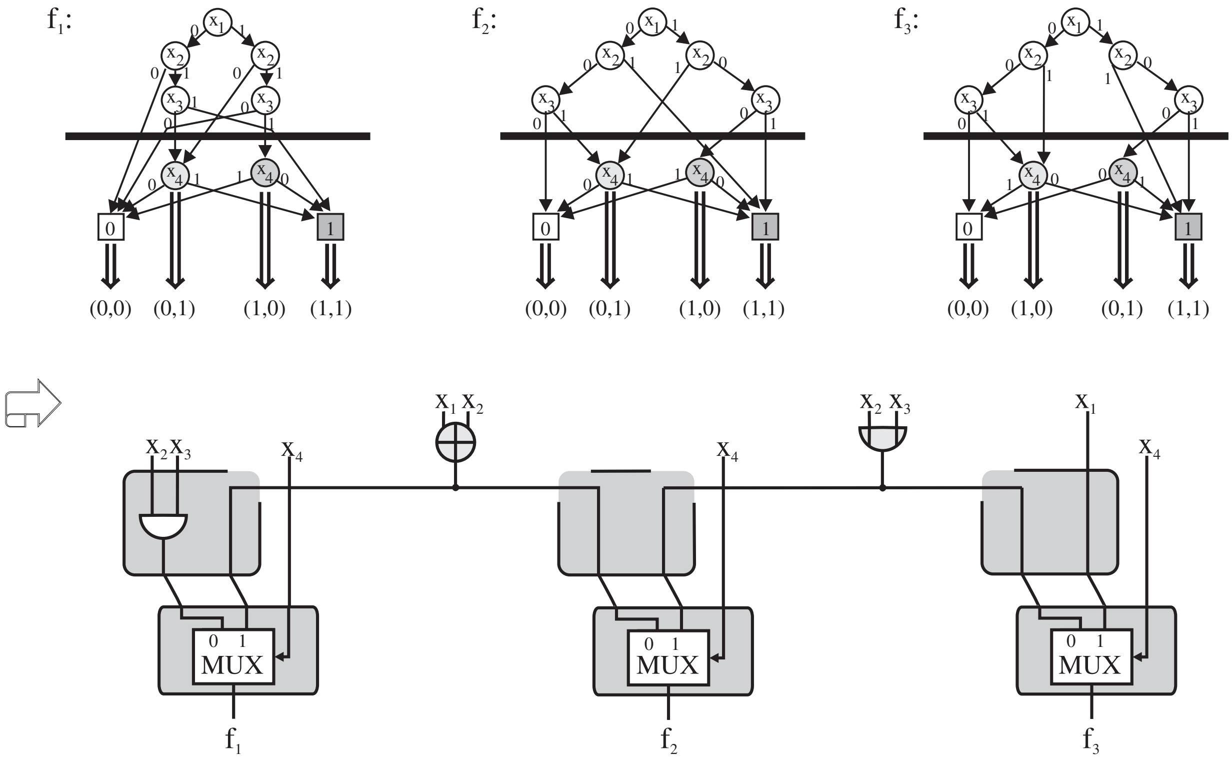


Abbildung 3.16: Zerlegung der Beispielsfunktion aus Beispiel 3.11. Es wurde eine andere Kodierung gewählt wie in Abbildung 3.15.

funktionen in einer Zerlegung von f der Form $(\star)$ ergibt als $\lceil \log(vz) \rceil$, wobei vz die Anzahl der verschiedenen Zeilenmuster in der Zerlegungsmatrix ist. Das Problem, das man bei Zerlegungen dieser Art hat, besteht allerdings darin, daß man (insbesondere bei größerem m) auch bei praktisch interessanten Funktionen nur selten nichttriviale Zerlegungen findet. Auch bei der Funktion aus Beispiel 3.11 hat man 8 verschiedene Zeilenmuster und damit 3 Zerlegungsfunktionen. Jeder Wert aus $\{0,1\}^3$ muß also durch einen eindeutigen Wert aus $\{0,1\}^3$ kodiert werden. Wählt man als Kodierung die Identität auf $\{0,1\}^3$, so erhält man die Realisierung in Abbildung 3.17. α ist die Identität, die Zusammensetzungsfunktionen g_i sind gleich f_i $(i = 1, 2, 3)$ und man hat durch die Zerlegung nichts gewonnen. (Würde man eine andere Kodierung wählen, so sind die Zusammensetzungsfunktionen g_i evtl. sogar noch komplexer als f_i.)

Aufgrund der Schwierigkeit, nichttriviale Zerlegungen zu finden, sind Zerlegungen vom Typ $(\star)$ nur in seltenen Fällen (insbesondere bei kleinem m) empfehlenswert und hier wird von solchen Zerlegungen für Funktionen mit mehreren Ausgängen abgesehen.

Das vorgeschlagene Verfahren, bei der Zerlegung der Ausgangsfunktionen $f_1, \ldots, f_m$ einer Funktion $f \in B_{n,m}$ nach *gemeinsamen* Zerlegungsfunktionen zu suchen, ist nur dann

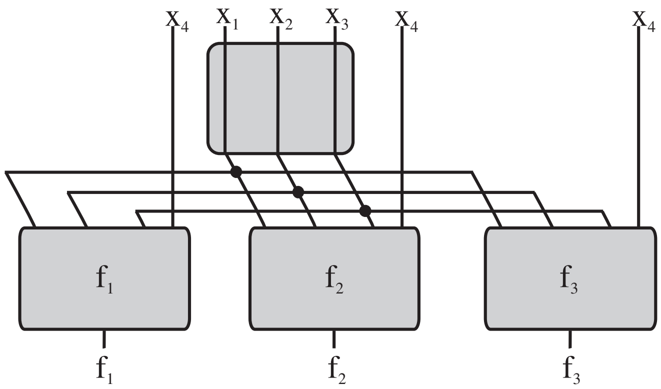


Abbildung 3.17: Zerlegung der Funktion aus Beispiel 3.11 (alle Ausgänge gemeinsam).

anwendbar, wenn die Zerlegung der Ausgangsfunktionen *hinsichtlich der gleichen Variablenaufteilung* erfolgt. Es ist jedoch nicht vorteilhaft, diese Forderung in jedem Fall aufrechtzuerhalten. Es kann vorkommen, daß die Anzahl der Zerlegungsfunktionen, die bei einer bestimmten Variablenaufteilung notwendig sind, für eine der Ausgangsfunktionen minimal ist, für andere Ausgangsfunktionen aber sehr weit vom Minimum entfernt liegt. Daher werden die Funktionen $f_1, \ldots, f_m$ durch eine Heuristik in *Gruppen* eingeteilt, wobei alle Funktionen aus einer dieser Gruppen hinsichtlich der gleichen Variablenaufteilung zerlegt werden. Die Gruppen werden gerade so gewählt, daß die Anzahl der Zerlegungsfunktionen, die bei Zerlegung hinsichtlich dieser Variablenaufteilung notwendig sind, für keine Funktion aus der Gruppe allzu stark vom Minimum abweicht. Im nächsten Abschnitt wird ein Verfahren angegeben, das die Ausgangsfunktionen in Gruppen einteilt und zu jeder Gruppe eine Variablenaufteilung wählt. Im darauffolgenden Abschnitt wird ein Verfahren angegeben, das die Ausgangsfunktionen aus einer solchen Gruppe hinsichtlich der gewählten Variablenaufteilung zerlegt und dabei möglichst viele mehrfach verwendbare Zerlegungsfunktionen berechnet.

3.4.1 Ausgangspartitionierung und Wahl einer geeigneten Variablenaufteilung

In diesem Abschnitt wird bei der Bestimmung einer geeigneten Variablenteilmenge, hinsichtlich der zerlegt werden soll, angenommen, daß die Mächtigkeit p dieser Variablenmenge noch nicht festgelegt worden ist. Ein analoger Algorithmus für festes p läßt sich leicht aus dem vorliegenden Algorithmus herleiten.

Zunächst wird das Verfahren aus Abschnitt 3.1.4 zur Bestimmung einer günstigen Variablenteilmenge für jede Ausgangsfunktion $f_1, \ldots, f_m$ getrennt durchgeführt. Im Verlauf des Verfahrens für Ausgangsfunktion f_i werden für jede Variablenordnung $<_{index}$, die durch das greedy–Verfahren erzeugt wird, und jedes $2 \leq p \leq n-1$ die geschätzten Kosten

$est\_cost_p(<_{index})$ bestimmt.
Da man zur Bestimmung gemeinsamer Zerlegungsfunktionen nach Möglichkeit verschiedene Ausgangsfunktionen hinsichtlich der gleichen Variablenteilmenge zerlegen will, wählt man zur Zerlegung einer Funktion f_i allerdings *nicht* die Variablenteilmenge (Variablenordnung mit optimalem p), die sich aus Anwendung des greedy–Verfahrens für f_i ergeben würde. Stattdessen werden sämtliche Variablenordnungen, die im Verlauf der Verfahren aufgetreten sind, (zusammen mit einem $p \in \{2, \ldots, n-1\}$) als potentielle Kandidaten für die Zerlegung von $f_1, \ldots, f_m$ betrachtet.
Um evtl. Variablenordnungen (bzw. Variablenteilmengen) zu finden, die für *alle* Funktionen $f_1, \ldots, f_m$ geeignet sind, wird noch ein greedy–Verfahren durchgeführt, bei dem die geschätzten Kosten für eine Variablenordnung $<_{index}$ und ein $p \in \{2, \ldots, n-1\}$ sich als Summe der geschätzten Kosten für die Zerlegung der einzelnen Ausgangsfunktionen ergeben. Auch diese Variablenordnungen werden zur Kandidatenmenge Φ hinzugefügt[3].

Nun wird ausgehend von der Kandidatenmenge Φ zu jedem f_i eine Variablenordnung $<_{index}$ aus Φ und ein $p \in \{2, \ldots, n-1\}$ gewählt, so daß f_i hinsichtlich $\{x_{index(1)}, \ldots, x_{index(p)}\}$ zerlegt wird. Hierbei wird versucht — soweit dies nicht zu allzu großen Verschlechterungen der geschätzten Kosten führt — möglichst viele Ausgangsfunktionen hinsichtlich der gleichen Variablenaufteilung zu zerlegen. Die Einteilung von $f_1, \ldots, f_m$ in Gruppen mit zugehöriger Variablenteilmenge wird durch folgenden Algorithmus durchgeführt:

Eingabe:
- Eine Menge $F = \{f_1, \ldots, f_m\}$ von Funktionen aus B_n $(n \geq 4)$.
- Eine Menge Φ von Variablenordnungen. Für jede Variablenordnung $<_{index}$ aus Φ, jedes $2 \leq p \leq n-1$ und jedes $1 \leq i \leq m$:
  Geschätzte Kosten $est\_cost_p^{(i)}(<_{index})$ für eine Zerlegung von f_i hinsichtlich $\{x_{index(1)}, \ldots, x_{index(p)}\}$[4].

Ausgabe: Eine Einteilung der Funktionen in F in Gruppen $G_1, \ldots, G_g$ mit $\cup_{1 \leq i \leq g} G_i = F$ und $G_i \cap G_j = \emptyset$ für $i \neq j$. Zu jeder Gruppe G_i gehört eine Variablenteilmenge $X_i^{(1)}$, hinsichtlich der die Funktionen aus G_i zerlegt werden sollen.

Algorithmus:

$g := 0$
$\forall f_i \in F : min\_cost^{(i)} := \min(\{est\_cost_p^{(i)}(<_{index}) \mid <_{index} \in \Phi, 2 \leq p \leq n-1\})$
while $F \neq \emptyset$ **do**
  $g := g + 1$
  Wähle $f_i \in F$ mit $min\_cost^{(i)} \geq min\_cost^{(j)}$ $\forall f_j \in F$
  Seien $<_{index_{opt}}$ und p_{opt} mit $min\_cost^{(i)} = est\_cost_{p_{opt}}^{(i)}(<_{index_{opt}})$
  $G_g := \{f_i\}$
  $F := F \setminus \{f_i\}$

[3] Ist n sehr klein, so wird man Φ als die Menge *aller* möglichen Variablenordnungen wählen.
[4] Hier werden einseitige Zerlegungen betrachtet. Analog läßt sich ein Algorithmus für zweiseitige Zerlegungen angeben.

9 $X_g^{(1)} := \{x_{index_{opt}(1)}, \dots, x_{index_{opt}(p_{opt})}\}$
10 *gruppe_beendet* := **false**
11 **while** ((*gruppe_beendet* = **false**) **and** $(F \neq \emptyset)$) **do**
12 /* Es gelte $G_g = \{f_{i_1}, \dots, f_{i_k}\}$ */
13 Bestimme $f_j \in F, <_{index_{neu}} \in \Phi$ und $2 \leq p_{neu} \leq n-1$, so daß
14 $\frac{sharing(k+1)}{k+1} \left(\left(\sum_{l=1}^{k} est\_cost_{p_{neu}}^{(i_l)}(<_{index_{neu}}) \right) + est\_cost_{p_{neu}}^{(j)}(<_{index_{neu}}) \right) -$
15 $\frac{sharing(k)}{k} \left(\sum_{l=1}^{k} est\_cost_{p_{opt}}^{(i_l)}(<_{index_{opt}}) \right) + min\_cost^{(j)}$ minimal ist.
16 **if** $\left(\frac{sharing(k+1)}{k+1} \left(\left(\sum_{l=1}^{k} est\_cost_{p_{neu}}^{(i_l)}(<_{index_{neu}}) \right) + est\_cost_{p_{neu}}^{(j)}(<_{index_{neu}}) \right) \leq \right.$
17 $\left. \frac{sharing(k)}{k} \left(\sum_{l=1}^{k} est\_cost_{p_{opt}}^{(i_l)}(<_{index_{opt}}) \right) + min\_cost^{(j)} \right)$
18 /* Wenn die geschätzten Kosten für eine Realisierung der Funktionen aus G_g
19 mit f_j zusammen nicht höher sind als bei einer getrennten Realisierung
20 der Funktionen aus G_g (mit der auf G_g optimierten Variablenaufteilung)
21 und f_j (mit der für f_j optimalen Variablenaufteilung) */
22 **then**
23 $<_{index_{opt}} := <_{index_{neu}}$
24 $p_{opt} := p_{neu}$
25 $G_g := G_g \cup \{f_j\}$
26 $F := F \setminus \{f_j\}$
27 $X_g^{(1)} := \{x_{index_{opt}(1)}, \dots, x_{index_{opt}(p_{opt})}\}$
28 **else**
29 *gruppe_beendet* := **true**
30 **fi**
31 **od**
32 **od**

Der Algorithmus geht von einer aktuelle Gruppe G_g von Funktionen aus, die zusammen hinsichtlich der Variablenteilmenge $\{x_{index_{opt}(1)}, \dots, x_{index_{opt}(p_{opt})}\}$ zerlegt würden und versucht im Durchlauf der Schleife in den Zeilen 11–31, eine weitere Funktion f_j zu G_g hinzuzufügen. Die Funktion f_j, die neue Variablenordnung $<_{index_{neu}}$ und p_{neu} werden dabei so gewählt, daß der geschätzte Gewinn maximiert wird, der durch Logiksharing bei Zerlegung (hinsichtlich der Variablenteilmenge $\{x_{index_{neu}(1)}, \dots, x_{index_{neu}(p_{neu})}\}$) der Funktionen aus $G_g \cup f_j$ entsteht (im Vergleich zu einer getrennten Zerlegung der Funktionen aus G_g hinsichtlich $\{x_{index_{opt}(1)}, \dots, x_{index_{opt}(p_{opt})}\}$ und der Funktion f_j hinsichtlich der für f_j besten Variablenteilmenge) (Zeilen 13–15 des Algorithmus). (Bei der Abschätzung ist *sharing* wie in Abschnitt 3.1.4 eine streng monoton wachsende Funktion mit $sharing(k) \leq k$, z.B. $sharing(k) = (k)^c$ mit $0 < c \leq 1$.) Gibt es eine Funktion f_j, so daß sich bei Aufnahme von f_j in G_g eine Verringerung der geschätzten Kosten ergeben würde (Zeilen 16–17), so wird f_j zur Gruppe G_g hinzugefügt, ansonsten wird mit dem Aufbau einer neuen Gruppe begonnen.

Am Ende des Algorithmus wurden alle Funktionen einer Gruppe G_i zugeordnet, wobei alle Funktionen aus G_i hinsichtlich der gleichen Variablenteilmenge $X_i^{(1)}$ zerlegt werden.

Es ist zu beachten, daß durch den obigen Algorithmus ausschließlich nichttriviale Zerlegungen ausgewählt werden. Der Grund liegt darin, daß die Kostenabschätzung für Zerlegungen, die nicht nichttrivial sind, den Wert $+\infty$ liefert (vgl. Abschnitt 3.1.4). Da (zumindest für

$p = n - 1$) aber zu jeder Ausgangsfunktion nichttriviale Zerlegungen existieren, werden auch nur nichttriviale Zerlegungen ausgewählt.

Die Arbeitsweise der angegebenen Heuristik soll anhand zweier kleinerer Beispiele demonstriert werden:

Beispiel 3.12 Es sei eine Funktion $f = (f_0, f_1) \in B_{4,2}$ gegeben. Es gelte $\forall(x_0, x_1, x_2, x_3) \in \{0,1\}^4$:

$$\begin{aligned} f_0(x_0, x_1, x_2, x_3) &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ f_1(x_0, x_1, x_2, x_3) &= (x_0 \oplus x_2) \cdot (x_1 \oplus x_3) \end{aligned}$$

Es soll eine zweiseitige Zerlegung von f durchgeführt werden. Die Gruppierung der Ausgangsfunktionen und die Bestimmung einer Variablenaufteilung für die zweiseitige Zerlegung erfolgt genau analog zu der obigen Heuristik für einseitige Zerlegungen.

Angenommen die Kosten für die Realisierung einer Funktion mit 2 Eingängen werde mit $cost(2) = 1$ abgeschätzt (ein 2–Input–Gatter), die Kosten für die Realisierung einer Funktion mit 3 Eingängen werde mit $cost(3) = 3$ abgeschätzt.

Für die Anzahl von Zerlegungsfunktionen und die geschätzten Kosten gilt:

- Für f_0: Für alle nicht gleichmächtigen Variablenaufteilungen ist die Anzahl der Zerlegungsfunktionen 2 und die geschätzten Kosten sind 4 (f_0 ist totalsymmetrisch). Für alle gleichmächtigen Zerlegungen ist die Anzahl der Zerlegungsfunktionen ebenfalls 2, die geschätzten Kosten sind 3.
- Für f_1: Für die Variablenaufteilung $\{\{x_0, x_2\}, \{x_1, x_3\}\}$ ist die Anzahl der Zerlegungsfunktionen gleich 2 und die geschätzten Kosten sind 3. Für alle anderen Variablenaufteilungen sind die geschätzten Kosten größer.

Der Algorithmus beginnt die erste Gruppe beispielsweise mit f_0. Auch f_1 wird dieser Gruppe zugeordnet, da bei Wahl von $\{\{x_0, x_2\}, \{x_1, x_3\}\}$ als Variablenaufteilung selbst bei der Wahl $sharing(n) = n$ die geschätzen Kosten bei gemeinsamer Realisierung von f_0 und f_1 in Zeile 16 des Algorithmus nicht größer sind als die geschätzten Kosten bei getrennter Realisierung. Die Bedingung in Zeile 16 des Algorithmus ist also auf jeden Fall erfüllt.

Als Zerlegungsfunktionen auf den Variablen x_0 und x_2 kommen dann sowohl bei f_0 als auch bei f_1 lediglich

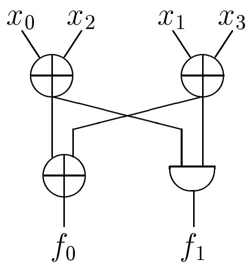
$$\alpha_1(x_0, x_2) = x_0 \oplus x_2 \text{ bzw. } \alpha_1'(x_0, x_2) = \overline{x_0 \oplus x_2}$$

in Frage. Ebenso kommen als Zerlegungsfunktionen auf den Variablen x_1 und x_3 lediglich

$$\beta_1(x_1, x_3) = x_1 \oplus x_3 \text{ bzw. } \beta_1'(x_1, x_3) = \overline{x_1 \oplus x_3}$$

in Frage.

Wählt man die Zerlegungsfunktionen so aus, daß für f_0 und f_1 Zerlegungsfunktionen gemeinsam verwendet werden können (z.B. bei Wahl von α_1 und β_1 sowohl für f_0 als auch für f_1), so ergibt sich folgende Realisierung unter Ausnutzung der Zerlegung[5]:



Beispiel 3.13 Als weiteres kleines Beispiel wird ein 2–Bit–Multiplizierer betrachtet. Es handelt sich um eine Boolesche Funktion $mult_2 \in B_{4,4}$,

$$mult_2 : \{0,1\}^4 \rightarrow \{0,1\}^4, mult_2(a_1, a_0, b_1, b_0) = (r_3, r_2, r_1, r_0),$$

wobei

$$(2a_1 + a_0) \cdot (2b_1 + b_0) = \sum_{i=0}^{3} (r_i 2^i).$$

Die einzelnen Ausgangsfunktionen von $mult_2$ werden mit $p_3, \ldots, p_0$ bezeichnet, also $mult_2 = (p_3, p_2, p_1, p_0)$.

Für die Kostenabschätzung wird wieder wie im obigen Beispiel $cost(2) = 1$ und $cost(3) = 3$ angenommen. Die Funktion *sharing* zur Abschätzung der Kostenverringerung durch Logiksharing wird als $sharing(n) = n^{0.9}$ angenommen.

Für die Anzahl von Zerlegungsfunktionen und die geschätzten Kosten gilt:

- Für p_3: Für alle nicht gleichmächtigen Variablenaufteilungen ist die Anzahl der Zerlegungsfunktionen 2 (p_3 ist totalsymmetrisch) und die geschätzten Kosten sind 4. Für alle gleichmächtigen Zerlegungen ist die Anzahl der Zerlegungsfunktionen ebenfalls 2, die geschätzten Kosten sind 3.

- Für p_2: Für die nicht gleichmächtigen Variablenaufteilungen $\{\{a_1\}, \{a_0, b_0, b_1\}\}$ und $\{\{b_1\}, \{a_0, a_1, b_0\}\}$ ergibt sich jeweils 1 Zerlegungsfunktion, die von $\{a_0, b_0, b_1\}$ bzw. $\{a_0, a_1, b_0\}$ abhängt, so daß die Kosten mit 4 abgeschätzt werden. Für die beiden verbleibenden nicht gleichmächtigen Variablenaufteilungen ergeben sich 2 Zerlegungsfunktionen, die von der Variablenteilmenge mit Mächtigkeit 3 abhängen, und damit

[5]Näheres zur Wahl von gemeinsamen Zerlegungsfunktionen für den allgemeinen Fall findet man im nächsten Abschnitt.

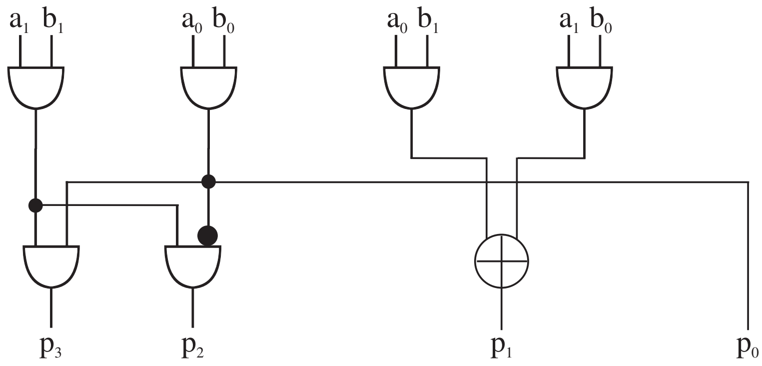
Kosten 9. Sowohl für die Variablenaufteilung $\{\{a_0, a_1\}, \{b_0, b_1\}\}$ als auch für die Variablenaufteilung $\{\{a_0, b_1\}, \{a_1, b_0\}\}$ ist die Zerlegung nicht nichttrivial, so daß die Kosten mit $+\infty$ abgeschätzt werden.
Für die Variablenaufteilung $\{\{a_0, b_0\}, \{a_1, b_1\}\}$ ergeben sich 2 Zerlegungsfunktionen, so daß die Kosten mit 3 abgeschätzt werden.

- Für p_1: Für alle nicht gleichmächtigen Variablenaufteilungen sind die geschätzten Kosten gleich 9 (jeweils 2 Zerlegungsfunktionen, die von der Variablenteilmenge mit 3 Variablen abhängen). Sowohl für die Variablenaufteilung $\{\{a_0, a_1\}, \{b_0, b_1\}\}$ als auch für die Variablenaufteilung $\{\{a_0, b_0\}, \{a_1, b_1\}\}$ ist die Zerlegung nicht nichttrivial, so daß die Kosten mit $+\infty$ abgeschätzt werden.
  Für die Variablenaufteilung $\{\{a_0, b_1\}, \{a_1, b_0\}\}$ ergeben sich 2 Zerlegungsfunktionen, so daß die Kosten mit 3 abgeschätzt werden.

- Für p_0: Für alle Variablenaufteilungen betragen die geschätzten Kosten 1.

Die obige Heuristik beginnt die erste Gruppe beispielsweise mit p_3. Danach wird p_2 ausgewählt mit der Variablenaufteilung $\{\{a_0, b_0\}, \{a_1, b_1\}\}$ für p_3 und p_2. Als nächstes wird p_0 zu dieser Gruppe hinzugefügt, die Variablenaufteilung bleibt $\{\{a_0, b_0\}, \{a_1, b_1\}\}$ für p_3, p_2 und p_0. Die gewählte Variablenaufteilung ist an dieser Stelle immer noch für alle Funktionen der Gruppe optimal. p_1 kann im nächsten Schritt *nicht* zu dieser Gruppe hinzugefügt werden: Es gibt keine gleichmächtige Variablenaufteilung, so daß die Zerlegung *sowohl* für p_2 *als auch* für p_1 nichttrivial ist. Unter den nicht gleichmächtigen Variablenaufteilungen wären $\{\{a_1\}, \{a_0, b_0, b_1\}\}$ bzw. $\{\{b_1\}, \{a_0, a_1, b_0\}\}$ die Kandidaten, bei denen die Summe der geschätzten Kosten am geringsten ist (bei p_3 Kosten 4, bei p_2 Kosten 4, bei p_1 Kosten 9 und bei p_0 Kosten 1). Allerdings ist die Verschlechterung durch das Erzwingen der gemeinsamen Variablenaufteilung in beiden Fällen so groß, daß p_1 nicht in die Gruppe aufgenommen wird (Zeile 16 des Algorithmus): Die geschätzen Kosten bei gemeinsamer Realisierung aller Ausgangsfunktionen sind dann $\frac{sharing(4)}{4} \cdot (4 + 4 + 9 + 1) > 15.66$, während die geschätzten Kosten für die Realisierung von p_3, p_2 und p_0 zusammen $\frac{sharing(3)}{3} \cdot (3 + 3 + 1) < 6.28$ sind und die geschätzten Kosten für die Realisierung von p_1 (bei optimaler Variablenaufteilung) 3 sind, so daß die Kosten bei getrennter Realisierung von p_1 insgesamt kleiner als 9.28 geschätzt werden . Die Auswertung in Zeile 16 des Algorithmus liefert also ein negatives Resultat, so daß p_1 einer eigenen Gruppe mit Variablenaufteilung $\{\{a_0, b_1\}, \{a_1, b_0\}\}$ zugeordnet wird.

Als Realisierung für $mult_2$ liefert der implementierte Logiksynthesealgorithmus schließlich die folgende Schaltung:



Es ist zu beachten, daß die Zerlegungsfunktion $a_1 \cdot b_1$ in der Zerlegung von p_3 und p_2 gemeinsam verwendet wird, und die Zerlegungsfunktion $a_0 \cdot b_0$ in der Zerlegung von p_3, p_2 und p_0 gemeinsam verwendet wird.

3.4.2 Berechnung gemeinsamer Zerlegungsfunktionen

In diesem Abschnitt soll untersucht werden, wie man Zerlegungsfunktionen berechnen kann, die bei der Zerlegung mehrerer Ausgangsfunktionen gemeinsam verwendet werden können. Dabei wird vorausgesetzt, daß alle betrachteten Ausgangsfunktionen hinsichtlich der gleichen Variablenteilmenge bzw. Variablenaufteilung zerlegt werden.

Zu diesem Zweck wird zunächst ein Lemma bewiesen, das ein Kriterium dafür liefert, ob eine bestimmte Menge von Funktionen $\alpha'_1, \ldots, \alpha'_h$ als Zerlegungsfunktionen in einer kommunikationsminimalen Zerlegung einer Funktion $f \in B_n$ verwendet werden können:

Lemma 3.9 *Sei $f \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_n$, sei $X^{(1)} = \{x_1, \ldots, x_p\}$ und sei $r = \lceil \log(vz(X^{(1)}, f)) \rceil$. Dann existiert genau dann eine einseitige Zerlegung hinsichtlich $X^{(1)}$ der Form*

$$f(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) = g(\alpha'_1(\mathbf{x^{(1)}}), \ldots, \alpha'_h(\mathbf{x^{(1)}}), \alpha_{h+1}(\mathbf{x^{(1)}}), \ldots, \alpha_r(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}),$$

wenn gilt

$$vz(X^{(1)}, f, \alpha') \leq 2^{r-h}.$$

Hierbei ist die Bezeichnung $vz(X^{(1)}, f, \alpha')$ folgendermaßen definiert:

Definition 3.10 *Sei $Z(X^{(1)})$ die Zerlegungsmatrix von f hinsichtlich $X^{(1)}$ und sei A das Bild von $\alpha' = (\alpha'_1, \ldots, \alpha'_h)$. Zu $a \in A$ sei*

$$vz(X^{(1)}, f, a) = |\{f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}} \mid \alpha'(\epsilon_1, \ldots, \epsilon_p) = a\}|.$$

Betrachtet man alle Zeilen von $Z(X^{(1)})$, deren Index durch α' der Wert a zugeordnet wird, so gibt $vz(X^{(1)}, f, a)$ also die Anzahl der verschiedenen Zeilenmuster in diesen Zeilen an. $vz(X, f, \alpha')$ ist definiert als das Maximum

$$vz(X, f, \alpha') = \max_{a \in A}(vz(X^{(1)}, f, a)).$$

Beweis: Der Beweis erfolgt im wesentlichen analog zum Beweis von Satz 3.1.

„$\Longrightarrow$“: Sei eine Zerlegung gegeben durch

$$f(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) = g(\alpha'_1(\mathbf{x^{(1)}}), \ldots, \alpha'_h(\mathbf{x^{(1)}}), \alpha_{h+1}(\mathbf{x^{(1)}}), \ldots, \alpha_r(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}).$$

Die Funktion

$$\alpha'' = (\alpha_{h+1}, \ldots, \alpha_r) \ \in B_{p,r-h}$$

kann genau 2^{r-h} verschiedene Funktionswerte annehmen.
Angenommen

$$vz(X^{(1)}, f, \alpha') > 2^{r-h}.$$

Dann muß es ein $a \in \{0,1\}^h$ im Bild von α', $\epsilon^{(1)} = (\epsilon_1^{(1)}, \ldots, \epsilon_p^{(1)})$ und $\epsilon^{(2)} = (\epsilon_1^{(2)}, \ldots, \epsilon_p^{(2)}) \in \{0,1\}^p$ geben, so daß

$$\alpha'(\epsilon^{(1)}) = \alpha'(\epsilon^{(2)}) = a,$$

$$\alpha''(\epsilon^{(1)}) = \alpha''(\epsilon^{(2)}),$$

aber die zu $\epsilon^{(1)}$, $\epsilon^{(2)}$ gehörigen Zeilen verschieden sind.
Es kann also keine Zerlegung geben mit

$$\alpha'_1, \ldots, \alpha'_h, \alpha_{h+1}, \ldots, \alpha_r$$

als Zerlegungsfunktionen.

„$\Longleftarrow$“: Sei

$$vz(X^{(1)}, f, \alpha') \leq 2^{r-h}.$$

Betrachte ein a aus dem Bild von α'.
Sei $X_a^{(1)} \subseteq \{0,1\}^p$ die Menge aller Urbilder von a.
Dann kann man $\alpha''_a : X_a^{(1)} \to \{0,1\}^{r-h}$ so definieren, daß

$$\alpha''_a(\epsilon^{(1)}) \neq \alpha''_a(\epsilon^{(2)})$$

für alle $\epsilon^{(1)}, \epsilon^{(2)} \in X_a^{(1)}$, für die die zugehörigen Zerlegungsmatrixzeilen verschieden sind.
Definiert man auf diese Weise für alle a aus dem Bild von α' eine Funktion α''_a, so kann man diese Funktionen zu einer Funktion α'' auf $\{0,1\}^p$ zusammensetzen. Es gilt dann:

$$(\alpha', \alpha'')(\epsilon^{(1)}) \neq (\alpha', \alpha'')(\epsilon^{(2)})$$

für alle $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$, für die die zugehörigen Zerlegungsmatrixzeilen verschieden sind.

□

Aus dem Kriterium aus Lemma 3.9 für Funktionen mit einem Ausgang ergibt sich leicht ein entsprechendes Kriterium für Funktionen mit mehreren Ausgängen:

Lemma 3.10 *Seien $f_1, \ldots, f_m \in B_n$. Seien die Eingangsvariablen von $f_1, \ldots, f_m$ $x_1, \ldots, x_n$, sei $X^{(1)} = \{x_1, \ldots, x_p\}$ und sei für $1 \leq i \leq m$ $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$. Dann können $\alpha_1, \ldots, \alpha_h$ genau dann alle als Zerlegungsfunktionen in einseitigen kommunikationsminimalen Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ verwendet werden, d.h. es gibt genau dann einseitige Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ der Form*

$$\begin{aligned}
f_1(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(1)}(\alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \alpha^{(1)}_{h+1}(\mathbf{x^{(1)}}), \ldots, \alpha^{(1)}_{r_1}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\
&\vdots \\
f_m(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(m)}(\alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \alpha^{(m)}_{h+1}(\mathbf{x^{(1)}}), \ldots, \alpha^{(m)}_{r_m}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}),
\end{aligned}$$

wenn für alle $1 \leq i \leq m$ gilt:

$$vz(X^{(1)}, f_i, \alpha) \leq 2^{r_i - h}. \qquad (\alpha = (\alpha_1, \ldots, \alpha_h).)$$

Bei der Suche nach Zerlegungsfunktionen wird häufig die Situation auftreten, daß für die Zerlegung von Funktionen $f_1, \ldots, f_m$ ein Teil der Zerlegungsfunktionen schon bestimmt ist und man möglichst viele der restlichen Zerlegungsfunktionen für $f_1, \ldots, f_m$ gemeinsam wählen will (vergleiche Abschnitt 3.4.2.2). In diesem Zusammenhang ist die Aussage von Lemma 3.11 von Bedeutung, das eine einfache Folgerung aus Lemma 3.10 darstellt. Vorher wird noch eine Bezeichnung vereinbart:

Bezeichnung 3.7 *Sei eine Funktion $f_i \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_n$ gegeben. Sei $Z_i(X^{(1)})$ die Zerlegungsmatrix von f_i hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$. Sei weiterhin die durch die Gleichheit von Zerlegungsmatrixzeilen in $Z_i(X^{(1)})$ auf $\{0,1\}^p$ induzierte Äquivalenzklasseneinteilung $\{K_1^{(i)}, \ldots, K^{(i)}_{vz(X^{(1)}, f_i)}\}$. Seien $\alpha_1^{(i)}, \ldots, \alpha^{(i)}_{k_i}, \alpha_1, \ldots, \alpha_h \in B_p$.*
Es gelte weiterhin $k_i + h \leq r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$.
Für $a \in \{0,1\}^{k_i}$ aus dem Bild von $\alpha^{(i)}$ und $a' \in \{0,1\}^h$ aus dem Bild von α sei $X^{(1)}_{aa'} \subseteq \{0,1\}^p$ die Menge der Urbilder von (a, a') bzgl. $(\alpha^{(i)}, \alpha)$.

Dann wird mit $S^{(i)}_{aa'}$ folgende Menge bezeichnet:

$$S^{(i)}_{aa'} = \{1 \leq j \leq vz(X^{(1)}, f_i) \mid K_j^{(i)} \cap X^{(1)}_{aa'} \neq \emptyset\}$$

Betrachtet man alle binären Zeilenindizes $x \in \{0,1\}^p$ von $Z_i(X^{(1)})$, für die $(\alpha^{(i)}, \alpha)(x) = (aa')$ ist und dazu alle Äquivalenzklassen (hinsichtlich Zeilengleichheit) $K^{(i)}_{j_1}, \ldots, K^{(i)}_{j_l}$, in die diese binären Zeilenindizes fallen, so umfaßt $S_{aa'}^{(i)}$ gerade die Indizes $j_1, \ldots, j_l$. ($|S_{aa'}^{(i)}|$ gibt also die Anzahl der *verschiedenen* Zeilen von $Z_i(X^{(1)})$ an, auf deren Indizes $(\alpha^{(i)}, \alpha)$ den Wert (aa') liefert. Mit den obigen Bezeichnungen gilt also $|S_{aa'}^{(i)}| = vz(X^{(1)}, f, (aa'))$.)

Mit der angegebenen Bezeichnung ergibt sich dann Lemma 3.11:

Lemma 3.11 *Seien $f_1, \ldots, f_m$ Funktionen aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$. Für $1 \leq i \leq m$ sei $Z_i(X^{(1)})$ die Zerlegungsmatrix von f_i hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$ und $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$.*
Für alle $1 \leq i \leq m$ seien Zerlegungsfunktionen

$$\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)} \in B_p$$

vorgegeben. Dann sind unter diesen Voraussetzungen

$$\alpha_1, \ldots, \alpha_h \in B_p$$

genau dann gemeinsame Zerlegungsfunktionen bzgl. einer kommunikationsminmalen Zerlegung von $f_1, \ldots, f_m$, d.h. es gibt genau dann einseitige Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ der Form

$$\begin{aligned}
f_1(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(1)}(\alpha_1^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_1}^{(1)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_1+h+1}^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\
&\vdots \\
f_m(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(m)}(\alpha_1^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_m}^{(m)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_m+h+1}^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_m}^{(m)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}),
\end{aligned}$$

wenn für alle $1 \leq i \leq m$ gilt:
Für alle $a \in \{0,1\}^{k_i}$ aus dem Bild von $\alpha^{(i)} = (\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)})$ und $a' \in \{0,1\}^h$ aus dem Bild von $\alpha = (\alpha_1, \ldots, \alpha_h)$ ist

$$|S_{aa'}^{(i)}| \leq 2^{r_i - k_i - h}.$$

Lemma 3.11 liefert also ein notwendiges und hinreichendes Kriterium für die Existenz gemeinsamer Zerlegungsfunktionen bei der kommunikationsminimalen Zerlegung von Funktionen $f_1, \ldots, f_m$, bei denen evtl. schon gewisse Zerlegungsfunktionen fest vorgegeben sind.

Nach der Herleitung der für die Berechnung gemeinsamer Zerlegungsfunktionen grundlegenden Lemmata 3.9, 3.10 und 3.11 soll im folgenden Abschnitt die Komplexität des zu lösenden Problems untersucht werden und ein Algorithmus zur Lösung angegeben werden.

3.4.2.1 Das Problem CDF

Folgendes Problem ist bei der Berechnung gemeinsamer Zerlegungsfunktionen zu lösen:

Problem CDF (Common Decomposition Functions)

Gegeben: Funktionen $f_1, \ldots, f_m \in B_n$ mit den Eingangsvariablen $x_1, \ldots, x_n$ und eine Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$. Die minimale Anzahl der Zerlegungsfunktionen, die bei Zerlegung von f_i hinsichtlich $X^{(1)}$ benötigt werden, betrage für $1 \le i \le m$ $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$. Eine natürliche Zahl h mit $h \le r_i$ für alle $1 \le i \le m$.

Gesucht: Gibt es Funktionen $\alpha_1, \ldots, \alpha_h \in B_p$, die alle als Zerlegungsfunktionen in einseitigen Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ verwendet werden können, d.h. gibt es einseitige Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ der Form

$$\begin{aligned} & f_1(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) = \\ & \qquad g^{(1)}(\alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \alpha_{h+1}^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\ & \quad \vdots \\ & f_m(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) = \\ & \qquad g^{(m)}(\alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \alpha_{h+1}^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_m}^{(m)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}), \end{aligned}$$

Es gilt:

Satz 3.10 *Das Problem* **CDF** *ist NP–hart.*

Der Beweis von Satz 3.10 erfolgt durch Polynomzeittransformation ausgehend von Problem 3–PARTITION (siehe [GJ79]). Er ist in Anhang A angegeben. Da das Problem NP–hart ist, wenn $f_1, \ldots, f_m$ durch Funktionstabellen gegeben sind, ist es auch NP–hart, wenn $f_1, \ldots, f_m$ durch ROBDDs gegeben sind.

Aus dem Beweis in Anhang A ergeben sich folgende beiden Korollare:

Korollar 3.5 *Das Problem CDF bleibt NP–hart, auch wenn die Anzahl der gegebenen Booleschen Funktionen gleich 2 ist.*

Korollar 3.6 *Das Problem CDF bleibt NP–hart, auch wenn man sich bei der Wahl von Zerlegungsfunktionen beschränkt auf strikte Zerlegungsfunktionen.*

Die Gültigkeit von Korollar 3.6 ergibt sich mit Hilfe von Lemma 3.5 auf Seite 95: Für die im Beweis von Satz 3.10 konstruierten Funktionen f_1 und f_2 (siehe Anhang A) sind $vz(X^{(1)}, f_1)$ und $vz(X^{(1)}, f_2)$ gerade Zweierpotenzen, so daß in einer kommunikationsminimalen Zerlegung nur *strikte* Zerlegungsfunktionen vorkommen können.

Prinzipielle Lösung von CDF Ausgehend von dem Kriterium aus Lemma 3.11 wird zunächst ein branch–and–bound–Algorithmus entwickelt, der das Problem CDF prinzipiell löst. Dieser Algorithmus wird in dem implementierten Logiksyntheseverfahren *nicht* verwendet und soll im wesentlichen zum Verständnis des im nächsten Abschnitt daraus entwickelten modifizierten branch–and–bound–Algorithmus dienen. Der Algorithmus baut die Funktionstabelle von $\alpha = (\alpha_1, \ldots, \alpha_h)$ schrittweise auf und testet für die schon aufgestellte Teiltabelle, ob das Kriterium aus Lemma 3.11 evtl. schon verletzt wird. Ist dies nicht der Fall, so wird der Funktionswert der nächsten Zeile der Funktionstabelle festgelegt, ansonsten wird die Belegung für *diese* Zeile (und, falls für diese Zeile schon alle möglichen Funktionswerte getestet wurden, auch für *vorangehende* Zeilen) zurückgenommen. Verletzt ein Anfangsstück der Funktionstabelle von $\alpha = (\alpha_1, \ldots, \alpha_h)$ die Bedingung aus Lemma 3.11, so verletzen natürlich auch alle Fortsetzungen diese Bedingung und brauchen in der Folge nicht mehr betrachtet zu werden. Im folgenden wird das Verfahren genauer beschrieben:

Eingabe:
- Funktionen $f_1, \ldots, f_m$ aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$.
- Für $1 \le i \le m$: Zerlegungsmatrizen $Z_i(X^{(1)})$ von f_i hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$ und $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$.
- Für $1 \le i \le m$: Sei $\{K_1^{(i)}, \ldots, K_{vz(X^{(1)}, f_i)}^{(i)}\}$ die durch Zeilengleichheit in $Z_i(X^{(1)})$ auf $\{0,1\}^p$ induzierte Äquivalenzklasseneinteilung. Sei $clnr_i$ eine Funktion, die jedem v aus einer Äquivalenzklasse $K_j^{(i)}$ den Index j zuordnet.
- Für $1 \le i \le m$: Zerlegungsfunktionen $\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)} \in B_p$ $(k_i < r_i)$.
- Natürliche Zahl h mit $h \le r_i - k_i$ für alle $1 \le i \le m$. (h gibt die Anzahl gemeinsamer Zerlegungsfunktionen an, nach denen gesucht wird.)

Ausgabe:
- $\alpha_1, \ldots, \alpha_h \in B_p$, so daß es einseitige Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ gibt der Form

$$
\begin{aligned}
f_1(x_1, \ldots, x_n) &= g^{(1)}(\alpha_1^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_1}^{(1)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_1+h+1}^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\
&\ \vdots \\
f_m(x_1, \ldots, x_n) &= g^{(m)}(\alpha_1^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_m}^{(m)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_m+h+1}^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_m}^{(m)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}),
\end{aligned}
$$

  falls es überhaupt h Funktionen aus B_p mit dieser Eigenschaft gibt.
- Sonst: Eine Meldung daß es keine h Funktionen aus B_p mit der angegebenen Eigenschaft gibt.

Algorithmus: Siehe Abbildung 3.18.

Für alle $1 \leq i \leq m, a \in \{0,1\}^{k_i}, a' \in \{0,1\}^h : S^{(i)}_{aa'} = \emptyset$
$\forall v \in \{0,1\}^p : \alpha(v) := undef.$
$\alpha(0,\ldots,0) := (0,\ldots,0)$
$\forall 1 \leq i \leq m : S^{(i)}_{\alpha^{(i)}(0,\ldots,0)(0,\ldots,0)} = S^{(i)}_{\alpha^{(i)}(0,\ldots,0)(0,\ldots,0)} \cup \{clnr_i(0,\ldots,0)\}$
$v = (0,\ldots,0,1) \in \{0,1\}^p$
$a' = (0,\ldots,0) \in \{0,1\}^h$
while $(\exists v \in \{0,1\}^p$ mit $\alpha(v) = undef.)$ **do**
    /* Funktionstabelle von α noch nicht für alle $v \in \{0,1\}^p$ aufgestellt */
    $\alpha(v) = a'$
    if ($\forall 1 \leq i \leq m$ |„$S^{(i)}_{\alpha^{(i)}(v)a'} \cup \{clnr_i(v)\}$“| $\leq 2^{r_i - k_i - h}$)
      then
          $\forall 1 \leq i \leq m$: „$S^{(i)}_{\alpha^{(i)}(v)a'} = S^{(i)}_{\alpha^{(i)}(v)a'} \cup \{clnr_i(v)\}$“
          Erhöhe v um 1.
          $a' = (0,\ldots,0) \in \{0,1\}^h$
       else
          while $(\alpha(v) = (1,\ldots,1))$ **do**
              $\alpha(v) = undef.$
              Verringere v um 1.
              $\forall 1 \leq i \leq m$: „$S^{(i)}_{\alpha^{(i)}(v)\alpha(v)} = S^{(i)}_{\alpha^{(i)}(v)\alpha(v)} \setminus \{clnr_i(v)\}$“
          od
          $a' = \alpha(v)$
          Erhöhe a' um 1.
          $\alpha(v) = undef.$
    fi
    if $(v = (0,\ldots,0))$
      /* Man ist am Ausgangspunkt angekommen ist, ohne eine geeignete
        Funktion α gefunden zu haben */
      then
          return „Keine Lösung für α.“
    fi
od
return α

Abbildung 3.18: Prinzipieller Algorithmus zur Lösung von CDF.

Anmerkung zum Algorithmus:
Die Operationen „$\cup$" und „$\setminus$" in den Zeilen 10, 12 und 19 wurden der Einfachheit halber etwas ungenau angegeben. Wird ein Eintrag an der Stelle v in der Funktionstabelle von α wieder gelöscht, so müssen für alle $1 \leq i \leq m$ die Mengen $S^{(i)}_{\alpha^{(i)}(v)\alpha(v)}$ korrigiert werden. Allerdings darf der Äquivalenzklassenindex $clnr_i(v)$ nur dann aus $S^{(i)}_{\alpha^{(i)}(v)\alpha(v)}$ entfernt werden, wenn es keine vorangehende Stelle v' in der Funktionstabelle gibt mit $\alpha^{(i)}(v') = \alpha^{(i)}(v)$, $\alpha(v') = \alpha(v)$ und $clnr_i(v) = clnr_i(v')$, so daß also $clnr_i(v)$ wegen dieser Stelle in $S^{(i)}_{\alpha^{(i)}(v)\alpha(v)}$ bleiben muß. Der Index $clnr_i(v)$ darf erst dann endgültig aus der Menge $S^{(i)}_{\alpha^{(i)}(v)\alpha(v)}$ entfernt werden, wenn er genauso oft „entfernt" wurde, wie er „hinzuvereinigt" wurde. Die Operationen „$\cup$" und „$\setminus$" in Zeilen 10, 12 und 19 des Algorithmus sind also als Operationen auf Multimengen zu implementieren.

Einige Punkte des Algorithmus sollen noch kurz erläutert werden:

Zeile 3: Der Funktionswert für α an der Stelle $(0, \ldots, 0)$ wird o.B.d.A. auf $(0, \ldots, 0)$ festgelegt. Falls es eine Funktion α gibt mit den gewünschten Eigenschaften, so gibt es auch eine mit $\alpha(0, \ldots, 0) = (0, \ldots, 0)$. (Denn gibt es allgemein eine Zerlegung mit den Zerlegungsfunktionen $\alpha'_1, \ldots, \alpha'_i, \ldots, \alpha'_r$, so gibt es auch eine mit den Zerlegungsfunktionen $\alpha'_1, \ldots, \overline{\alpha'_i}, \ldots, \alpha'_r$.)

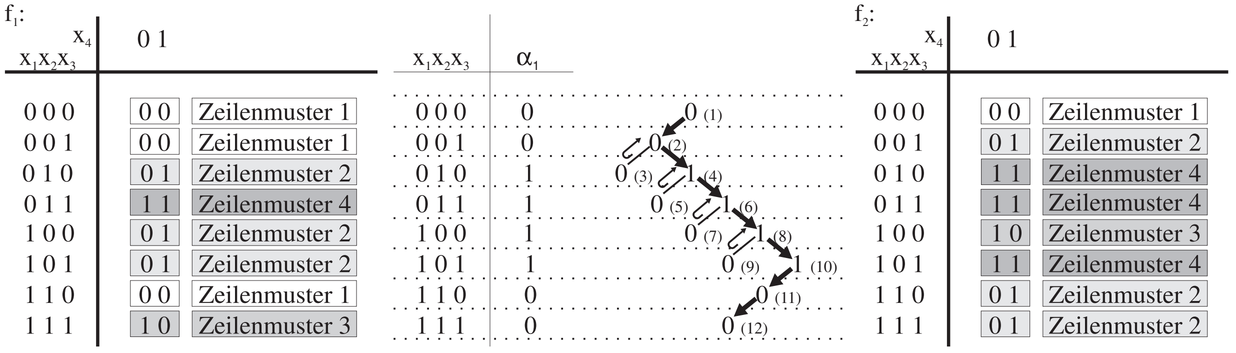
Zeilen 7–31: Die Funktionstabelle zu α wird schrittweise aufgebaut.
Wird die Bedingung von Lemma 3.11 durch das Anfangsstück der Funktionstabelle *nicht* verletzt, so versucht man, dieses Anfangsstück fortzusetzen zu einer vollständigen Funktionstabelle von α (Zeilen 10–14).
Wird die Bedingung von Lemma 3.11 schon von dem aktuellen Anfangsstück der Funktionstabelle verletzt, so werden alle Fortsetzungen dieser Tabelle die Bedingung verletzen. Also muß „in einen anderen Ast des branch–and–bound–Verfahrens verzweigt werden" (Zeilen 16–23).
Zu Beginn des Durchlauf der **while**–Schleife (Zeilen 7–31) ist ein Anfangsstück der Funktionstabelle von α schon aufgebaut: Die Funktionswerte $\alpha(0, \ldots, 0)$ bis $\alpha(v-1)$[6] sind festgelegt und das aktuelle Anfangsstück der Funktionstabelle verletzt die Bedingung aus Lemma 3.11 noch nicht. Falls $a' > (0, \ldots, 0)$, so wurde vorher bereits erfolglos versucht, $\alpha(v)$ mit den Funktionswerten zu belegen, die kleiner als a' sind.
Zunächst wird getestet, ob bei einer Belegung $\alpha(v) = a'$ die Bedingung aus Lemma 3.11 schon verletzt wird. Ist dies nicht der Fall, so bleibt diese Belegung bestehen, für den nächsten Durchlauf der **while**–Schleife wird v um 1 erhöht und a' auf $(0, \ldots, 0)$ zurückgesetzt.
Wird die Bedingung verletzt, so wird a' $(= \alpha(v))$ um 1 erhöht, der Funktionswert von $\alpha(v)$ wird wieder auf „undefiniert" gesetzt und im nächsten Durchlauf der Schleife wird eine Belegung von $\alpha(v)$ mit dem neuen a' versucht. Tritt allerdings der Fall auf,

[6] An dieser Stelle ist für $v \in \{0,1\}^p$ mit $v-1$ das $v' \in \{0,1\}^p$ gemeint mit $int(v') = int(v) - 1$. Ebenso gelte für v_1 und $v_2 \in \{0,1\}^p$ $v_1 < v_2$ genau dann, wenn $int(v_1) < int(v_2)$.

daß a' $(= \alpha(v))$ schon gleich $(1, \ldots, 1)$ ist, so wird die nächstkleinere Stelle v' in der Funktionstabelle von α gesucht, für die $\alpha(v')$ noch nicht gleich $(1, \ldots, 1)$ ist. Für v' und alle größeren schon belegten Stellen v der Funktionstabelle wird die Belegung rückgängig gemacht (einschließlich einer Korrektur der Mengen $S^{(i)}_{\alpha^{(i)}(v)\alpha(v)}$) (siehe Zeilen 16–20). Nach Abarbeitung der **while**–Schleife ist v für den nächsten Durchlauf auf dieses v' gesetzt. Damit im nächsten Durchlauf eine Belegung von $\alpha(v)$ mit dem um 1 erhöhten Wert versucht wird, wird a' noch entsprechend belegt.

Aus Laufzeitgründen berechnet das branch–and–bound–Verfahren nur *einen* erfolgreichen Ast, d.h. nur ein h–Tupel von Zerlegungsfunktionen, die das gestellte Problem lösen.

Beispiel 3.14 Der Algorithmus wird durch ein Beispiel illustriert. Es handelt sich dabei um die Funktionen f_1 und f_2 aus Beispiel 3.11 (Seite 140), für die bei kommunikationsminimaler Zerlegung hinsichtlich $\{x_1, x_2, x_3\}$ genau eine gemeinsame Zerlegungsfunktion gesucht wird. Es gilt $f_1(x_1, \ldots, x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$ und $f_2(x_1, \ldots, x_4) = \overline{x_4}(x_1 \oplus x_2) + x_4(x_2 + x_3)$.
Aufgrund der Gleichheit von Zerlegungsmatrixzeilen in $Z_1(\{x_1, x_2, x_3\})$ für f_1 ergibt sich eine Äquivalenzklasseneinteilung von $\{0,1\}^3$ mit den Äquivalenzklassen $K^{(1)}_1 = \{(000), (001), (110)\}$, $K^{(1)}_2 = \{(010), (100), (101)\}$, $K^{(1)}_3 = \{(111)\}$ und $K^{(1)}_4 = \{(011)\}$. Analog ergibt sich für f_2 eine Äquivalenzklasseneinteilung mit den Klassen $K^{(2)}_1 = \{(000)\}$, $K^{(2)}_2 = \{(001), (110), (111)\}$, $K^{(2)}_3 = \{(100)\}$ und $K^{(2)}_4 = \{(010), (011), (101)\}$.
Bei kommunikationsminimaler Zerlegung werden für f_1 und f_2 jeweils 2 Zerlegungsfunktionen benötigt ($r_1 = r_2 = 2$ mit den obigen Bezeichnungen).
Es seien noch keine Zerlegungsfunktionen vorgegeben ($k_1 = k_2 = 0$ mit den obigen Bezeichnungen).
Sucht man eine gemeinsame Zerlegungsfunktion α_1 für f_1 und f_2 ($h = 1$), so ist sowohl für f_1 als auch für f_2 noch genau eine Zerlegungsfunktion übrig.
Die in der Bedingung von Lemma 3.11 zu betrachtenden Mengen sind $S^{(1)}_0$ und $S^{(1)}_1$ für f_1 bzw. $S^{(2)}_0$ und $S^{(2)}_1$ für f_2. Betrachtet man die Zeilenindizes der Zerlegungsmatrixzeilen von $Z_i(\{x_1, x_2, x_3\})$ $(i = 1, 2)$, denen durch α_1 der Wert 0 (bzw. 1) zugeordnet wird, so dürfen unter den zugehörigen Zeilenmustern höchstens 2 verschiedene Muster vorkommen ($|S^{(i)}_0| \leq 2$ (bzw. $|S^{(i)}_1| \leq 2$)). Ist dies der Fall, so kann mit der verbleibenden Zerlegungsfunktion $\alpha^{(i)}_2$ dafür gesorgt werden, daß Zeilenindizes von Zeilen mit *verschiedenen* Zeilenmustern durch $(\alpha_1, \alpha^{(i)}_2)$ auch *verschiedene* Zerlegungsfunktionswerte zugeordnet werden.
In Abbildung 3.19 ist der Ablauf des branch–and–bound–Algorithmus zur Bestimmung einer gemeinsamen Zerlegungsfunktion von f_1 und f_2 graphisch dargestellt. In der Mitte der Abbildung ist der Teil des Entscheidungsbaumes dargestellt, der von dem Algorithmus durchlaufen wird. Die durchgeführten Belegungen sind der Reihe nach durchnumeriert. Der Pfad aus fett gezeichneten Pfeilen entspricht dem berechneten Endergebnis für α. Die restlichen Kanten entsprechen Entscheidungen, die aufgrund einer Verletzung der Bedingung aus Lemma 3.11 wieder zurückgenommen wurden. Es ergibt sich schließlich $\alpha_1 : \{0,1\}^3 \to \{0,1\}$ mit $\alpha_1(v) = 1 \iff v \in \{(010), (011), (100), (101)\}$. Im unteren



Mengen $S_0^{(i)}$, $S_1^{(i)}$ im Verlauf des branch-and-bound-Algorithmus:

(1)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{\}$	$S_0^{(2)} = \{1\}$, $S_1^{(2)} = \{\}$	
(2)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{\}$	
(3)	$S_0^{(1)} = \{1, 2\}$, $S_1^{(1)} = \{\}$	$S_0^{(2)} = \{1, 2, 4\}$, $S_1^{(2)} = \{\}$	↯
(4)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{2\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{4\}$	
(5)	$S_0^{(1)} = \{1, 4\}$, $S_1^{(1)} = \{2\}$	$S_0^{(2)} = \{1, 2, 4\}$, $S_1^{(2)} = \{4\}$	↯
(6)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{4\}$	
(7)	$S_0^{(1)} = \{1, 2\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2, 3\}$, $S_1^{(2)} = \{4\}$	↯
(8)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{3, 4\}$	
(9)	$S_0^{(1)} = \{1, 2\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2, 4\}$, $S_1^{(2)} = \{3, 4\}$	↯
(10)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{3, 4\}$	
(11)	$S_0^{(1)} = \{1\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{3, 4\}$	
(12)	$S_0^{(1)} = \{1, 3\}$, $S_1^{(1)} = \{2, 4\}$	$S_0^{(2)} = \{1, 2\}$, $S_1^{(2)} = \{3, 4\}$	

Abbildung 3.19: Beispiel zu branch–and–bound–Algorithmus

Teil der Abbildung sind die Mengen $S_0^{(1)}$, $S_1^{(1)}$, $S_0^{(2)}$ und $S_1^{(2)}$ für die einzelnen Schritte des Algorithmus dargestellt.

Lösung von CDF bei strikten Zerlegungsfunktionen Der oben angegebene Algorithmus hat noch den Nachteil, daß er evtl. auch zu nichtstrikten Zerlegungsfunktionen führt. In Abschnitt 3.1.3.2 wurde aber schon festgestellt, daß *strikte* Zerlegungsfunktionen einer Funktion f den Vorteil haben, daß sie Symmetrieeigenschaften von f erhalten. Im vorliegenden Abschnitt wird der branch–and–bound–Algorithmus so verändert, daß er nur noch strikte Zerlegungsfunktionen liefert.

Dabei ergibt sich der zusätzliche Vorteil, daß die Beschränkung auf gemeinsame, *strikte* Zerlegungsfunktionen auch erheblich zur Beschleunigung der Berechnung beitragen kann, da die Effizienz des Algorithmus durch Vorberechnungen gesteigert werden kann.

Sucht man nämlich bei der Zerlegung zweier Funktionen f_i und f_j (mit den Äquivalenzklasseneinteilungen $\{K_1^{(i)}, \ldots, K_{vz(X^{(1)}, f_i)}^{(i)}\}$ für f_i und $\{K_1^{(j)}, \ldots, K_{vz(X^{(1)}, f_j)}^{(j)}\}$ für f_j) eine gemeinsame, strikte Zerlegungsfunktion α_k von f_i und f_j, so muß für ϵ und δ aus $\{0,1\}^p$ $\alpha_k(\epsilon) = \alpha_k(\delta)$ gelten, wenn $\epsilon, \delta \in K_l^{(i)}$ für beliebiges $1 \leq l \leq vz(X^{(1)}, f_i)$ *oder* $\epsilon, \delta \in K_l^{(j)}$ für beliebiges $1 \leq l \leq vz(X^{(1)}, f_j)$. Man erhält bei der Berechnung gemeinsamer, strikter Zerlegungsfunktionen also im allgemeinen größere Teilmengen von $\{0,1\}^p$, von denen man aussagen kann, daß gemeinsame, strikte Zerlegungsfunktionen auf diesen Teilmengen gleiche Funktionswerte liefern müssen.

Bezeichnung 3.8 *Sei $f \in B_{n,m}$ mit den Eingangsvariablen $x_1, \ldots, x_n$. f werde zerlegt hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$.*

- *Für alle $1 \leq i \leq m$ sei eine Äquivalenzrelation $\equiv_i$ auf $\{0,1\}^p$ definiert mit*

  $$\epsilon^{(1)} \equiv_i \epsilon^{(2)} \iff (f_i)_{x_1^{\epsilon_1^{(1)}} \ldots x_p^{\epsilon_p^{(1)}}} = (f_i)_{x_1^{\epsilon_1^{(2)}} \ldots x_p^{\epsilon_p^{(2)}}}$$

  für alle $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$.
  Die zugehörige Äquivalenzklasseneinteilung wird mit

  $$\{0,1\}^p/_{\equiv_i} = \{K_1^{(i)}, \ldots, K_{vz(X^{(1)},f_i)}^{(i)}\}$$

  bezeichnet.

- *Sei die Relation $\sim'$ auf $\{0,1\}^p$ definiert durch*

  $$\epsilon^{(1)} \sim' \epsilon^{(2)} \iff \exists 1 \leq i \leq m \text{ mit } \epsilon^{(1)} \equiv_i \epsilon^{(2)}$$

  für alle $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$.
  Sei die Äquivalenzrelation $\sim$ der transitive Abschluß von $\sim'$.
  Die zugehörige Äquivalenzklasseneinteilung wird mit

  $$\{0,1\}^p/_{\sim} = \{E_1, \ldots, E_l\}$$

  bezeichnet.

Bezeichnung 3.9 *Man bezeichnet eine Partition $P = \{P_1, \ldots, P_k\}$ genau dann als* Vergröberung *der Partition $Q = \{Q_1, \ldots, Q_l\}$ (bzw. Q als* Verfeinerung *von P), wenn $\cup_{i=1}^k P_i = \cup_{i=1}^l Q_i$ und es für alle Q_i $(1 \leq i \leq l)$ ein $j \in \{1, \ldots, k\}$ gibt mit $Q_i \subseteq P_j$.*

Bemerkung 3.6 *Die Partition $\{0,1\}^p/_{\sim}$ ist die „feinste" gemeinsame Vergröberung der Partitionen $\{0,1\}^p/_{\equiv_i}$ $(1 \leq i \leq m)$. Betrachtet man die Partitionen auf $\{0,1\}^p$ als Verband, wobei für zwei Partitionen P und Q genau dann $P \leq Q$ gilt, wenn P eine Vergröberung von Q ist, dann ist $\{0,1\}^p/_{\sim}$ einfach das Infimum über alle $\{0,1\}^p/_{\equiv_i}$ $(1 \leq i \leq m)$.*

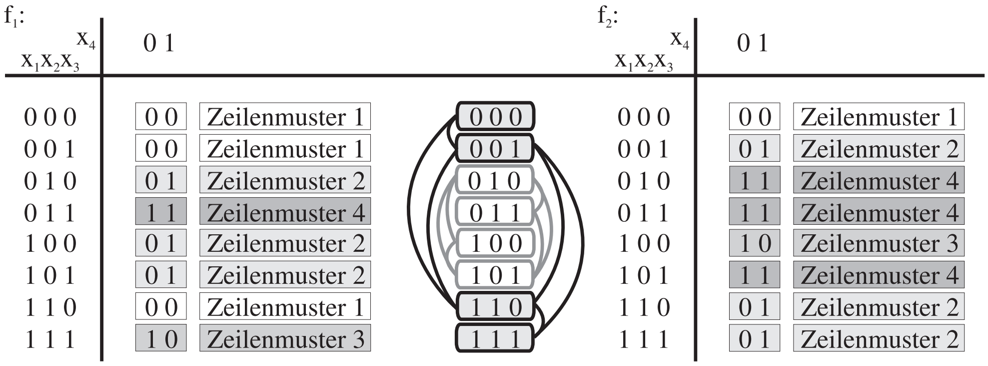


Abbildung 3.20: Berechnung von $\{0,1\}^p/_\sim$.

Eine strikte Zerlegungsfunktion von f_i muß allen Elementen einer Klasse $K_j^{(i)}$ aus $\{0,1\}^p/_{\equiv_i}$ die gleichen Funktionswerte zuordnen. Folglich muß eine gemeinsame, strikte Zerlegungsfunktion von $f_1,\ldots,f_m$ allen Elementen einer Klasse E_j aus $\{0,1\}^p/_\sim$ die gleichen Funktionswerte zuordnen.

Die Äquivalenzklasseneinteilung $\{0,1\}^p/_\sim$ läßt sich anhand der Zerlegungsmatrizen von $f_1,\ldots,f_m$ leicht bestimmen. Dies wird anhand eines Beispiels verdeutlicht:

Beispiel 3.15 In Abbildung 3.20 ist für die Funktionen f_1 und f_2 aus Beispiel 3.14 (Seite 157) die Bestimmung von $\{0,1\}^3/_\sim$ bei Zerlegung hinsichtlich $\{x_1,x_2,x_3\}$ dargestellt. Die Elemente von $\{0,1\}^3$ sind als Knoten eines Graphen dargestellt, wobei es zwischen 2 Knoten ϵ und δ genau dann eine Kante gibt, wenn $\epsilon \equiv_1 \delta$ (Kanten auf der linken Seite) oder wenn $\epsilon \equiv_2 \delta$ (Kanten auf der rechten Seite). $\{0,1\}^3/_\sim$ erhält man durch Bestimmung der Zusammenhangskomponenten in diesem Graphen. Es gibt genau zwei Zusammenhangskomponenten (weiße und grau schattierte Knoten), so daß sich für $\{0,1\}^3/_\sim$

$$\{0,1\}^3/_\sim = \{\{(000),(001),(110),(111)\},\{(010),(011),(100),(101)\}\}$$

ergibt.

Ist α_1 also eine gemeinsame, strikte Zerlegungsfunktion von f_1 und f_2, so ist mit Angabe des Funktionswertes von α_1 auf (000) sofort auch der Funktionswert auf (001), (110) und (111) festgelegt.

Will man für die Zerlegung von Funktionen $f_1,\ldots,f_m$ gemeinsame, strikte Zerlegungsfunktionen $\alpha_1,\ldots,\alpha_h$ durch ein branch–and–bound–Verfahren bestimmen, so kann man von dem Wissen Gebrauch machen, daß $\alpha_1,\ldots,\alpha_h$ *allen* Elementen einer Klasse E_j aus $\{0,1\}^p/_\sim$ den gleichen Wert zuordnen. Wie im obigen Beispiel gezeigt, läßt sich die Äquivalenzklasseneinteilung $\{0,1\}^p/_\sim$ leicht durch Bestimmung der Zusammenhangskomponenten eines (impliziten) Graphen mit Knotenmenge $\{0,1\}^p$ bestimmen.

Ein modifiziertes branch–and–bound–Verfahren baut die Funktionstabelle für $\alpha = (\alpha_1, \ldots, \alpha_h)$ nicht Zeile für Zeile auf, sondern weist den *einzelnen Äquivalenzklassen* E_j aus $\{0,1\}^p/_\sim$ Funktionswerte zu. Dadurch wird der Aufwand wesentlich verringert.

Auf den nächsten beiden Seiten sind die genaueren Änderungen des Verfahrens dargestellt, die sich aus der Verwendung *strikter* Zerlegungsfunktionen ergeben. Zunächst werden die benötigten Bezeichnungen definiert und schließlich das eigentliche Verfahren angegeben. Danach wird die Wirkungsweise des Algorithmus anhand eines Beispiels demonstriert.

Das ursprüngliche branch–and–bound–Verfahren berechnet bei Vorgabe von Zerlegungsfunktionen $(\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)}) = \alpha^{(i)}$ für jede der Funktionen f_i gemeinsame Zerlegungsfunktionen $\alpha_1, \ldots, \alpha_h$ und verwaltet dabei Mengen $S_{aa'}^{(i)}$. Wird der Funktionswert von α für ein weiteres Element $v \in \{0,1\}^p$ z.B. auf den Wert a' festgelegt, so wird $S_{\alpha^{(i)}(v)a'}^{(i)}$ aktualisiert durch

$$„S_{\alpha^{(i)}(v)\alpha(v)}^{(i)} = S_{\alpha^{(i)}(v)\alpha(v)}^{(i)} \cup \{clnr_i(v)\}“$$

($clnr_i(v) = j$ mit $v \in K_j^{(i)}$, vgl. Zeile 12 in Abbildung 3.18).

Wird also im modifizierten branch–and–bound–Verfahren für eine Menge E_j $\alpha(E_j) = a'$ festgelegt ($a' \in \{0,1\}^h$), so muß für alle $v \in E_j$ und alle $1 \leq i \leq m$ die Operation

$$„S_{\alpha^{(i)}(v)a'}^{(i)} = S_{\alpha^{(i)}(v)a'}^{(i)} \cup \{clnr_i(v)\}“ \qquad (\star)$$

durchgeführt werden.

Definiert man für alle a aus dem Bild $\alpha^{(i)}(E_j)$ der für f_i schon vorgegebenen Zerlegungsfunktionen $(\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)})$

$$CLNR_{aj}^{(i)} = \{k \mid K_k^{(i)} \subseteq E_j \text{ und } \alpha^{(i)}(K_k^{(i)}) = a\},$$

so läßt sich $(\star)$ ersetzen durch

$$„S_{aa'}^{(i)} = S_{aa'}^{(i)} \cup CLNR_{aj}^{(i)}“$$

für alle $a \in \alpha^{(i)}(E_j)$.
($|CLNR_{aj}^{(i)}|$ gibt also an, wieviele Zeilen von $Z_i(X^{(1)})$, die einen Index aus E_j haben und durch $\alpha^{(i)}$ auf a abgebildet werden, verschieden sind.)

Schließlich ist noch festzustellen, daß für $1 \leq j_1, j_2 \leq l$, $j_1 \neq j_2$, $a_1 \in \alpha^{(i)}(E_{j_1})$, $a_2 \in \alpha^{(i)}(E_{j_2})$ gilt:

$$CLNR_{a_1 j_1}^{(i)} \cap CLNR_{a_2 j_2}^{(i)} = \emptyset.$$

Dies ergibt sich direkt aus den Tatsachen, daß

$$E_{j_1} = \bigcup_{K_j^{(i)} \subseteq E_{j_1}} K_j^{(i)},$$

$$E_{j_2} = \bigcup_{K_j^{(i)} \subseteq E_{j_2}} K_j^{(i)}$$

und daß $\{0,1\}^p/_\sim$ eine Partition ist.
Also sind im Algorithmus die Mengenvereinigungen

$$S_{aa'}^{(i)} = S_{aa'}^{(i)} \cup CLNR_{aj}^{(i)}$$

immer *disjunkte* Vereinigungen. Da man sich beim Test, ob die Bedingung aus Lemma 3.11 verletzt ist, ohnehin nur für die Mächtigkeiten der Mengen $S_{aa'}^{(i)}$ interessiert, genügt es, die Mächtigkeit von $CLNR_{aj}^{(i)}$ zur Mächtigkeit von $S_{aa'}^{(i)}$ zu addieren. Man kann also die Verwaltung von Mengen $S_{aa'}^{(i)}$ durch die Verwaltung von natürlichen Zahlen ersetzen. Mit den Bezeichnungen $MS_{aa'}^{(i)}$ für $|S_{aa'}^{(i)}|$ und $CLANZ_{aj}^{(i)}$ für $|CLNR_{aj}^{(i)}|$ erhält man also aus der ursprünglichen Version folgenden modifizierten branch–and–bound–Algorithmus:

Eingabe:
- Funktionen $f_1, \ldots, f_m$ aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$.
- Für $1 \le i \le m$: Zerlegungsmatrizen $Z_i(X^{(1)})$ von f_i hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$ und $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$.
- Für $1 \le i \le m$: Zerlegungsfunktionen $\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)} \in B_p$ $(k_i < r_i)$.
- Für $1 \le i \le m$: Äquivalenzklasseneinteilungen
  $\{0,1\}^p/_{\equiv_i} = \{K_1^{(i)}, \ldots, K_{vz(X^{(1)},f_i)}^{(i)}\}$ und $\{0,1\}^p/_\sim = \{E_1, \ldots, E_l\}$.
- Für $1 \le i \le m$, $1 \le j \le l$, $a \in \alpha^{(i)}(E_j)$:
  $CLNR_{aj}^{(i)} = \{k \mid K_k^{(i)} \subseteq E_j \text{ und } \alpha^{(i)}(K_k^{(i)}) = a\}$,
  $CLANZ_{aj}^{(i)} = |CLNR_{aj}^{(i)}|$.
- Natürliche Zahl h mit $h \le r_i - k_i$ für alle $1 \le i \le m$. (h gibt die Anzahl gemeinsamer, *strikter* Zerlegungsfunktionen an, nach denen gesucht wird.)

Ausgabe:
- $\alpha_1, \ldots, \alpha_h \in B_p$, so daß es einseitige Zerlegungen von $f_1, \ldots, f_m$ hinsichtlich $X^{(1)}$ gibt der Form

$$\begin{aligned}
f_1(x_1, \ldots, x_n) &= g^{(1)}(\alpha_1^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_1}^{(1)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_1+h+1}^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\
&\vdots \\
f_m(x_1, \ldots, x_n) &= g^{(m)}(\alpha_1^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{k_m}^{(m)}(\mathbf{x^{(1)}}), \alpha_1(\mathbf{x^{(1)}}), \ldots, \alpha_h(\mathbf{x^{(1)}}), \\
&\qquad \alpha_{k_m+h+1}^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_m}^{(m)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}),
\end{aligned}$$

  wobei $\alpha_1 \ldots, \alpha_h$ strikt sind. (Falls es überhaupt h Funktionen aus B_p mit dieser Eigenschaft gibt.)

- Sonst: Eine Meldung daß es keine h Funktionen aus B_p mit der angegebenen Eigenschaft gibt.

Algorithmus: Siehe Abbildung 3.21.

Häufig ergeben sich bei der Vorberechnung von $\{0,1\}^p/_\sim$ schon relativ große Äquivalenzklassen von Elementen aus $\{0,1\}^p$, denen durch α gleiche Funktionswerte zugeordnet werden müssen, so daß sich die Laufzeit des modifizierten branch–and–bound–Verfahrens stark verringert.

Wendet man das modifizierte Verfahren auf Beispiel 3.14 an (das auch zur Illustration des ursprünglichen branch–and–bound–Algorithmus diente), so erkennt man, daß nun das Problem wesentlich einfacher wird:

Beispiel 3.16 Durch $\equiv_1$ werden die Äquivalenzklassen $K_1^{(1)} = \{(000),(001),(110)\}$, $K_2^{(1)} = \{(010),(100),(101)\}$, $K_3^{(1)} = \{(111)\}$ und $K_4^{(1)} = \{(011)\}$ erzeugt und durch $\equiv_2$ die Äquivalenzklassen $K_1^{(2)} = \{(000)\}$, $K_2^{(2)} = \{(001),(110),(111)\}$, $K_3^{(2)} = \{(100)\}$ und $K_4^{(2)} = \{(010),(011),(101)\}$.
Somit ergibt sich $\{0,1\}^3/_\sim = \{E_1, E_2\}$ mit $E_1 = \{\{(000),(001),(110),(111)\}$ und $E_2 = \{(010),(011),(100),(101)\}\}$.
Es sind keine Zerlegungsfunktionen vorgegeben und es wird eine gemeinsame Zerlegungsfunktion von f_1 und f_2 gesucht.
Man hat also $CLNR_1^{(1)} = \{1,3\}$, $CLNR_2^{(1)} = \{2,4\}$, $CLNR_1^{(2)} = \{1,2\}$ und $CLNR_2^{(2)} = \{3,4\}$ und damit $CLANZ_1^{(1)} = CLANZ_2^{(1)} = CLANZ_1^{(2)} = CLANZ_2^{(2)} = 2$.
Nach der Belegung $\alpha(E_1) = 0$ ergibt sich $MS_0^{(1)} = MS_0^{(2)} = 2$ und $MS_1^{(1)} = MS_1^{(2)} = 0$.
Die Belegung $\alpha(E_2) = 0$ führt dann zu einem Widerspruch zur Bedingung aus Lemma 3.11 (α wäre dann konstant) und man muß $\alpha(E_2) = 1$ setzen. Dies führt zu $MS_0^{(1)} = MS_0^{(2)} = 2$ und $MS_1^{(1)} = MS_1^{(2)} = 2$ und man hat eine gemeinsame, strikte Zerlegungsfunktion von f_1 und f_2 gefunden.

Sucht man beispielsweise eine gemeinsame, strikte Zerlegungsfunktion nicht von f_1 und f_2, sondern von f_1 und f_3 aus Beispiel 3.11 von Seite 140 ($f_1(x_1,\ldots,x_4) = \overline{x_4}x_2x_3 + x_4(x_1 \oplus x_2)$, $f_3(x_1,\ldots,x_4) = \overline{x_4}(x_2+x_3) + x_1x_4$), so sieht man direkt durch Berechnung von $\{0,1\}^3/_\sim$, daß es in einer kommunikationsminimalen Zerlegung von f_1 und f_3 keine solche Zerlegungsfunktion geben kann:
Wegen $K_1^{(3)} = \{(000)\}$, $K_2^{(3)} = \{(001),(010),(011)\}$, $K_3^{(3)} = \{(100)\}$ und $K_4^{(3)} = \{(101),(110),(111)\}$ ergibt sich $\{0,1\}^3/_\sim = \{\{0,1\}^3\}$. Eine gemeinsame, strikte Zerlegungsfunktion von f_1 und f_3 müßte folglich allen Elementen aus $\{0,1\}^3$ den gleichen Wert zuordnen, also konstant sein. Konstante Funktionen können in kommunikationsminimalen Zerlegungen allerdings nie als Zerlegungsfunktionen vorkommen.

Lösung von CDF ausgehend von ROBDD–Darstellungen In den vorangegangenen Abschnitten wurde vorausgesetzt, daß die zu zerlegenden Booleschen Funktionen durch

Zunächst wird eine „komprimierte Version“ β des gesuchten α auf den Äquivalenzklassen aus $\{0,1\}^p/_\sim$ konstruiert. Man erhält dann α indem man für alle $1 \leq j \leq l$ definiert:

$$\alpha(E_j) = a' \iff \beta(j) = a'.$$

if $(l = 1)$
  then
    return „Keine Lösung für α.“
fi
Für alle $1 \leq i \leq m, a \in \{0,1\}^{k_i}, a' \in \{0,1\}^h : MS^{(i)}_{aa'} = 0$
$\forall 1 \leq j \leq l : \beta(j) := undef.$
$\beta(1) := (0,\ldots,0)$
$\forall 1 \leq i \leq m, \forall a \in \alpha^{(i)}(E_1):\ MS^{(i)}_{a(0,\ldots,0)} = MS^{(i)}_{a(0,\ldots,0)} + CLANZ^{(i)}_{a1}$
if $(\exists 1 \leq i \leq m, a \in \alpha^{(i)}(E_1)$ mit $MS^{(i)}_{a(0,\ldots,0)} > 2^{r_i-k_i-h})$
  then
    return „Keine Lösung für α.“
fi
$j = 2$
$a' = (0,\ldots,0) \in \{0,1\}^h$
while $((j > 1)$ **and** $(j \leq l))$ **do**
    $\beta(j) = a'$
    if $(\forall 1 \leq i \leq m, \forall a \in \alpha^{(i)}(E_j):\ MS^{(i)}_{aa'} + CLANZ^{(i)}_{aj} \leq 2^{r_i-k_i-h})$
      then
        $\forall 1 \leq i \leq m, \forall a \in \alpha^{(i)}(E_j):\ MS^{(i)}_{aa'} = MS^{(i)}_{aa'} + CLANZ^{(i)}_{aj}$
        $j = j + 1$
        $a' = (0,\ldots,0) \in \{0,1\}^h$
      else
        while $(\beta(j) = (1,\ldots,1))$ **do**
          $\beta(j) = undef.$
          $j = j - 1$
          $a' = \beta(j)$
          $\forall 1 \leq i \leq m, \forall a \in \alpha^{(i)}(E_j):\ MS^{(i)}_{aa'} = MS^{(i)}_{aa'} - CLANZ^{(i)}_{aj}$
        od
        Erhöhe a' um 1.
        $\beta(j) = undef.$
    fi
od
if $(j = 1)$
  /* Man ist am Ausgangspunkt angekommen ist, ohne eine geeignete
    Funktion β gefunden zu haben */
  then
    return „Keine Lösung für α.“
fi
$\forall 1 \leq j \leq l, \forall v \in E_j:\ \alpha(v) = \beta(j)$
return α

Abbildung 3.21: Modifizierter Algorithmus zur Lösung von CDF bei Beschränkung auf *strikte* Zerlegungsfunktionen.

Funktionstabellen bzw. Zerlegungsmatrizen gegeben sind. Hier soll nun gezeigt werden, wie man die Suche nach gemeinsamen, strikten Zerlegungsfunktionen durchführen kann, wenn die zugrundeliegenden Booleschen Funktionen durch ROBDDs repräsentiert sind.

Wie schon mehrfach erwähnt wurde, ist dieser Schritt für die Anwendung der Algorithmen auf praktische Beispiele von großer Bedeutung. Es gelingt nachzuweisen, daß man unter Voraussetzung einer kompakten ROBDD–Beschreibung der zu zerlegenden Funktionen das gesamte Verfahren auf der Basis kompakter ROBDD–Beschreibungen durchführen kann.

Durch das modifizierte branch–and–bound–Verfahren werden die Funktionswerte der gemeinsamen Zerlegungsfunktionen $\alpha_1, \dots, \alpha_h$ der Funktionen $f_1, \dots, f_m$ auf den Elementen $E_1, \dots, E_l$ von $\{0,1\}^p/_\sim$ berechnet und damit auch die Funktionswerte auf $K_1^{(i)}, \dots,$ $K_{vz(X^{(1)},f_i)}^{(i)}$ $(1 \le i \le m)$, da eine Klasse E_j $(1 \le j \le l)$ eine Vereinigung von Klassen aus $\{0,1\}^p/_{\equiv_i}$ ist. Schon in Abschnitt 3.1.3 wurde angegeben, wie aus Codierungen der Klassen $K_1^{(i)}, \dots, K_{vz(X^{(1)},f_i)}^{(i)}$ ROBDDs für die Zerlegungs– und Zusammensetzungsfunktionen in der Zerlegung von f_i bestimmt werden. Es bleibt also lediglich zu zeigen, wie man in effizienter Weise aus den ROBDDs für f_1 bis f_m die benötigten Eingaben für den modifizierten branch–and–bound–Algorithmus erzeugen kann.

Es handelt sich also im wesentlichen um das Problem, Repräsentationen für die Klassen $K_1^{(i)}, \dots, K_{vz(X^{(1)},f_i)}^{(i)}$ $(1 \le i \le m)$ und die Klassen $E_1, \dots, E_l$ zu erzeugen. Diese Klassen werden durch ROBDDs für ihre charakteristischen Funktionen repräsentiert.

ROBDDs $bdd_j^{(i)}$ für die charakteristischen Funktionen der Klassen $K_j^{(i)} \subseteq \{0,1\}^p$ $(1 \le j \le vz(X^{(1)}, f_i))$ lassen sich leicht aus dem ROBDD zu f_i bestimmen. Dabei wird ausgenutzt, daß jede Klasse $K_j^{(i)}$ eineindeutig einem „Verbindungsknoten" n_j im ROBDD zu f_i (bei Schnitt nach den Variablen aus $X^{(1)} = \{x_1, \dots, x_p\}$) entspricht. Die Menge aller $(\epsilon_1, \dots, \epsilon_p) \in \{0,1\}^p$, durch die der Verbindungsknoten n_j erreichbar ist, ist gerade die Menge $K_j^{(i)}$. Man erhält dann (wie in Abbildung 3.22 dargestellt) einen (evtl. noch nicht reduzierten) OBDD für die charakteristische Funktion von $K_j^{(i)}$, indem man den Knoten n_j durch die Konstante 1 und alle anderen Verbindungsknoten durch die Konstante 0 ersetzt (vergleiche auch Abschnitt 3.1.3).

Es bleibt das Problem, die Klassen $E_1, \dots, E_l$ von $\{0,1\}^p/_\sim$ zu bestimmen. Ausgehend von Funktionstabellen wurden diese Klassen durch Bestimmung von Zusammenhangskomponenten in einem Graphen berechnet, dessen Knoten gerade die Elemente von $\{0,1\}^p$ waren. Auch ausgehend von ROBDDs werden $E_1, \dots, E_l$ berechnet durch Bestimmung von Zusammenhangskomponenten eines implizit gegebenen ungerichteten Graphen $G_\sim = (V, E)$: Die Menge V der Knoten des Graphen ist gegeben durch die Menge aller ROBDDs $bdd_j^{(i)}$ $(1 \le i \le m, 1 \le j \le vz(X^{(1)}, f_i))$, die die Äquivalenzklassen $K_j^{(i)}$ repräsentieren. Es gibt genau dann eine Kante $\{bdd_{j_1}^{(i_1)}, bdd_{j_2}^{(i_2)}\}$, wenn $bdd_{j_1}^{(i_1)} \wedge bdd_{j_2}^{(i_2)} \neq 0$, d.h. wenn $K_{j_1}^{(i_1)} \cap K_{j_2}^{(i_2)} \neq \emptyset$ (siehe Abbildung 3.23).

Offensichtlich erhält man zu einer Klasse $K_j^{(i)}$ die Äquivalenzklasse von $\{0,1\}^p/_\sim$, die $K_j^{(i)}$ enthält, indem man in $G_\sim$ die Zusammenhangskomponente bestimmt, die $bdd_j^{(i)}$ enthält.

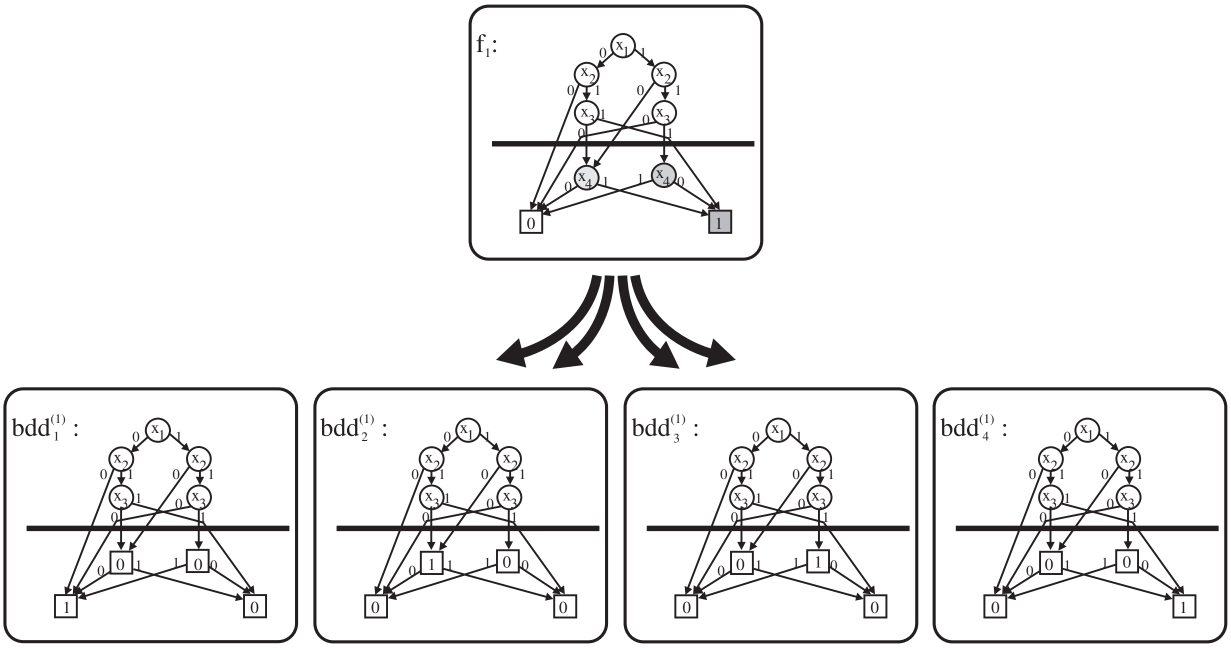


Abbildung 3.22: Bestimmung von OBDDs zu den Äquivalenzklassen $K_1^{(1)}$, $K_2^{(1)}$, $K_3^{(1)}$ und $K_4^{(1)}$ der Funktion f_1 aus Beispiel 3.11 bei Zerlegung hinsichtlich $\{x_1, x_2, x_3\}$. (Die OBDDs sind noch nicht reduziert.)

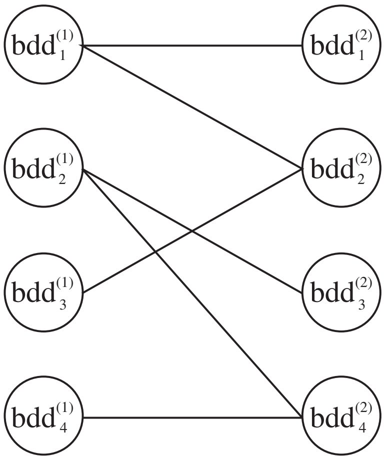


Abbildung 3.23: Berechnung der Äquivalenzklassen aus $\{0,1\}^p/_\sim$ für f_1 und f_2 aus Beispiel 3.14. Die Zusammenhangskomponenten des Graphen $G_\sim$ repräsentieren die Äquivalenzklassen E_i.

Sind für ein beliebiges $k \in \{1, \ldots, m\}$ genau die Knoten $bdd_{j_1}^{(k)}, \ldots, bdd_{j_q}^{(k)}$ in dieser Zusammenhangskomponente enthalten, so ist die gesuchte Äquivalenzklasse E_t von $\{0,1\}^p/_\sim$ gegeben durch $\bigcup_{s=1}^q K_{j_s}^{(k)}$. Die charakteristische Funktion zu E_t wird repräsentiert durch den ROBDD zu $\bigvee_{s=1}^q bdd_{j_s}^{(k)}$.

Man sieht, daß auch hier die Bestimmung der Äquivalenzklassen aus $\{0,1\}^p/_\sim$ reduziert werden kann auf die Suche nach Zusammenhangskomponenten in einem Graphen, nämlich in dem Graphen $G_\sim$. Wenn man bereit ist, für jedes Paar $bdd_{j_1}^{(i_1)}, bdd_{j_2}^{(i_2)}$ von Knoten aus der Knotenmenge von $G_\sim$ zu testen, ob $bdd_{j_1}^{(i_1)} \wedge bdd_{j_2}^{(i_2)} \neq 0$, so ist das Problem gelöst. Es stellt sich aber heraus, daß es nicht nötig ist, $G_\sim$ explizit aufzubauen und für alle solche Paare die genannte Operation durchzuführen. Man kann mit einer Anzahl von apply–Operationen (*and*, *or*, ... von 2 ROBDDs [Bry86]) auskommen, die linear ist in der Anzahl der *tatsächlich in $G_\sim$ vorhandenen Kanten.*

Abbildung 3.24 zeigt einen Algorithmus, der dies leistet. Ausgabe des Algorithmus sind Mengen $CLNR_1^{(i)}, \ldots, CLNR_l^{(i)}$ (für alle $1 \leq i \leq m$), so daß für die Äquivalenzklassen $E_1, \ldots, E_l$ von $\{0,1\}^p/_\sim$ gilt: Für $1 \leq j \leq l$ ist $E_j = \bigcup_{s \in CLNR_j^{(i)}} K_s^{(i)}$ für beliebige $i \in \{1, \ldots, m\}$.

Der Algorithmus führt eine Tiefensuche auf dem implizit gegebenen Graphen $G_\sim$ durch. Die Idee, die zu einer Reduzierung der apply–Operationen gegenüber dem naiven Ansatz führt, besteht darin, daß man zu jedem Zeitpunkt im Verlauf des Algorithmus, wenn man sich bei einem Knoten $bdd_j^{(i)}$ befindet, sämtliche Knoten bestimmen kann, die in $G_\sim$ mit $bdd_j^{(i)}$ verbunden sind und noch nicht besucht wurden, ohne daß man $G_\sim$ explizit kennen muß.

Folgende Eigenschaften von ROBDDs gehen hierbei ein: Es ist leicht, zu testen, ob der Schnitt zweier ROBDDs nicht leer ist und es ist leicht, zu einem ROBDD f ein Element aus $ON(f)$ zu bestimmen [Bry86].

Besucht man während der Tiefensuche ausgehend von $bdd_j^{(i)}$ vb mit $bdd_j^{(i)}$ verbundene, noch unbesuchte Knoten, so genügen $vb + m - 1$ apply–Operationen, um diese Knoten zu identifizieren (m ist die Anzahl der Funktionen $f_1, \ldots, f_m$).

Dies wird ermöglicht, indem man bei der Bearbeitung einer Zusammenhangskomponente ROBDDs $cc^{(1)}, \ldots, cc^{(m)}$ verwaltet, wobei $cc^{(i)}$ die zu f_i gehörigen Knoten $bdd_j^{(i)}$ repräsentiert, die zu der aktuellen Zusammenhangskomponente gehören und schon besucht wurden.

Wurden für $1 \leq i \leq m$ in der aktuellen Zusammenhangskomponente schon die Knoten $bdd_{j_1}^{(i)}, \ldots, bdd_{j_q}^{(i)}$ besucht, so gilt $cc^{(i)} = \bigvee_{s=1}^q bdd_{j_s}^{(i)}$ (Invariante aus den Zeilen 6 und 7 des Algorithmus), so daß $cc^{(i)}$ die charakteristische Funktion von $\bigcup_{s=1}^q K_{j_s}^{(i)}$ darstellt.

Vor der Bearbeitung einer neuen Zusammenhangskomponente werden folglich $cc^{(i)}$ mit 0 initialisiert (Zeile 31 des Algorithmus) und beim Besuchen eines Knotens $bdd_j^{(i)}$ wird die Operation $cc^{(i)} = cc^{(i)} \vee bdd_j^{(i)}$ durchgeführt (Zeile 4).

In der Schleife von Zeile 8–19 werden nacheinander für alle $k \in \{1, \ldots, m\} \setminus \{i\}$ alle Knoten $bdd_{j_s}^{(k)}$ besucht, die mit $bdd_j^{(i)}$ in $G_\sim$ verbunden sind und noch unbesucht sind.

Sind in Zeile 11 des Algorithmus in der aktuellen Zusammenhangskomponente schon die

Eingabe:

- Funktionen $f_1, \ldots, f_m$ aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$.
- Für $1 \leq i \leq m$:
  Äquivalenzklasseneinteilungen $\{0,1\}^p/_{\equiv_i} = \{K_1^{(i)}, \ldots, K_{vz(X^{(1)}, f_i)}^{(i)}\}$ bei einer Zerlegung von f_i hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$.
  Die Mengen $K_j^{(i)}$ sind gegeben durch ROBDDs $bdd_j^{(i)}$ für ihre charakteristischen Funktionen.

Ausgabe: Für alle $1 \leq i \leq m$ Mengen $CLNR_1^{(i)}, \ldots, CLNR_l^{(i)}$, so daß für die Äquivalenzklassen $E_1, \ldots, E_l$ von $\{0,1\}^p/_\sim$ gilt:
$\forall 1 \leq j \leq l \; E_j = \bigcup_{s \in CLNR_j^{(i)}} K_s^{(i)}$ für beliebige $i \in \{1, \ldots, m\}$.

Algorithmus:

```
procedure search(bdd bdd_j^(i), int i)
begin
  if (i = 1) then Markiere bdd_j^(i) als besucht. fi
  cc^(i) = cc^(i) ∨ bdd_j^(i)
  CLNR^(i)_zhkanz = CLNR^(i)_zhkanz ∪ {j}
  /* Falls Knoten bdd^(i)_j1, ..., bdd^(i)_jq schon besucht, dann gilt
     cc^(i) = ⋁_{s=1}^{q} bdd^(i)_js und CLNR^(i)_zhkanz = {j_1, ..., j_q} */
  for k = 1 to m do
    if (k ≠ i)
      then
          nicht_überdeckt = bdd_j^(i) ∧ ¬cc^(k)
          while (nicht_überdeckt ≠ 0) do
                Sei v ∈ ON(nicht_überdeckt) beliebig.
                Sei u ∈ {1, ..., vz(f_k, X^(1))}, so daß v ∈ ON(bdd_u^(k)).
                search(bdd_u^(k), k)
                nicht_überdeckt = bdd_j^(i) ∧ ¬cc^(k)
          od
    fi
  od
end

begin
  ∀1 ≤ j ≤ vz(f_1, X^(1)) :  Markiere bdd_j^(1) als unbesucht.
  zhkanz = 0
  for j = 1 to vz(f_1, X^(1)) do
    if (bdd_j^(1) noch nicht besucht)
      then
          zhkanz = zhkanz + 1
          ∀1 ≤ i ≤ m : cc^(i) = 0, CLNR^(i)_zhkanz = ∅
          search(bdd_j^(1), 1)
    fi
  od
end
```

Abbildung 3.24: Algorithmus zur Bestimmung von $\{0,1\}^p/_\sim$.

Knoten $bdd_{j'}^{(k)}$, $j' \in J$, von f_k besucht, so gilt $\overline{cc^{(k)}} = \bigvee_{j' \in \{1,\ldots,vz(f_k,X^{(1)})\} \setminus J} bdd_{j'}^{(k)}$. Ein Knoten $bdd_{j'}^{(k)}$ ist in $G_\sim$ aber genau dann mit dem aktuellen Knoten $bdd_j^{(i)}$ verbunden, wenn $bdd_{j'}^{(k)} \wedge bdd_j^{(i)} \neq 0$. Folglich wird mit dem Test $\overline{cc^{(k)}} \wedge bdd_j^{(i)} \neq 0$ in den Zeilen 11 und 12 *parallel* für alle noch nicht besuchten Knoten $bdd_{j'}^{(k)}$ mit $j' \in \{1,\ldots,vz(f_k,X^{(1)})\} \setminus J$ getestet, ob es in $G_\sim$ eine Kante vom aktuellen Knoten $bdd_j^{(i)}$ zu $bdd_{j'}^{(k)}$ gibt. Ist dies der Fall, so wird ein beliebiger unbesuchter Knoten $bdd_u^{(k)}$ ausgesucht (Zeilen 13 und 14) und die Tiefensuche wird mit diesem Knoten fortgesetzt (Zeile 15). Nach Ende der rekursiven Bearbeitung wird wiederum getestet, ob es einen weiteren unbesuchten Knoten $bdd_u^{(k)}$ gibt (Zeilen 16 und 12). Dies geschieht so lange, bis es in der **while**–Schleife keinen solchen Knoten mehr gibt ($nicht\_überdeckt = 0$).
Man benötigt also für jede Kante im (impliziten) Graphen $G_\sim$, die die Tiefensuche zu einem unbesuchten Knoten verfolgt, eine *and*–Operation auf 2 ROBDDs (Zeile 11 bzw. 16) und für jeden Knoten $bdd_j^{(i)}$ und jedes $k \in \{1,\ldots,m\} \setminus \{i\}$ eine zusätzliche *and*–Operation (Zeile 11 bzw. 16), deren Ergebnis 0 ist und also nicht zum Verfolgen einer Kante führt, so daß man insgesamt zu $vb + m - 1$ *and*–Operationen pro Knoten $bdd_j^{(i)}$ kommt, wenn vb die Anzahl der Kanten ist, die im Verlauf der Tiefensuche von $bdd_j^{(i)}$ aus zu noch unbesuchten Knoten führen.
Da die Kanten von $G_\sim = (V, E)$, die bei der Tiefensuche verfolgt werden für jede Zusammenhangskomponente aus einem aufspannenden Baum sind, ist die Anzahl der *and*–Operationen begrenzt durch $|V| - 1 + |V| \times (m-1)$. Berücksichtigt man noch pro Knoten die *or*–Operation aus Zeile 4, so erhält man insgesamt höchstens

$$|V| \times (m+1) - 1$$

apply–Operationen auf 2 ROBDDs.
Berücksichtigt man noch, daß der Grad eines Knotens in $G_\sim$ aber auf jeden Fall größer oder gleich $m-1$ ist, so ist die Gesamtzahl der apply–Operationen linear in der Größe von $G_\sim$.

Man kann also tatsächlich mit einer Anzahl von apply–Operationen auskommen, die linear ist in der Anzahl der in $G_\sim$ vorhandenen Kanten und das folgende Lemma ist somit bewiesen:

Lemma 3.12 *Die Anzahl der apply–Operationen, die in Algorithmus search aus Abbildung 3.24 vorkommen ist beschränkt durch $|V| \times (m+1) - 1$ und ist somit linear in der Größe des Graphen $G_\sim$.*

Es ist klar, daß man aus den Mengen $CLNR_1^{(i)}, \ldots, CLNR_l^{(i)}$, die der Algorithmus in Abbildung 3.24 ausgibt, und aus den (evtl.) schon vorgegebenen Zerlegungsfunktionen $\alpha_1^{(i)}, \ldots, \alpha_{k_i}^{(i)}$ die für den modifizierten branch–and–bound–Algorithmus benötigten Mengen $CLNR_{aj}^{(i)} = \{k \mid K_k^{(i)} \subseteq E_j$ und $\alpha^{(i)}(K_k^{(i)}) = a\}$ (für $1 \leq i \leq m$, $1 \leq j \leq l$, $a \in \alpha^{(i)}(E_j)$) bzw. deren Mächtigkeiten $CLANZ_{aj}^{(i)} = |CLNR_{aj}^{(i)}|$ leicht bestimmen kann.

Eine weitere wichtige Beobachtung besteht darin, daß ausgehend von kompakten Beschreibungen der zu zerlegenden Funktionen die im Verlauf des Algorithmus auftretenden ROBDD–Größen nicht „explodieren“ können. Dies sieht man leicht ein, wenn man bedenkt, daß die im Verlauf des Verfahrens auftretenden ROBDDs $cc^{(i)}$ Disjunktionen von ROBDDs $bdd_j^{(i)}$ $(1 \leq j \leq vz(f_i, X^{(1)}))$ sind und somit (in diesem speziellen Fall) gewonnen werden können durch Ersetzung von „Verbindungsknoten“ im ROBDD zu f_i durch Konstanten 1 und 0 und anschließendes Reduzieren des resultierenden OBDD.

3.4.2.2 Anwendung der Lösung von CDF

Dieser Teilabschnitt hat zur Aufgabe, zu klären, wie die gerade beschriebene Lösung des Problems CDF in ein Gesamtverfahren zur Bestimmung von gemeinsamen Zerlegungsfunktionen möglichst vieler Ausgangsfunktionen eingebunden werden kann.

Bisher wurde beschrieben, wie man h gemeinsame, strikte Zerlegungsfunktionen für die Zerlegung der Funktionen $f_1, \dots, f_m$ bestimmen kann. Mit einer Binärsuche bzgl. h kann man dann eine maximale Anzahl h_{max} von gemeinsamen, strikten Zerlegungsfunktionen von $f_1, \dots, f_m$ finden ($1 \leq h \leq \min_{1 \leq i \leq m} r_i - k_i$, wobei r_i die Gesamtzahl der Zerlegungsfunktionen in der Zerlegung von f_i ist, k_i die Anzahl der schon vorgegebenen Zerlegungsfunktionen in der Zerlegung von f_i).

Sind h_{max} gemeinsame Zerlegungsfunktionen von $f_1, \dots, f_m$ bestimmt, so kann es keine weiteren gemeinsamen Zerlegungsfunktionen von $f_1, \dots, f_m$ geben, die man zusätzlich in einer kommunikationsminimalen Zerlegung verwenden kann. Es kann aber trotzdem (unter Voraussetzung der Verwendung der h_{max} gefundenen gemeinsamen Zerlegungsfunktionen für $f_1, \dots, f_m$) weiterhin Zerlegungsfunktionen geben, die man in der Zerlegung von (echten) Teilmengen von $\{f_1, \dots, f_m\}$ gemeinsam verwenden kann.

Beispiel 3.11 (Fortsetzung):
In Beispiel 3.11 (Seite 140) gibt es keine gemeinsame Zerlegungsfunktion von f_1, f_2 und f_3 ($h_{max} = 0$). Allerdings gibt es eine gemeinsame Zerlegungsfunktion von f_1 und f_2 und eine gemeinsame Zerlegungsfunktion von f_2 und f_3.

Es würde sich also anbieten (solange noch Zerlegungsfunktionen zu bestimmen sind), für Teilmengen von $\{f_1, \dots, f_m\}$ — beginnend mit den größten — gemeinsame Zerlegungsfunktionen zu suchen (unter Voraussetzung der Verwendung von bereits gefundenen gemeinsamen Zerlegungsfunktionen). Hierbei würde man dann allerdings in vielen Fällen unnötig viele verschiedene Teilmengen untersuchen, z.B. auch solche, die Funktionen f_{i_1} und f_{i_2} enthalten, die von vorneherein in kommunikationsminimalen Zerlegungen überhaupt keine gemeinsamen Zerlegungsfunktionen haben können.

Um dies zu verhindern, macht man von der folgenden Feststellung Gebrauch: Das Problem CDF ist zwar NP–hart, auch wenn man sich auf 2 Funktionen und auf strikte Zerlegungsfunktionen beschränkt (siehe Korollare 3.5 und 3.6), es ist aber effizient zu lösen, wenn

- man sich auf 2 Funktionen f_1 und f_2 beschränkt,
- man sich auf strikte Zerlegungsfunktionen beschränkt,
- keine Zerlegungsfunktionen $\alpha_1^{(1)}, \ldots, \alpha_{k_1}^{(1)}$ bzw. $\alpha_1^{(2)}, \ldots, \alpha_{k_2}^{(2)}$ vorgegeben sind ($k_1 = k_2 = 0$) und
- man nach genau einer gemeinsamen Zerlegungsfunktion sucht ($h = 1$).

Für diese Teilklasse von Instanzen von CDF kann man durch dynamische Programmierung effizient eine Lösung finden. Abbildung 3.25 zeigt ein dynamisches Programm zur Lösung solcher Probleme.

Eingabe:
- Funktionen f_1 und f_2 aus B_n mit den Eingangsvariablen $x_1, \ldots, x_n$.
- $r_1 = \lceil \log(vz(X^{(1)}, f_1)) \rceil > 0$, $r_2 = \lceil \log(vz(X^{(1)}, f_2)) \rceil > 0$ bei Zerlegung von f_1 und f_2 hinsichtlich $X^{(1)} = \{x_1, \ldots, x_p\}$.
- Für $i \in \{1, 2\}$:
  Äquivalenzklasseneinteilungen $\{0,1\}^p/_{\equiv_i} = \{K_1^{(i)}, \ldots, K_{vz(X^{(1)}, f_i)}^{(i)}\}$. Äquivalenzklasseneinteilung $\{0,1\}^p/_{\sim} = \{E_1, \ldots, E_l\}$.
- Für $i \in \{1, 2\}$, $1 \le j \le l$, $CLNR_j^{(i)} = \{k | K_k^{(i)} \subseteq E_j\}$, $CLANZ_j^{(i)} = |CLNR_j^{(i)}|$.

Ausgabe:
- $\alpha_1 \in B_p$, so daß es einseitige Zerlegungen von f_1 und f_2 hinsichtlich $X^{(1)}$ gibt der Form

$$\begin{aligned} f_1(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(1)}(\alpha_1(\mathbf{x^{(1)}}), \alpha_2^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\ f_2(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) &= g^{(2)}(\alpha_1(\mathbf{x^{(1)}}), \alpha_2^{(2)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_2}^{(2)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}), \end{aligned}$$

  wobei α_1 strikt ist. (Falls es überhaupt eine Funktion aus B_p mit dieser Eigenschaft gibt.)
- Sonst: Eine Meldung daß es keine Funktion aus B_p mit der angegebenen Eigenschaft gibt.

Algorithmus:
Siehe Abbildung 3.25.

Der Algorithmus beruht auf folgenden Überlegungen:

Mit Lemma 3.10 erkennt man, daß es genau dann eine Zerlegungsfunktion α_1 mit den angegebenen Eigenschaften gibt, wenn es eine Aufteilung von $\{1, \ldots, l\}$ in 2 Mengen

$$D_0 = \{j \mid \alpha_1(x) = 0 \ \forall x \in E_j\} \text{ und } D_1 = \{j \mid \alpha_1(x) = 1 \ \forall x \in E_j\}$$

```
∀0 ≤ l_1 ≤ 2^{r_1-1}, 0 ≤ l_2 ≤ 2^{r_2-1} : B[l_1][l_2] = false, D[l_1][l_2] = ∅
B[0][0] = true
for j = 1 to l do
    for l_1 = 2^{r_1-1} downto CLANZ_j^{(1)} do
        for l_2 = 2^{r_2-1} downto CLANZ_j^{(2)} do
            if (B[l_1 - CLANZ_j^{(1)}][l_2 - CLANZ_j^{(2)}] = true)
              then
                    B[l_1][l_2] = true
                    D[l_1][l_2] = D[l_1 - CLANZ_j^{(1)}][l_2 - CLANZ_j^{(2)}] ∪ {j}
                    /* Invariante: Es gibt genau dann eine Teilmenge D von {1, ..., j} mit
                       ∑_{l∈D} CLANZ_l^{(1)} = l_1 und ∑_{l∈D} CLANZ_l^{(2)} = l_2, wenn B[l_1][l_2] = true.
                       Dann ist D = D[l_1][l_2] eine mögliche Wahl für D. */
            fi
        od
    od
od
if (∃ l_1, l_2 mit
  vz(X, f_1) - 2^{r_1-1} ≤ l_1 ≤ 2^{r_1-1} und vz(X, f_2) - 2^{r_2-1} ≤ l_2 ≤ 2^{r_2-1} und B[l_1][l_2] = true)
  then
        for j = 1 to l do
            if (j ∈ D[l_1][l_2])
              then
                    ∀x ∈ E_j : α_1(x) = 0
               else
                    ∀x ∈ E_j : α_1(x) = 1
            fi
        od
  else
        Gebe aus, daß es kein solches α_1 gibt.
fi
```

Abbildung 3.25: Dynamisches Programm zur Bestimmung einer gemeinsamen Zerlegungsfunktion zweier Funktionen f_1 und f_2.

gibt mit

$$\sum_{k \in D_0} CLANZ_k^{(1)} \leq 2^{r_1 - 1} \text{ und } \sum_{k \in D_1} CLANZ_k^{(1)} \leq 2^{r_1 - 1} \text{ sowie}$$

$$\sum_{k \in D_0} CLANZ_k^{(2)} \leq 2^{r_2 - 1} \text{ und } \sum_{k \in D_1} CLANZ_k^{(2)} \leq 2^{r_2 - 1}.$$

Da

$$\sum_{k=1}^{l} CLANZ_k^{(1)} = vz(X^{(1)}, f_1),$$

ist

$$\sum_{k \in D_1} CLANZ_k^{(1)} = \sum_{k=1}^{l} CLANZ_k^{(1)} - \sum_{k \in D_0} CLANZ_k^{(1)} \leq 2^{r_1 - 1}$$

äquivalent zu der Aussage

$$\sum_{k \in D_0} CLANZ_k^{(1)} \geq vz(X^{(1)}, f_1) - 2^{r_1 - 1}.$$

Folglich gibt es genau dann eine solche Zerlegungsfunktion α_1, wenn es eine Menge

$$D_0 = \{j \mid \alpha_1(x) = 0 \; \forall x \in E_j\}$$

gibt mit

$$vz(X^{(1)}, f_1) - 2^{r_1 - 1} \leq \sum_{k \in D_0} CLANZ_k^{(1)} \leq 2^{r_1 - 1} \text{ und}$$

$$vz(X^{(1)}, f_2) - 2^{r_2 - 1} \leq \sum_{k \in D_0} CLANZ_k^{(2)} \leq 2^{r_2 - 1}.$$

Mit Hilfe des dynamischen Programms wird berechnet, ob es eine solche Menge D_0 gibt. Dazu werden 2 zweidimensionale Felder B und D benutzt. Die Einträge in B werden zunächst alle auf **false** gesetzt, die Einträge in D auf $\emptyset$ (Zeile 1). In einer Schleife für $j = 1 \ldots, l$ (Zeilen 3–16) werden B und D verändert. Wesentlich für die Korrektheit des dynamischen Programms ist hierbei die Gültigkeit der Invariante aus Zeilen 10–12 des Algorithmus: Im j–ten Schleifendurchlauf gibt es genau dann eine Teilmenge D von $\{1, \ldots, j\}$ mit $\sum_{l \in D} CLANZ_l^{(1)} = l_1$ und $\sum_{l \in D} CLANZ_l^{(2)} = l_2$, wenn $B[l_1][l_2] =$ **true**. Ein mögliche Wahl für D ist dann in $D[l_1][l_2]$ abgespeichert. Nach l Durchläufen der Schleife aus den Zeilen 3–16 gibt es genau dann eine Menge D_0 mit

$$vz(X^{(1)}, f_1) - 2^{r_1 - 1} \leq \sum_{k \in D_0} CLANZ_k^{(1)} \leq 2^{r_1 - 1} \text{ und}$$

$$vz(X^{(1)}, f_2) - 2^{r_2 - 1} \leq \sum_{k \in D_0} CLANZ_k^{(2)} \leq 2^{r_2 - 1},$$

wenn es l_1 und l_2 gibt $B[l_1][l_2] =$ **true** und

$$vz(X, f_1) - 2^{r_1 - 1} \leq l_1 \leq 2^{r_1 - 1} \text{ und}$$

$$vz(X, f_2) - 2^{r_2 - 1} \leq l_2 \leq 2^{r_2 - 1}.$$

Man kann dann $D_0 = D[l_1][l_2]$ wählen.

Die Laufzeit des dynamischen Programms beträgt

$$O(2^{r_1 - 1} \cdot 2^{r_2 - 1} \cdot l) = O(vz(X^{(1)}, f_1) \cdot vz(X^{(1)}, f_2) \cdot l).$$

Mit Hilfe des obigen dynamischen Programms kann für alle Paare von Funktionen f_{i_1} und f_{i_2} festgestellt werden, ob es eine gemeinsame, strikte Zerlegungsfunktion von f_{i_1} und f_{i_2} gibt. Aufgrund dieser Information ist man in der Lage, einen Graphen $G_{CDF} = (\{f_1, \ldots, f_m\}, E)$ aufzustellen, wobei es genau dann eine Kante zwischen 2 Knoten f_{i_1} und f_{i_2} gibt, wenn es eine gemeinsame, strikte Zerlegungsfunktion von f_{i_1} und f_{i_2} gibt. Will man das branch–and–bound–Verfahren zur Bestimmung gemeinsamer, strikter Zerlegungsfunktionen auf eine Teilmenge T der Funktionen $\{f_1, \ldots, f_m\}$ anwenden, so ist dies nur dann sinnvoll, wenn die Funktionen aus T in G_{CDF} auch eine Clique bilden.

Beispiel 3.17 Betrachte die binäre Addition zweier 2^k–Bit–Zahlen

$$add_{2^k} : \{0,1\}^{2^{k+1}} \to \{0,1\}^{2^k}, add_{2^k}(a_{2^k-1}, \ldots, a_0, b_{2^k-1}, \ldots, b_0) = (r_{2^k-1}, \ldots, r_0),$$

wobei

$$\sum_{i=0}^{2^k-1} r_i 2^i = (\sum_{i=0}^{2^k-1} a_i 2^i + \sum_{i=0}^{2^k-1} b_i 2^i) \bmod 2^{2^k}.$$

Die einzelnen Ausgangsfunktionen von add_{2^k} werden mit $s_{2^k-1}, \ldots, s_0$ bezeichnet, also $add_{2^k} = (s_{2^k-1}, \ldots, s_0)$. Wird add_{2^k} zweiseitig und gleichmächtig zerlegt, so ergibt sich bei der Bestimmung der Variablenaufteilung durch den implementierten Logiksynthesealgorithmus (siehe Kapitel 4) für alle s_i $(0 \leq i \leq 2^k - 1)$ die Aufteilung

$$A = \{\{a_{2^k-1}, \ldots, a_{2^{k-1}}, b_{2^k-1}, \ldots, b_{2^{k-1}}\}\{a_{2^{k-1}-1}, \ldots, a_0, b_{2^{k-1}-1}, \ldots, b_0\}\}.$$

Die zweiseitige Zerlegung von add_{2^k} hinsichtlich A läßt sich zurückführen auf zwei einseitige Zerlegungen hinsichtlich $A_1 = \{a_{2^k-1}, \ldots, a_{2^{k-1}}, b_{2^k-1}, \ldots, b_{2^{k-1}}\}$ und $A_0 = \{a_{2^{k-1}-1}, \ldots, a_0, b_{2^{k-1}-1}, \ldots, b_0\}$ (vergleiche Kapitel 3.3).
Bei der (kommunikationsminimalen) Zerlegung hinsichtlich A_0 ergibt sich für alle s_i genau eine Zerlegungsfunktion. Im zugehörigen Graph G_{CDF} gibt es 2^{k-1} Knoten $s_0, \ldots, s_{2^{k-1}-1}$, von denen keine Kante ausgeht und weitere 2^{k-1} Knoten $s_{2^{k-1}}, \ldots, s_{2^k-1}$, die eine Clique in G_{CDF} bilden (vergleiche Abbildung 3.26 links für add_{16}). Tatsächlich gibt es auch eine gemeinsame Zerlegungsfunktion von $s_{2^{k-1}}, \ldots, s_{2^k-1}$, nämlich die Carry–Funktion der Addition von $(a_{2^{k-1}-1}, \ldots, a_0)$ und $(b_{2^{k-1}-1}, \ldots, b_0)$ (oder die Negation der Carryfunktion).
In einer (kommunikationsminimalen) Zerlegung hinsichtlich A_1 gibt es 0 Zerlegungsfunktionen für $s_0, \ldots, s_{2^{k-1}-1}$ (diese Funktionen sind unabhängig von $a_{2^k-1}, \ldots, a_{2^{k-1}}, b_{2^k-1}, \ldots, b_{2^{k-1}}$), 1 Zerlegungsfunktion für $s_{2^{k-1}}$ und jeweils 2 Zerlegungsfunktionen für $s_{2^{k-1}+1} \ldots, s_{2^k-1}$. Im Graph G_{CDF} für diese Zerlegung gibt es lediglich eine Kante von $s_{2^{k-1}}$ nach $s_{2^{k-1}+1}$ (vergleiche Abbildung 3.26 rechts für add_{16}). Es macht also nur Sinn, nach einer gemeinsamen Zerlegungsfunktion von $s_{2^{k-1}}$ und $s_{2^{k-1}+1}$ zu suchen. Andere Teilmengen von Funktionen s_i brauchen aufgrund von G_{CDF} nicht betrachtet zu werden.

Falls für 2 Funkionen f_{i_1} und f_{i_2} die Relationen $\equiv_{i_1}$ und $\equiv_{i_2}$ für die Zerlegung hinsichtlich einer Variablenteilmenge genau gleich sind, so brauchen nicht sowohl f_{i_1} als auch f_{i_2} in den entsprechenden Graph G_{CDF} aufgenommen zu werden, da man die Zerlegungsfunktionen von f_{i_1} und f_{i_2} *alle* gleich wählen kann. Auch Funktionen, die unabhängig sind von der Variablenteilmenge, hinsichtlich der zerlegt wird, und folglich 0 Zerlegungsfunktionen aufweisen, kann man natürlich aus G_{CDF} entfernen.

Beispiel 3.17 (Fortsetzung):
Bei der Funktion add_{2^k} sind bei Zerlegung hinsichtlich A_0 für alle $i \in \{2^{k-1}, \ldots, 2^k - 1\}$ die Relationen $\equiv_i$ zu s_i identisch. Folglich kann man G_{CDF} zu dem Graph aus Abbildung 3.27 links reduzieren.
In Graph G_{CDF} (in Abbildung 3.27 rechts) für die Zerlegung hinsichtlich A_1 kann man

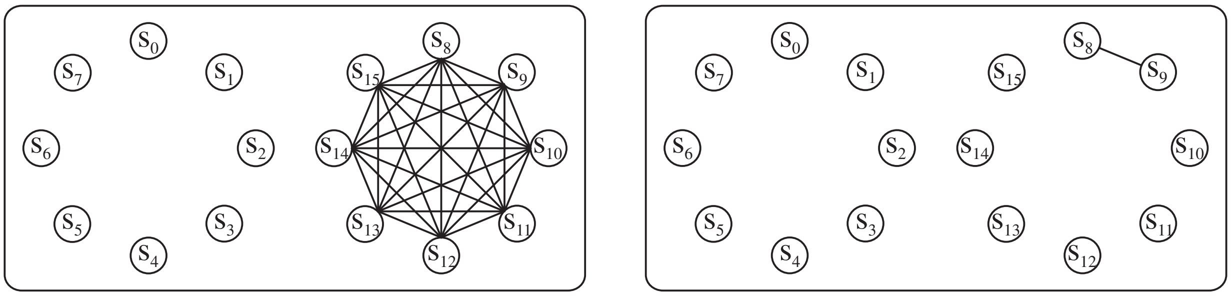


Abbildung 3.26: Graph G_{CDF} für die Funktion add_{16}.

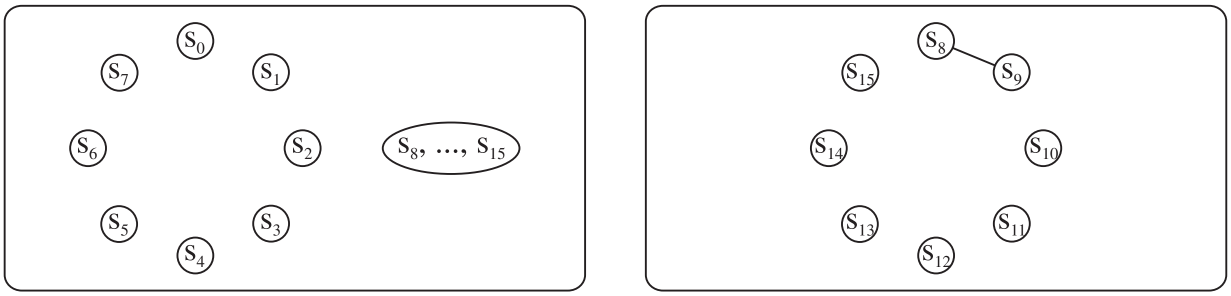


Abbildung 3.27: Reduzierter Graph G_{CDF} zu add_{16}.

die Knoten $s_0, \ldots, s_{2^{k-1}-1}$ weglassen, da die entsprechenden Funktionen bei Zerlegung hinsichtlich A_1 0 Zerlegungsfunktionen aufweisen.

Zur Bestimmung gemeinsamer Zerlegungsfunktionen genügt es also Teilmengen von $\{f_1, \ldots, f_m\}$ zu betrachten, die in dem reduzierten Graph G_{CDF} eine Clique bilden. Das auf dieser Grundlage implementierte Logiksyntheseverfahren beginnt mit einer maximalen Clique in G_{CDF} und bestimmt mit Hilfe des branch–and–bound–Verfahrens die maximale Anzahl von gemeinsamen Zerlegungsfunktionen für die Ausgangsfunktionen in dieser Clique. Die gefundenen gemeinsamen Zerlegungsfunktionen werden in darauffolgenden Aufrufen des branch–and–bound–Verfahrens als schon vorgegebene Zerlegungsfunktionen vorausgesetzt. Das Verfahren bearbeitet dann in analoger Weise die Cliquen in G_{CDF} in der Reihenfolge abnehmender Größe. Sind zu einer Ausgangsfunktion im Verlauf des Verfahrens schon alle Zerlegungsfunktionen bestimmt, so wird der entsprechende Knoten aus G_{CDF} gelöscht. (In Anhang B ist ein Algorithmus zur Bestimmung der Cliquen mit abnehmender Größe angegeben.)

Beispiel 3.17 (Fortsetzung):
Bei der Funktion add_{2^k} sind bei Zerlegung hinsichtlich A_0 im *reduzierten* Graphen G_{CDF}

keine Kanten mehr vorhanden. Der branch–and–bound–Algorithmus muß (unter Voraussetzung des reduzierten Graphen) überhaupt nicht aufgerufen werden. Die Zerlegungsfunktionen werden für die einzelnen Ausgänge getrennt bestimmt. Daß die Zerlegungsfunktion für $s_{2^{k-1}}, \ldots, s_{2^k-1}$ identisch gewählt werden kann, ist schon vorher klar, da die zugehörigen Äquivalenzrelationen $\equiv_{2^{k-1}}, \ldots, \equiv_{2^k-1}$ identisch sind.
Bei Zerlegung hinsichtlich A_1 gibt es genau eine Clique, die mehr als eine Funktion beinhaltet. Für diese Clique wird eine gemeinsame Zerlegungsfunktion bestimmt, $s_{2^{k-1}}$ wird aus G_{CDF} gestrichen, da für $s_{2^{k-1}}$ auch nur eine Zerlegungsfunktion benötigt wird und danach werden für alle verbliebenen Ausgangsfunktionen einzeln die restlichen Zerlegungsfunktionen bestimmt.
Die Realisierung, die sich durch rekursive Anwendung des Verfahrens für add_{2^k} ergibt, wird in Kapitel 4 genauer untersucht.

Bemerkung 3.7 *Auch abgesehen von der oben angegebenen Teilklasse des Problems CDF treten in praktischen Beispielen häufig eine Reihe weiterer „Sonderfälle“ auf (siehe [SM93]), bei denen man die Lösung effizienter bestimmen kann als durch den angegebenen modifizierten branch–and–bound–Algorithmus.*

3.4.3 Behandlung partieller Funktionen

Schließlich stellt sich noch die Frage, wie sich die Verfahren zur Zerlegung von Funktionen mit mehreren Ausgängen und zur Behandlung von partiellen Funktionen verbinden lassen.

Zu diesem Zweck wird in diesem Teilabschnitt eine Modifikation des Verfahrens zur Lösung von EKM aus den Abschnitten 3.2.2 und 3.2.3 eingeführt.

Eine einfache Möglichkeit würde darin bestehen, bei der Zerlegung der partiellen Funktionen $f_1, \ldots, f_m$ hinsichtlich einer Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ zunächst (wie in den Abschnitten 3.2.2 und 3.2.3) für alle Funktionen f_i die don't cares so zu belegen, daß die Anzahl der benötigten Zerlegungsfunktionen bei der Zerlegung von f_i minimiert wird. Danach wird dann das Verfahren zur Berechnung gemeinsamer Zerlegungsfunktionen angewendet.

Andererseits wäre es wünschenswert, die don't cares schon im Hinblick auf eine *gemeinsame Zerlegung* von $f_1, \ldots, f_m$ zu belegen. Bei der don't care–Belegung soll berücksichtigt werden, daß man plant, $f_1, \ldots, f_m$ gemeinsam (d.h. unter Berechnung gemeinsamer Zerlegungsfunktionen) zu zerlegen.

Um dies zu erreichen, wird hier vorgeschlagen, eine untere Schranke für die *Gesamtzahl* der Zerlegungsfunktionen zu minimieren.

Zerlegt man eine totale Boolesche Funktion $f \in B_{n,m}$ *als Ganzes*, d.h. in der Form

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n) \qquad (\star)$$

mit Funktionen $\alpha \in B_{p,r}$ und $g \in B_{n-p+r,m}$, so ergibt sich auch hier die minimale Anzahl r von Zerlegungsfunktionen anhand der Zerlegungsmatrix $Z(X^{(1)})$, wobei in diesem Fall die Einträge der Zerlegungsmatrix aber aus $\{0,1\}^m$ sind. Man zeigt genau analog zum Beweis von Satz 3.1, daß die minimale Anzahl r von Zerlegungsfunktionen in einer Zerlegung vom Typ $(\star)$ gleich $\lceil \log(vz(X^{(1)}, f)) \rceil$ ist, wobei $vz(X^{(1)}, f)$ die Anzahl der verschiedenen Zeilenmuster in $Z(X^{(1)})$ ist.

Da $\lceil \log(vz(X^{(1)}, f)) \rceil$ eine untere Schranke für die Anzahl von Zerlegungsfunktionen in Zerlegungen vom Typ $(\star)$ ist, gilt auch für Zerlegungen vom Typ

$$\begin{array}{rcl} f_1(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) & = & g^{(1)}(\alpha_1^{(1)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_1}^{(1)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}) \\ & \vdots & \\ f_m(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) & = & g^{(m)}(\alpha_1^{(m)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_m}^{(m)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}}), \end{array}$$

mit $r_i = \lceil \log(vz(X^{(1)}, f_i)) \rceil$:

$$|\bigcup_{i=1}^{m} \{\alpha_1^{(i)}, \ldots, \alpha_{r_i}^{(i)}\}| \geq \lceil \log(vz(X^{(1)}, f)) \rceil.$$

$\lceil \log(vz(X^{(1)}, f)) \rceil$ stellt also nicht einfach eine untere Schranke für $\sum_{i=1}^{m} \lceil \log(vz(X^{(1)}, f_i)) \rceil$ dar, sondern liefert auch ein Maß dafür, inwieweit man damit rechnen kann, gemeinsame Zerlegungsfunktionen in der Zerlegung der Funktionen f_i zu finden[7].

Analog zur Behandlung von partiellen Funktionen mit einem Ausgang werden Erweiterungen $f'_1, \ldots, f'_m$ der partiellen Funktionen $f_1, \ldots, f_m$ gesucht, so daß die untere Schranke $vz(X^{(1)}, f')$ minimiert wird. Ähnlich wie bei Funktionen mit einem Ausgang werden auch hier die Zeilenindizes der Zerlegungsmatrix $Z(X^{(1)})$ in Klassen von „kompatiblen" Zeilen eingeteilt.

Zu diesem Zweck wird der Begriff des Kofaktors auf mehrere partielle Funktionen ausgedehnt:

Definition 3.11 *Seien für $1 \leq i \leq m$ $(m > 1)$ $f_i \in S(D_i)$, $D_i \subseteq \{0,1\}^n$. Der* Kofaktor *von $(f_1, \ldots, f_m)$ hinsichtlich $x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}$ ist eine Funktion $f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}} : \{0,1\}^{n-p} \to \{0,1,\star\}^m$, wobei für alle $(y_1, \ldots, y_{n-p}) \in \{0,1\}^{n-p}$ gilt:*

$$f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}(y_1, \ldots, y_{n-p}) = (\delta_1, \ldots, \delta_m)$$

mit

$$\delta_i = \begin{cases} \star, & falls\ (\epsilon_1, \ldots, \epsilon_p, y_1, \ldots, y_{n-p}) \notin D_i \\ f_i(\epsilon_1, \ldots, \epsilon_p, y_1, \ldots, y_{n-p}), & falls\ (\epsilon_1, \ldots, \epsilon_p, y_1, \ldots, y_{n-p}) \in D_i \end{cases}$$

[7] Aussagekraft hat das Maß $vz(X^{(1)}, f)$ allerdings nur, wenn es sich genügend stark von 2^p $(p = |X^{(1)}|)$ unterscheidet.

Bemerkung 3.8 *Setzt man für den Kofaktor* $f_{x_1^{\epsilon_1}\dots x_p^{\epsilon_p}}$ *von* $(f_1,\dots,f_m)$

$$D_i' = \{(y_1,\dots,y_{n-p}) \mid f_{x_1^{\epsilon_1}\dots x_p^{\epsilon_p}}(y_1,\dots,y_{n-p}) = (\delta_1,\dots,\delta_m) \text{ mit } \delta_i \neq \star\}$$

und $h : D_i' \to \{0,1\}$, *so daß für alle* $(y_1,\dots,y_{n-p}) \in D_i'$

$$h(y_1,\dots,y_{n-p}) = \delta_i \text{ mit } f_{x_1^{\epsilon_1}\dots x_p^{\epsilon_p}}(y_1,\dots,y_{n-p}) = (\delta_1,\dots,\delta_m),$$

so ist h *der Kofaktor von* f_i *hinsichtlich* $x_1^{\epsilon_1}\dots x_p^{\epsilon_p}$.

Zwei Elemente $\epsilon^{(1)}$ und $\epsilon^{(2)}$ aus $\{0,1\}^p$ heißen dann kompatibel, wenn es kein $(y_1,\dots,y_{n-p})$ $\in \{0,1\}^{n-p}$ gibt mit $f_{x_1^{\epsilon_1^{(1)}}\dots x_p^{\epsilon_p^{(1)}}}(y_1,\dots,y_{n-p}) = (\delta_1^{(1)},\dots,\delta_m^{(1)})$, $f_{x_1^{\epsilon_1^{(2)}}\dots x_p^{\epsilon_p^{(2)}}}(y_1,\dots,y_{n-p})$ $= (\delta_1^{(2)},\dots,\delta_m^{(2)})$ und $\delta_i^{(1)} = 0$, $\delta_i^{(2)} = 1$ oder $\delta_i^{(1)} = 1$, $\delta_i^{(2)} = 0$ für ein $1 \le i \le m$.

Beispiel 3.18 Gegeben sei folgende Zerlegungsmatrix zu (f_1, f_2, f_3, f_4):

x_{p+1} / x_{p+2} / $x_1\dots x_p$	0 0	0 1	1 0	1 1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\epsilon_1^{(1)}\dots\epsilon_p^{(1)}$	(1100)	($\star$011)	(0$\star$10)	($\star\star\star\star$)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\epsilon_1^{(2)}\dots\epsilon_p^{(2)}$	($\star$100)	(0011)	(0010)	($\star\star\star\star$)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\epsilon_1^{(3)}\dots\epsilon_p^{(3)}$	(0$\star$00)	(0$\star$11)	($\star$0$\star$0)	($\star\star\star\star$)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Eine Zeile mit Index ϵ stellt dann eine Funktionstabelle zum Kofaktor $f_{x_1^{\epsilon_1}\dots x_p^{\epsilon_p}}$ dar. Die Paare von Zeilenindizes $(\epsilon^{(1)},\epsilon^{(2)})$ und $(\epsilon^{(2)},\epsilon^{(3)})$ sind dann kompatibel, das Paar $(\epsilon^{(1)},\epsilon^{(3)})$ nicht.

Sind zwei Zeilenindizes kompatibel, so kann man don't cares so belegen, daß die zugehörigen Zeilen gleich werden.
Aus den Zeilen

$$\boxed{(1100)\quad(\star011)\quad(0\star10)\quad(\star\star\star\star)}$$

und

$$\boxed{(\star100)\quad(0011)\quad(0010)\quad(\star\star\star\star)}$$

ergibt sich z.B. mit Belegung einer minimalen Anzahl von don't cares

$$\boxed{(1100)\quad(0011)\quad(0010)\quad(\star\star\star\star)}.$$

Auch hier wird wieder (ähnlich wie bei Funktionen mit einem Ausgang) eine gemeinsame Erweiterung kompatibler Kofaktoren benötigt:

Bezeichnung 3.10 *Seien für* $1 \leq i \leq m$ $(m > 1)$ $f_i \in S(D_i)$, $D_i \subseteq \{0,1\}^n$. *Seien* $cof_1 = f_{x_1^{\epsilon_1^{(1)}} \dots x_p^{\epsilon_p^{(1)}}}$, $\dots$, $cof_l = f_{x_1^{\epsilon_1^{(l)}} \dots x_p^{\epsilon_p^{(l)}}}$ *Kofaktoren von* $f = (f_1, \dots, f_m)$ *und seien* $\epsilon^{(1)}$, $\dots$, $\epsilon^{(l)}$ *paarweise kompatibel. Dann heißt die Funktion*

$$gem\_erw(\{cof_1, \dots, cof_l\}) : \{0,1\}^{n-p} \to \{0,1,\star\}$$

mit

$$gem\_erw(\{cof_1, \dots, cof_l\})(y_1, \dots, y_{n-p}) = (\delta_1, \dots, \delta_m),$$

wobei

$$\delta_i = \begin{cases} 0 & \text{falls für ein } j \in \{1, \dots, l\} \ cof_j(y_1, \dots, y_{n-p}) = (\epsilon_1, \dots, \epsilon_m) \text{ mit } \epsilon_i = 0, \\ 1 & \text{falls für ein } j \in \{1, \dots, l\} \ cof_j(y_1, \dots, y_{n-p}) = (\epsilon_1, \dots, \epsilon_m) \text{ mit } \epsilon_i = 1, \\ \star & \text{sonst} \end{cases}$$

gemeinsame Erweiterung von $cof_1, \dots, cof_l$.

Um die Anzahl der verschiedenen Zeilenmuster in der Zerlegungsmatrix $Z(X^{(1)})$ zu minimieren, wird $\{0,1\}^p$ in Klassen $PK_1, \dots PK_l$ eingeteilt, wobei die Elemente einer solchen Klasse PK_i $(1 \leq i \leq l)$ paarweise kompatibel sind. Die Minimierung von l erfolgt hierbei analog zum Verfahren bei partiellen Funktionen mit einem Ausgang durch Lösung einer Instanz des Problems „Partition into Cliques“ (bis auf die Tatsache, daß Kompatibilität hier anders definiert ist). Ist $\{0,1\}^p/_\equiv = \{K_1, \dots, K_{vz(\{x_1,\dots,x_p\},f)}\}$ die durch *Zeilengleichheit* in $Z(\{x_1, \dots, x_p\})$ induzierte Äquivalenzklasseneinteilung von $\{0,1\}^p$, so kann man auch hier $\{PK_1, \dots, PK_l\}$ so wählen, daß $PK_i = \bigcup_{j=1}^{s_i} K_{i_j}$, d.h. daß alle Indizes gleicher Zeilen von $Z(\{x_1, \dots, x_p\})$ in der gleichen Kompatibilitätsklasse sind.
Für alle $i \in \{1, \dots, l\}$ werden dann die Kofaktoren hinsichtlich der Elemente von PK_i ersetzt durch ihre gemeinsame Erweiterung. So erhält man $f'_1, \dots, f'_m$, so daß die Zerlegungsmatrix $Z(X^{(1)})$ von f' genau l verschiedene Zeilenmuster hat.

Lösung für ROBDD–Darstellungen Es soll abschließend noch kurz angegeben werden, wie sich das Verfahren überträgt, wenn die partiellen Booleschen Funktionen durch ROBDDs zu $ext(f_1), \dots, ext(f_m)$ gegeben sind. Nimmt man an, daß die Variablen aus $X^{(1)} = \{x_1, \dots, x_p\}$ in der Ordnung vor den Variablen aus $\{x_{p+1}, \dots, x_n\}$ stehen und die zusätzliche z–Variable der $ext(.)$–Darstellung ebenfalls in der Ordnung nach den Variablen aus $X^{(1)}$ steht, so kann man für die einzelnen Funktionen f_i die Äquivalenzklasseneinteilungen hinsichtlich $\equiv_i$ durch Schnitt des ROBDD zu $ext(f_i)$ nach den Variablen aus $X^{(1)}$ leicht berechnen (siehe Abschnitt 3.2.3). $\equiv_i$ ist die Äquivalenzrelation, die durch Zeilengleichheit in der Zerlegungsmatrix $Z(X^{(1)})$ von f_i induziert wird. Man erhält $\{0,1\}^p/_{\equiv_i} = \{K_1^{(i)}, \dots, K_{vz(\{x_1,\dots,x_p\},f_i)}^{(i)}\}$. Für das obige Verfahren benötigt man jedoch die durch Zeilengleichheit in der Zerlegungsmatrix zu f induzierte Äquivalenzklasseneinteilung von $\{0,1\}^p$.

Da zwei Zeilen in der Zerlegungsmatrix zu f genau dann gleich sind, wenn die entsprechenden Zeilen *aller* Zerlegungsmatrizen zu f_i $(1 \leq i \leq m)$ gleich sind, ergibt sich $\{0,1\}^p/_\equiv$ als

$$\{\bigcap_{i=1}^{m} K_{j_i}^{(i)} \mid 1 \leq j_i \leq vz(\{x_1,\ldots,x_p\},f_i) \text{ und } \bigcap_{i=1}^{m} K_{j_i}^{(i)} \neq \emptyset\}.$$

Um $\{0,1\}^p/_\equiv$ zu bestimmen, muß man aber nicht alle Konjunktionen $\bigcap_{i=1}^m K_{j_i}^{(i)}$ bilden. Man kann $\{0,1\}^p/_\equiv = \{K_1,\ldots,K_{vz(\{x_1,\ldots,x_p\},f)}\}$ auch direkt mit BDD–Techniken berechnen [W95]:
Man schneidet die ROBDDs zu $ext(f_i)$ nach den Variablen aus $X^{(1)}$. Man erhält bei $ext(f_i)$ $vz(\{x_1,\ldots,x_p\},f_i)$ verschiedene Verbindungsknoten $n_1^{(i)},\ldots,n_{vz(\{x_1,\ldots,x_p\},f_i)}^{(i)}$. Der Knoten $n_j^{(i)}$ ist genau durch die Vektoren aus $K_j^{(i)}$ erreichbar. Ersetze nun die Verbindungsknoten $n_j^{(i)}$ durch die ROBDDs zu eindeutigen neuen Variablen $v_j^{(i)}$. Man erhält dadurch aus dem ROBDD zu $ext(f_i)$ einen ROBDD zu

$$kl^{(i)} = \bigvee_{j=1}^{vz(X^{(1)},f_i)} v_j^{(i)} \cdot \chi_{K_j^{(i)}},$$

wobei $\chi_{K_j^{(i)}}$ die charakteristische Funktion zu $K_j^{(i)}$ ist. Die Klassen K_i erhält man, indem man den ROBDD zu $kl = \bigwedge_{i=1}^m kl^{(i)}$ berechnet:

$$\begin{aligned}
kl &= \bigwedge_{i=1}^{m} kl^{(i)} \\
&= \bigwedge_{i=1}^{m} \Big(\bigvee_{j=1}^{vz(X^{(1)},f_i)} v_j^{(i)} \cdot \chi_{K_j^{(i)}} \Big) \\
&= \bigvee_{\substack{1 \leq j_1 \leq vz(X^{(1)},f_1) \\ \vdots \\ 1 \leq j_m \leq vz(X^{(1)},f_m)}} (v_{j_1}^{(1)} \cdot \ldots \cdot v_{j_m}^{(m)}) \cdot (\chi_{K_{j_1}^{(1)}} \cdot \ldots \cdot \chi_{K_{j_m}^{(m)}}) \\
&= \bigvee_{\substack{1 \leq j_1 \leq vz(X^{(1)},f_1) \\ \vdots \\ 1 \leq j_m \leq vz(X^{(1)},f_m)}} (v_{j_1}^{(1)} \cdot \ldots \cdot v_{j_m}^{(m)}) \cdot (\chi_{K_{j_1}^{(1)} \cap \ldots \cap K_{j_m}^{(m)}}) \\
&= \bigvee_{\substack{i=1 \\ K_{i_1}^{(1)} \cap \ldots \cap K_{i_m}^{(m)} = K_i}}^{vz(X^{(1)},f)} (v_{i_1}^{(1)} \cdot \ldots \cdot v_{i_m}^{(m)}) \cdot \chi_{K_i}
\end{aligned}$$

Schneidet man den ROBDD zu kl nach den Variablen in $X^{(1)}$, so erhält man $vz(X^{(1)},f)$ Verbindungsknoten $n_1,\ldots,n_{vz(X^{(1)},f)}$ direkt unterhalb der Schnittlinie. Die Menge aller Elemente von $\{0,1\}^p$, durch die der Knoten n_i erreicht wird, ist K_i. (Der ROBDD, dessen Wurzel n_i ist, definiert eine Funktion $v_{i_1}^{(1)} \cdot \ldots \cdot v_{i_m}^{(m)}$, wobei $K_i = K_{i_1}^{(1)} \cap \ldots \cap K_{i_m}^{(m)}$.)

Zwei Klassen $K_j = K_{j_1}^{(1)} \cap \ldots \cap K_{j_m}^{(m)}$ und $K_k = K_{k_1}^{(1)} \cap \ldots \cap K_{k_m}^{(m)}$ sind genau dann kompatibel, wenn für alle $1 \leq i \leq m$ die Klassen $K_{j_i}^{(i)}$ und $K_{k_i}^{(i)}$ kompatibel sind. Dies wird getestet wie in Abschnitt 3.2.3 angegeben.

Nach Lösung einer Instanz von „Partition into Cliques" erhält man eine Partition $\{PK_1, \ldots, PK_l\}$ von $\{0,1\}^p$. Die einzelnen Klassen PK_j sind Vereinigungen kompatibler Klassen K_{j_k}. Es gelte $PK_j = \bigcup_{k=1}^{s_j} K_{j_k}$. Der ROBDD zu den Funktionen $ext(f_i')$ ergibt sich dann völlig analog zum Verfahren aus Abschnitt 3.2.3:
Bei Schnitt des ROBDDs zu $ext(f_i)$ nach den Variablen aus $X^{(1)}$ erhält man genau $vz(\{x_1, \ldots, x_p\}, f_i)$ verschiedene Verbindungsknoten $n_1^{(i)}, \ldots, n_{vz(\{x_1,\ldots,x_p\},f_i)}^{(i)}$ direkt unterhalb der Schnittlinie. Der Knoten $n_j^{(i)}$ ist genau durch die Vektoren aus $K_j^{(i)}$ erreichbar. Der ROBDD, dessen Wurzel $n_j^{(i)}$ ist, realisiere die Funktion $cof_j^{(i)}$. Einen ROBDD zu $ext(f_i')$ erhält man dann durch Ersetzungen von Verbindungsknoten im ROBDD zu kl: Ist eine Klasse $PK_j = \bigcup_{k=1}^{s_j} K_{j_k}$, so müssen die Knoten n_{j_k} im ROBDD zu kl (die durch Vektoren aus K_{j_k} erreichbar sind) ersetzt werden durch eine gemeinsame Erweiterung bestimmter Kofaktoren von $ext(f_i)$. Sei $n_{ind(j_k)}^{(i)}$ für $1 \leq k \leq s_j$ der eindeutige Verbindungsknoten im ROBDD zu $ext(f_i)$, der durch einen beliebigen Vektor aus K_{j_k} ($K_{j_k} \subseteq K_{ind(j_k)}^{(i)}$) erreichbar ist und $cof_{ind(j_k)}^{(i)}$ der zugehörige Kofaktor. Dann müssen die Knoten n_{j_k} $(1 \leq k \leq s_j)$ im ROBDD zu kl ersetzt werden durch ROBDDs zu $gem\_erw(\{cof_{ind(j_1)}^{(i)}, \ldots, cof_{ind(j_{s_j})}^{(i)}\})$. Führt man diese Ersetzungen für alle $1 \leq j \leq l$ durch, so erhält man einen ROBDD zu $ext(f_i')$. Die ROBDDs zu $gem\_erw(\{cof_{ind(j_1)}^{(i)}, \ldots, cof_{ind(j_{s_j})}^{(i)}\})$ werden wiederum erzeugt wie in Abschnitt 3.2.3 beschrieben.

Das beschriebene Verfahren berechnet Erweiterungen $f_1', \ldots, f_m'$ von $f_1, \ldots, f_m$, so daß die Anzahl der verschiedenen Zeilenmuster der Zerlegungsmatrix $Z(X^{(1)})$ von f' minimiert wird. $f_1', \ldots, f_m'$ sind aber nicht notwendigerweise schon *totale* Funktionen. Es bleiben gegebenenfalls noch Freiheiten bei der Belegung der restlichen don't cares. Belegt man die don't cares nicht alle mit dem gleichen Wert, so kann die Anzahl der Zeilenmuster der Zerlegungsmatrix für alle Funktionen zusammen evtl. wieder anwachsen. Belegt man die don't cares jedoch durch das in Abschnitt 3.2.3 beschriebene Verfahren für die einzelnen Funktionen f_i', so wächst diese Anzahl nicht an. Dies liegt daran, daß Zeilen mit gleichem Zeilenmuster in der Zerlegungsmatrix zu der Gesamtfunktion natürlich auch in der Zerlegungsmatrix zu einer einzelnen Funktion f_i' gleich sind. Durch das Verfahren aus Abschnitt 3.2.3 können aber Zeilen mit gleichem Zeilenmuster nicht verschieden werden.

Es gilt also folgendes Lemma:

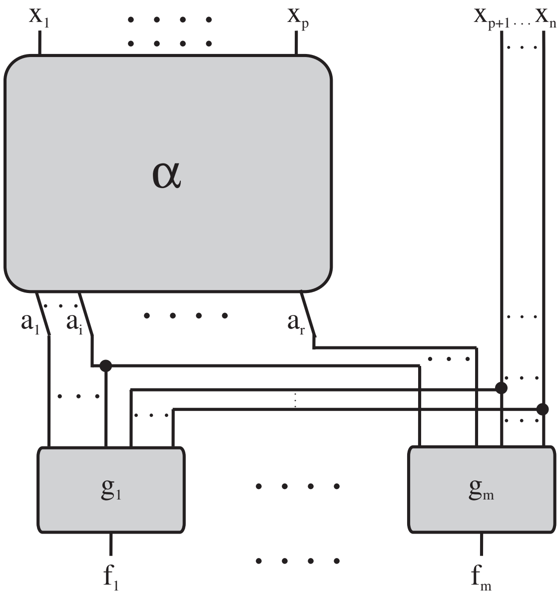
Lemma 3.13 *Sei $f = (f_1, \ldots f_m)$ gegeben mit $f_i \in S(D_i)$, $D_i \subseteq \{0,1\}^n$ $(1 \leq i \leq m)$. Sei $X^{(1)} \subseteq \{0,1\}^n$. Seien $\tilde{f}_i$ aus f_i $(1 \leq i \leq m)$ hervorgegangen durch das Verfahren zur Lösung von EKM aus den Abschnitten 3.2.2 und 3.2.3 bzgl. $X^{(1)}$. Dann gilt:*

$$vz(X^{(1)}, f) \geq vz(X^{(1)}, \tilde{f}).$$

Man hat also zwei Verfahren zur Ausnutzung der don't cares: Das gerade beschriebene Verfahren, das die untere Schranke $\lceil \log(vz(X^{(1)}, f)) \rceil$ für die Gesamtzahl der benötigten Zerlegungsfunktionen in der Zerlegung von $f_1, \ldots, f_m$ minimiert und das Verfahren aus Abschnitt 3.2.3, das die Anzahl der Zerlegungsfunktionen in der Zerlegung der einzelnen Funktionen f_i getrennt minimiert. Die beiden Verfahren sind insofern „verträglich" als man zuerst das Verfahren zur Minimierung der unteren Schranke (im Hinblick auf eine Zerlegung von $f'_1, \ldots, f'_m$ mit Berechnung gemeinsamer Zerlegungsfunktionen) anwenden kann und danach verbleibende don't cares durch das Verfahren aus Abschnitt 3.2.3 ausnutzen kann, ohne diese untere Schranke zu erhöhen.

3.4.4 Rekursive Zerlegungen

Durch die Zerlegung einer Booleschen Funktion f mit n Eingängen und m Ausgängen erhält man folgende Darstellung von f:



Mehrfachverwendbare Teillogik wird erzeugt durch die Bestimmung gemeinsamer Zerlegungsfunktionen.

Das Zerlegungsverfahren kann fortgesetzt werden durch rekursives Aufrufen des Algorithmus für die Blöcke α, g_1, $\ldots$, g_m. Dies hätte allerdings zur Folge, daß die Funktionen $g_1, \ldots, g_m$ alle getrennt realisiert würden. Aus diesem Grund werden diese Funktionen im implementierten Logiksyntheseverfahren mit Hilfe eines greedy–Verfahrens „zusammengruppiert", d.h. die rekursiven Aufrufe erfolgen für Funktionen $G_1, \ldots, G_k$, in denen jeweils mehrere Ausgangsfunktionen g_i zusammengefaßt sind. Im allgemeinen werden jedoch nicht alle Funktionen g_i zu einem einzigen G_1 zusammengefaßt, da beispielsweise Ausgangsfunktionen, die keine Eingangsvariablen gemeinsam haben, auch keine gemeinsame Teillogik nutzen können. Da die Laufzeit des Verfahrens zur Bestimmung von geeigneten Variablenaufteilungen für die Zerlegung aber von der Anzahl der Eingangsvariablen abhängt, soll die Anzahl der Eingangsvariablen der Funktionen G_l nicht unnötig groß werden. Kriterium für das Zusammenfassen von Ausgangsfunktionen g_i und g_j ist die Anzahl der gemeinsamen Eingangsvariablen von g_i und g_j, d.h. die Wahrscheinlichkeit, daß zwei Ausgangsfunktionen g_i und g_j in einem G_l zusammengefaßt sind, steigt mit der Anzahl der gemeinsamen Eingangsvariablen von g_i und g_j. Bei der Bestimmung von G_l werden solange Ausgangsfunktionen g_i zu G_l hinzugefügt, bis die Anzahl der Eingänge von G_l eine vorgegebene Maximalanzahl überschreiten würde.

3.5 Ausnutzung von Symmetrien

In diesem Abschnitt wird festgehalten, wie man Symmetrien im Rahmen des Zerlegungsverfahrens nutzen kann. Im ersten Teil über totale Funktionen wird gezeigt, wie sich Informationen über Symmetrien (Vertauschungs– oder Äquivalenzsymmetrien) ausnutzen lassen, um geeignete Variablenaufteilungen für die Zerlegung zu finden.

Der zweite Teil des Abschnitts befaßt sich mit partiellen Funktionen. Die Tatsache, daß sich Symmetrien für die Zerlegung ausnutzen lassen, motiviert die Suche nach totalen Erweiterungen mit möglichst vielen Symmetrieeigenschaften mit Hilfe der schon in Kapitel 2.2 angegebenen Verfahren.

Die Belegung von don't cares zur Erzeugung starker Symmetrien stellt im Gegensatz zu dem Verfahren aus Abschnitt 3.2.3 zur Minimierung der Anzahl von Zerlegungsfunktionen ein eher globales Verfahren dar, da es nicht an eine bestimmte Variablenaufteilung bzw. Variablenordnung eines ROBDDs gebunden ist und sich die Symmetrien bei Durchführung von *strikten* Zerlegungen auf die Zerlegungsfunktionen übertragen. Im vorliegenden Abschnitt wird gezeigt, daß die beiden Verfahren aber nicht unbedingt in Konkurrenz zueinander stehen, sondern sich gegenseitig ergänzen: Unter bestimmten Voraussetzungen kann garantiert werden, daß das Verfahren zur Minimierung der Anzahl von Zerlegungsfunktionen keine der bereits erzeugten starken Symmetrien zerstört.

Schließlich werden noch die Auswirkungen der beiden Verfahren anhand experimenteller Ergebnisse zur ROBDD–Minimierung demonstriert.

3.5.1 Symmetrieausnutzung bei totalen Funktionen

Sucht man eine Realisierung für eine Boolesche Funktion $f \in B_n$ und hat man Informationen über gewisse Symmetrieeigenschaften von f zur Verfügung, so kann man diese Informationen im Zusammenhang mit Zerlegungen ausnutzen.

Ist von einer Funktion $f \in B_n$ beispielsweise bekannt, daß sie symmetrisch ist in einer Teilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ der Eingangsvariablen ($p > 2$), d.h. daß sie invariant gegenüber sämtlichen Vertauschungen von Variablen aus $X^{(1)}$ ist, so kann dieses Wissen ausgenutzt werden, indem man eine nichttriviale Zerlegung hinsichtlich $X^{(1)}$ durchführt. Die Zerlegungsmatrix $Z(X^{(1)})$ hinsichtlich $X^{(1)}$ kann dann höchstens $(p+1)$ verschiedene Zeilen enthalten, denn für alle $(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_n)$ und $(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_n) \in \{0,1\}^n$ mit gleichem 1–Gewicht $w^1_{X^{(1)}}$ des $X^{(1)}$–Teils ist der Funktionswert von f gleich. Zeilen der Zerlegungsmatrix, deren Indizes das gleiche 1–Gewicht haben, sind also gleich. Man kann f (für $p > 2$) nichttrivial zerlegen in der Form

$$f(x_1, \ldots, x_p, x_{p+1}, \ldots, x_n) = g(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_r(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$$

mit $r \leq \lceil \log(p+1) \rceil$.

Betrachtet man außer Vertauschungssymmetrie auch Äquivalenzsymmetrie, so erhält man allgemein folgende Aussage:

Satz 3.11 *Sei $f \in B_n$ eine totale Boolesche Funktion auf der Variablenmenge $X = \{x_1, \ldots, x_n\}$ und sei $X^{(1)} = \{x_1, \ldots, x_p\}$ eine Teilmenge von X. Sei auf $X^{(1)}$ die Äquivalenzrelation $\sim_{esym}$ definiert mit*

$$x_i \sim_{esym} x_j \quad \Longleftrightarrow \quad \forall x_i, x_j \in X^{(1)}: \quad f \textit{ ist erweitert symmetrisch}^{8} \textit{in } (x_i, x_j).$$

Sei $P_{\sim_{esym}} = \{\mu_1, \ldots, \mu_l\}$ die Partition auf $X^{(1)}$, die durch die Äquivalenzklassen der Relation $\sim_{esym}$ gebildet wird. $\mu_1, \ldots, \mu_t$ $(0 \leq t \leq l)$ enthalten o.B.d.A. jeweils ein mehrfachsymmetrisches Variablenpaar (und damit sind alle *Paare von Variablen in μ_i $(1 \leq i \leq t)$ mehrfachsymmetrisch[9]).*
Für die Anzahl der Zeilenmuster der Zerlegungsmatrix $Z(X^{(1)})$ von f hinsichtlich $X^{(1)}$ gilt

$$vz(X^{(1)}, f) \leq 2^t \cdot (|\mu_{t+1}| + 1) \cdot (|\mu_{t+2}| + 1) \cdot \ldots \cdot (|\mu_l| + 1).$$

Folglich gibt es eine Zerlegung von f hinsichtlich $X^{(1)}$ mit höchstens

$$\lceil \log(2^t \cdot (|\mu_{t+1}| + 1) \cdot (|\mu_{t+2}| + 1) \cdot \ldots \cdot (|\mu_l| + 1)) \rceil = \lceil \sum_{i=t+1}^{l} \log(|\mu_i| + 1) \rceil + t$$

Zerlegungsfunktionen.

Beweis: Sei $P = (p_1, \ldots, p_p)$ ein Polaritätsvektor, der konstruiert wird wie im Beweis zu Satz 2.1 (Seite 43), d.h. bei $\mu_i = \{x_{i_i}, \ldots, x_{i_k}\}$ sind

- für $1 \leq i \leq t$, d.h. falls μ_i eine Menge mit mehrfachsymmetrischen Variablenpaaren ist, $p_{i_1}, \ldots, p_{i_k}$ beliebig,

[8] f ist nach Kapitel 2 erweitert symmetrisch in einem Variablenpaar, wenn f vertauschungssymmetrisch *oder* äquivalenzsymmetrisch in dem Variablenpaar ist.
[9] siehe Lemma 2.6, Seite 43

- und sonst $p_{i_1}, \ldots, p_{i_k}$ nach dem folgenden Verfahren bestimmt: Wähle $p_{i_1} = \epsilon \in \{0,1\}$ beliebig. Falls f in (x_{i_1}, x_{i_2}) vertauschungssymmetrisch, dann wähle $p_{i_2} = \epsilon$, falls f in (x_{i_1}, x_{i_2}) äquivalenzsymmetrisch, dann wähle $p_{i_2} = \overline{\epsilon}$. Führe das Verfahren analog fort mit (x_{i_2}, x_{i_3}) usw. bis p_{i_k} bestimmt ist.

(f_P mit $f_P(x_1, \ldots, x_n) = f(x_1^{p_1}, \ldots, x_p^{p_p}, x_{p+1}, \ldots, x_n)\ \forall (x_1, \ldots, x_n) \in \{0,1\}^n$ ist dann vertauschungssymmetrisch in $P_{\sim_{esym}}$.)

Sei für eine Menge $\mu_i = \{x_{i_i}, \ldots, x_{i_k}\}$ das P–Gewicht eines $(\epsilon_1, \ldots, \epsilon_p) \in \{0,1\}^p$ definiert durch

$$w^P_{\mu_i}(\epsilon_1, \ldots, \epsilon_p) = |\{\epsilon_{i_j} \mid \epsilon_{i_j} = p_{i_j}, j \in \{1, \ldots, k\}\}|.$$

(Falls $P = (1, \ldots, 1)$ (also bei reiner Vertauschungssymmetrie) ist $w^P_{\mu_i}$ also genau das 1–Gewicht $w^1_{\mu_i}$ des μ_i–Teils.)

Zur Berechnung von $vz(X^{(1)}, f)$ ist die Anzahl der verschiedenen Kofaktoren $f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}$ für alle $(\epsilon_1, \ldots, \epsilon_p) \in \{0,1\}^p$ zu bestimmen.

<u>Behauptung 1</u>: Für $i \in \{t+1, \ldots, l\}$ und beliebige $\epsilon, \delta \in \{0,1\}^p$ mit

- $\epsilon_j = \delta_j$, falls $x_j \notin \mu_i$ und
- $w^P_{\mu_i}(\epsilon_1, \ldots, \epsilon_p) = w^P_{\mu_i}(\delta_1, \ldots, \delta_p)$

gilt:

$$f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}} = f_{x_1^{\delta_1} \ldots x_p^{\delta_p}}.$$

Sei $\mu_i = \{x_{i_1}, \ldots, x_{i_k}\}$.
Konstruiere dann eine Folge $\delta^{(1)}, \ldots, \delta^{(k)} \in \{0,1\}^p$ mit

$$f_{x_1^{\delta_1^{(1)}} \ldots x_p^{\delta_p^{(1)}}} = \ldots = f_{x_1^{\delta_1^{(k)}} \ldots x_p^{\delta_p^{(k)}}}, \text{ wobei}$$

- $\delta^{(1)} = \delta$,
- $\delta_j = \delta_j^{(1)} = \ldots = \delta_j^{(k)} = \epsilon_j$ für $j \notin \{i_1, \ldots, i_k\}$,
- $(\delta_{i_1}^{(j+1)}, \ldots, \delta_{i_j}^{(j+1)}) = (\epsilon_{i_1}, \ldots, \epsilon_{i_j})$ für $1 \leq j \leq k-1$,
- $w^P_{\mu_i}(\delta^{(j+1)}) = w^P_{\mu_i}(\delta^{(j)})$ für $1 \leq j \leq k-1$.

Wegen $(\delta_{i_1}^{(k)}, \ldots, \delta_{i_k}^{(k)}) = (\epsilon_{i_1}, \ldots, \epsilon_{i_{k-1}}, \delta_{i_k}^{(k-1)})$ und $w^P_{\mu_i}(\delta^{(k)}) = w^P_{\mu_i}(\delta) = w^P_{\mu_i}(\epsilon)$ gilt dann $\delta^{(k)} = \epsilon$.
Die Konstruktion nutzt die Vertauschungs– und Äquivalenzsymmetrie in μ_i aus:
Falls $\delta_{i_j}^{(j)} = \epsilon_{i_j}$, wähle $\delta^{(j+1)} = \delta^{(j)}$.
Falls $\delta_{i_j}^{(j)} = \overline{\epsilon_{i_j}}$, sind 2 Fälle zu unterscheiden:

1. Fall: $\epsilon_{i_j} = p_{i_j}$

Dann ist $\delta_{i_j}^{(j)} \neq p_{i_j}$.

Bestimme nun m mit $j < m \leq k$, so daß $\delta_{i_m}^{(j)} = p_{i_m}$.

Ein solches m muß es auf jeden Fall geben, da sonst wegen $\delta_{i_j}^{(j)} \neq p_{i_j}$ und $\epsilon_{i_j} = p_{i_j}$ $w_{\mu_i}^P(\delta_1^{(j)}, \ldots, \delta_p^{(j)}) < w_{\mu_i}^P(\epsilon_1, \ldots, \epsilon_p) = w_{\mu_i}^P(\delta_1, \ldots, \delta_p)$ wäre.

Definiere

$$\delta_s^{(j+1)} = \begin{cases} \overline{\delta_s^{(j)}}, & \text{falls } s = i_j \text{ oder } s = i_m \\ \delta_s^{(j)}, & \text{sonst.} \end{cases}$$

Es gilt dann

$$(\delta_{i_1}^{(j+1)}, \ldots, \delta_{i_j}^{(j+1)}) = (\epsilon_{i_1}, \ldots, \epsilon_{i_j}).$$

- Falls nun $p_{i_m} = p_{i_j}$, d.h. falls f in den Variablen x_{i_j} und x_{i_m} vertauschungssymmetrisch ist, gilt wegen $\delta_{i_m}^{(j)} \neq \delta_{i_j}^{(j)}$ $f_{x_1^{\delta_1^{(j)}} \ldots x_p^{\delta_p^{(j)}}} = f_{x_1^{\delta_1^{(j+1)}} \ldots x_p^{\delta_p^{(j+1)}}}$.
- Falls $p_{i_m} \neq p_{i_j}$, d.h. falls f in den Variablen x_{i_j} und x_{i_m} äquivalenzsymmetrisch ist, gilt wegen $\delta_{i_m}^{(j)} = \delta_{i_j}^{(j)}$ $f_{x_1^{\delta_1^{(j)}} \ldots x_p^{\delta_p^{(j)}}} = f_{x_1^{\delta_1^{(j+1)}} \ldots x_p^{\delta_p^{(j+1)}}}$.

Da $\delta_{i_j}^{(j)} \neq p_{i_j}$, $\delta_{i_m}^{(j)} = p_{i_m}$ und damit $\delta_{i_j}^{(j+1)} = p_{i_j}$, $\delta_{i_m}^{(j+1)} \neq p_{i_m}$, gilt

$$w_{\mu_i}^P(\delta_1^{(j)}, \ldots, \delta_p^{(j)}) = w_{\mu_i}^P(\delta_1^{(j+1)}, \ldots, \delta_p^{(j+1)}).$$

2. Fall: $\epsilon_{i_j} \neq p_{i_j}$

Dann ist $\delta_{i_j}^{(j)} = p_{i_j}$.

Bestimme nun m mit $j < m \leq k$, so daß $\delta_{i_m}^{(j)} \neq p_{i_m}$.

Ein solches m muß es auf jeden Fall geben, da sonst wegen $\delta_{i_j}^{(j)} = p_{i_j}$ und $\epsilon_{i_j} \neq p_{i_j}$ $w_{\mu_i}^P(\delta_1^{(j)}, \ldots, \delta_p^{(j)}) > w_{\mu_i}^P(\epsilon_1, \ldots, \epsilon_p) = w_{\mu_i}^P(\delta_1, \ldots, \delta_p)$ wäre.

$\delta_1^{(j+1)}, \ldots, \delta_p^{(j+1)}$ werden definiert wie in Fall 1 und die restliche Argumentation erfolgt ebenfalls analog zu Fall 1.

Nach $k-1$ Schritten ergibt sich also

$$f_{x_1^{\delta_1} \ldots x_p^{\delta_p}} = f_{x_1^{\delta_1^{(1)}} \ldots x_p^{\delta_p^{(1)}}} = \ldots f_{x_1^{\delta_1^{(k)}} \ldots x_p^{\delta_p^{(k)}}} = f_{x_1^{\epsilon_1} \ldots x_p^{\epsilon_p}}$$

und Behauptung 1 ist bewiesen.

Behauptung 2: Für $i \in \{1, \ldots, t\}$ und beliebige $\epsilon, \delta \in \{0,1\}^p$ mit

- $\epsilon_j = \delta_j$, falls $x_j \notin \mu_i$ und
- $w_{\mu_i}^1(\epsilon_1, \ldots, \epsilon_p) \bmod 2 = w_{\mu_i}^1(\delta_1, \ldots, \delta_p) \bmod 2$

gilt:

$$f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}} = f_{x_1^{\delta_1} \dots x_p^{\delta_p}}.$$

Sei $\mu_i = \{x_{i_1}, \dots, x_{i_k}\}$.
Die Konstruktion für die Variablenmenge μ_i erfolgt in ähnlicher Weise wie bei Behauptung 1 ebenfalls in $|\mu_i|-1$ Schritten: Ist $\delta_{i_j}^{(j)} = \epsilon_{i_j}$, so ist in Schritt j nichts zu tun. Ansonsten wird bei $\delta_{i_j}^{(j)} \neq \delta_{i_{j+1}}^{(j)}$ die Vertauschungssymmetrie in den Variablen x_{i_j} und $x_{i_{j+1}}$ ausgenutzt und bei $\delta_{i_j}^{(j)} = \delta_{i_{j+1}}^{(j)}$ die Äquivalenzsymmetrie in den Variablen x_{i_j} und $x_{i_{j+1}}$. Mit der Definition

$$\delta_s^{(j+1)} = \begin{cases} \overline{\delta_s^{(j)}}, & \text{falls } s = i_j \text{ oder } s = i_{j+1} \\ \delta_s^{(j)}, & \text{sonst.} \end{cases}$$

gilt

$$f_{x_1^{\delta_1^{(j)}} \dots x_p^{\delta_p^{(j)}}} = f_{x_1^{\delta_1^{(j+1)}} \dots x_p^{\delta_p^{(j+1)}}}.$$

Außerdem gilt

$$(\delta_{i_1}^{(j+1)}, \dots, \delta_{i_j}^{(j+1)}) = (\epsilon_{i_1}, \dots, \epsilon_{i_j}).$$

und

$$w_{\mu_i}^1(\delta_1^{(j)}, \dots, \delta_p^{(j)}) \bmod 2 = w_{\mu_i}^1(\delta_1^{(j+1)}, \dots, \delta_p^{(j+1)}) \bmod 2.$$

Nach $k-1$ Schritten erhält man $(\delta_1^{(k)}, \dots, \delta_p^{(k)})$ mit

$$(\delta_{i_1}^{(k)}, \dots, \delta_{i_k}^{(k)}) = (\epsilon_{i_1}, \dots, \epsilon_{i_{k-1}}, \delta_{i_k}^{(k)})$$

und wegen $w_{\mu_i}^1(\delta_1, \dots, \delta_p) \bmod 2 = w_{\mu_i}^1(\delta_1^{(k)}, \dots, \delta_p^{(k)}) \bmod 2 = w_{\mu_i}^1(\epsilon_1, \dots, \epsilon_p) \bmod 2$ auch $\delta_{i_k}^{(k)} = \epsilon_{i_k}$.
Insgesamt gilt

$$f_{x_1^{\delta_1} \dots x_p^{\delta_p}} = \dots = f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}.$$

Da für $t+1 \leq i \leq l$ nur $|\mu_i|+1$ verschiedene Funktionswerte von $w_{\mu_i}^P$ vorkommen können und da es für $1 \leq i \leq t$ nur 2 verschiedene Werte für $w_{\mu_i}^1 \bmod 2$ gibt, kann es also insgesamt höchstens

$$2^t \cdot (|\mu_{t+1}|+1) \cdot \ldots \cdot (|\mu_l|+1)$$

verschiedene Kofaktoren $f_{x_1^{\epsilon_1} \dots x_p^{\epsilon_p}}$ für $(\epsilon_1, \dots, \epsilon_p) \in \{0,1\}^p$ geben. Somit ist der Satz bewiesen. □

Kennt man also größere Symmetriemengen einer Funktion f (wobei hier Symmetrie Vertauschungssymmetrie oder Äquivalenzsymmetrie bedeutet), so bietet es sich nach Satz 3.11 an, die Variablen aus diesen Symmetriemengen zusammen in die Teilmenge $X^{(1)}$ der Variablen zu plazieren, hinsichtlich der zerlegt wird. Obwohl man Beispielsfunktionen konstruieren kann, bei denen die Anzahl der benötigten Zerlegungsfunktionen kleiner wird, wenn man die Variablenpaare, in denen die Funktionen symmetrisch sind, *nicht* zusammen in die Menge $X^{(1)}$ plaziert (siehe [SM93]), scheint es doch eine gute Heuristik zu sein, symmetrische Variablen zusammenzufassen, da größere Symmetriemengen nach Satz 3.11 schon von vornherein zu einer geringen Anzahl von Zerlegungsfunktionen führen.

Gruppiert man also die Variablen aus den Symmetriemengen zusammen, so kann man das Verfahren zur Bestimmung geeigneter Inputpartitionierungen aus den Kapiteln 3.1.4 bzw. 3.4.1 abändern: Es werden dann nicht mehr einzelne Variablen miteinander vertauscht, sondern ganze Blöcke von Variablen, nämlich gerade die Variablen aus den Symmetriemengen. Ansonsten bleibt das Verfahren unverändert.

Ist $P_{\sim_{esym}} = \{\mu_1, \ldots, \mu_l\}$ die Partition auf der gesamten Variablenmenge $X = \{x_1, \ldots, x_n\}$ von f, die durch die Äquivalenzklassen der Relation $\sim_{esym}$ (Vertauschungssymmetrie *oder* Äquivalenzsymmetrie) auf X gebildet wird, und sind die Mengen μ_i ausreichend groß, so bietet sich noch eine andere Möglichkeit zur Ausnutzung dieser Symmetrien an, die in dem implementierten Logiksynthesealgorithmus genutzt wird: Man kann f l–seitig hinsichtlich der Variablenpartition $P_{\sim_{esym}}$ zerlegen. In Kapitel 4 wird unter anderem eine Realisierung eines partiellen Multiplizierers vorgestellt, die eine solche l–seitige Zerlegung benutzt.

3.5.2 Symmetrieausnutzung bei partiellen Funktionen

Satz 3.11 besagt, daß Symmetrieeigenschaften bei totalen Funktionen zu Zerlegungen mit einer geringen Anzahl von Zerlegungsfunktionen führen. Daher lohnt es sich im Hinblick auf die Zerlegung bei partiellen Funktionen, totale Erweiterungen zu suchen, die möglichst viele Symmetrieeigenschaften aufweisen. Dies kann mit Hilfe der Verfahren aus Kapitel 2.2 geschehen.

Auf der anderen Seite wurde in Abschnitt 3.2.3 ein lokales Verfahren vorgestellt, das die Anzahl der Zerlegungsfunktionen bei vorgegebener Variablenteilmenge $X^{(1)}$, hinsichtlich der zerlegt werden soll, minimiert und dabei ebenfalls don't cares belegt. ROBDD–Größen lassen sich durch wiederholte Anwendung dieses Verfahrens minimieren, z.B. durch Anwendung für $X^{(1)} = \{x_1, \ldots, x_i\}$ mit $i = 1, \ldots, n-1$ (siehe Kapitel 3.2.4). (Wie bereits festgestellt wurde, besteht ein enger Zusammenhang zwischen der Minimierung von ROBDD–Größen und der Minimierung der Anzahl von Zerlegungsfunktionen.)

In diesem Abschnitt wird gezeigt, daß sich die beiden Ansätze (Belegung von don't cares zum Erzeugen von Symmetrien und Belegen von don't cares zur Minimierung der Anzahl von Zerlegungsfunktionen bei vorgegebener Variablenteilmenge bzw. zur Minimierung von ROBDD–Größen) kombinieren lassen. Die Erzeugung von Symmetrien dient dabei in einem ersten Schritt sowohl der Erzeugung guter Variablenordnungen (d.h. „symmetrischer Ordnungen“ im Hinblick auf die Minimierung von ROBDD–Größen) bzw. der Bestimmung guter Variablenteilmengen $X^{(1)}$ als auch der don't care–Belegung zur Minimierung von ROBDD–Größen bzw. Zerlegungsfunktionsanzahlen bzgl. der resultierenden Ordnung. (Die Wahl „symmetrischer Ordnungen“ hat sich bei der ROBDD–Repräsentation von totalen Funktionen als günstig erwiesen [MMD94, PSP94].) In einem zweiten Schritt wird dann

x_3 / x_4 \ x_1x_2	0 / 0	0 / 1	1 / 0	1 / 1
0 0	1	0	$*$	0
0 1	$*$	0	0	0
1 0	$*$	0	0	0
1 1	0	0	0	1

$\longrightarrow$

x_3 / x_4 \ x_1x_2	0 / 0	0 / 1	1 / 0	1 / 1
0 0	1	0	0	0
0 1	1	0	0	0
1 0	1	0	0	0
1 1	0	0	0	1

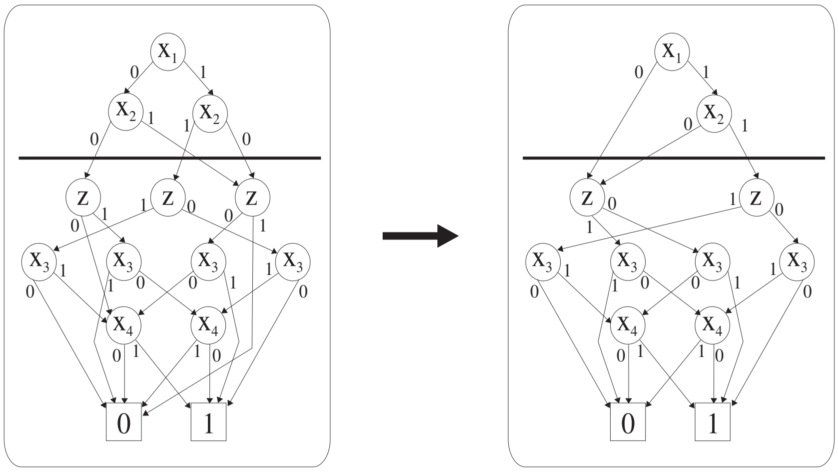


Abbildung 3.28: Zerstörung der Symmetrie in x_1 und x_3 durch Minimierung der Anzahl der Verbindungsknoten bei Schnitt nach x_2.

bei *vorgegebener* Variablenordnung bzw. *vorgegebener* Variablenteilmenge $X^{(1)}$ die ROBDD–Größe bzw. die Anzahl der Zerlegungsfunktionen bei Zerlegung hinsichtlich $X^{(1)}$ durch (einmaliges bzw. wiederholtes) Anwenden des Verfahrens aus Abschnitt 3.2.3 nochmals minimiert.

Es wird gezeigt, daß unter bestimmten Voraussetzungen die don't care–Belegung in Schritt 2 die in Schritt 1 erzeugten Symmetrien *nicht* zerstört. Daß dies allgemein aber nicht gilt, wird anhand des folgenden Beispiels deutlich:

Beispiel 3.19 Betrachte die partielle Funktion $f \in B_4$, die durch die Zerlegungsmatrix bzw. durch den ROBDD auf der linken Seite von Abbildung 3.28 definiert ist. Man sieht leicht, daß f stark symmetrisch ist in $\{x_1, x_2, x_3\}$. Minimiert man die Anzahl der „Verbindungsknoten“ unterhalb eines Schnitts nach der Variablen x_2, so erhält man die Zerlegungsmatrix und den ROBDD auf der rechten Seite der Abbildung. Das Ergebnis ist *nicht* mehr symmetrisch in x_1 und x_3.

Setzt man aber voraus, daß die Variablenordnung des ROBDD eine symmetrische Ordnung ist und schränkt man die Schnitte durch den ROBDD ein auf Grenzen zwischen den einzelnen Gruppen von Variablen, in denen die Funktion stark symmetrisch ist, so zerstört die

Minimierung der „Verbindungsknoten“ die Symmetrieeigenschaften nicht. Ist f also stark symmetrisch in einer Partition $P = \{\lambda_1, \ldots, \lambda_k\}$ der Eingangsvariablen und ist $X^{(1)}$ eine Vereinigung von Mengen λ_i, so zerstört eine Anwendung des Verfahrens zur Lösung von EKM bzgl. $X^{(1)}$ die starke Symmetrie in P nicht.

Dieser Zusammenhang wird in Satz 3.12 präzisiert:

Satz 3.12 *Sei $f \in S(D)$ mit $D \subseteq \{0,1\}^n$. Sei f* stark symmetrisch *in einer Partition $P = \{\lambda_1, \ldots, \lambda_k\}$ der Eingangsvariablen. Sei $X^{(1)} = \bigcup_{j=1}^{l} \lambda_{i_j}$ $(1 \leq l < k)$. Sei $K = \{K_1, \ldots, K_{vz(X^{(1)},f)}\}$ die durch Zeilengleichheit in $Z(X^{(1)})$ induzierte Äquivalenzklasseneinteilung. Sei $PK = \{PK_1, \ldots, PK_v\}$ eine Vergröberung von K, d.h. $PK_i = \bigcup_{j=1}^{l_i} K_{i_j}$, mit der Eigenschaft, daß für alle Mengen PK_i $(1 \leq i \leq v)$ $\epsilon \sim \delta$ gilt für alle $\epsilon, \delta \in PK_i$ (d.h. die Zeilen der Zerlegungsmatrix mit den Indizes ϵ und δ sind kompatibel). Sei $f' : D' \to \{0,1\}$ eine Erweiterung von f mit*

$$D' = \{(\epsilon_1, \ldots, \epsilon_n) \mid \text{ falls } (\epsilon_1, \ldots, \epsilon_p) \in PK_i, \text{ dann } \exists (\delta_1, \ldots, \delta_p) \in PK_i \text{ mit } (\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_n) \in D\}$$

und für alle $(\epsilon_1, \ldots, \epsilon_n) \in D'$:
$f'(\epsilon_1, \ldots, \epsilon_n) = f(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_n)$ mit $(\epsilon_1, \ldots, \epsilon_p) \in PK_i, (\delta_1, \ldots, \delta_p) \in PK_i$ und $(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_n) \in D$.
Dann ist auch f' stark symmetrisch in der Partition P.

Beweis: Wähle $\lambda_i \in P$.
Z.z.: f' ist stark symmetrisch in λ_i.
Sei o.B.d.A. $X^{(1)} = \{x_1, \ldots, x_p\}$.

<u>1. Fall</u>: $\lambda_i \subseteq X^{(1)}$

Seien x_{j_1} und $x_{j_2} \in \lambda_i$.
Seien $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$ beliebig mit

$$\epsilon^{(1)} = (\epsilon_1, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) \text{ und } \epsilon^{(2)} = (\epsilon_1, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n).$$

Da f stark symmetrisch ist in x_{j_1}, x_{j_2} gilt also

$$f_{x_1^{\epsilon_1} \ldots x_{j_1}^{\epsilon_{j_1}} \ldots x_{j_2}^{\epsilon_{j_2}} \ldots x_p^{\epsilon_p}} = f_{x_1^{\epsilon_1} \ldots x_{j_1}^{\epsilon_{j_2}} \ldots x_{j_2}^{\epsilon_{j_1}} \ldots x_p^{\epsilon_p}},$$

wobei zu beachten ist, daß die Kofaktoren evtl. partiell sind.
(Die Zeilen der Zerlegungsmatrix $Z(X^{(1)})$ mit Indizes $\epsilon^{(1)}$ und $\epsilon^{(2)}$ sind also exakt gleich.)
Daher gibt es ein $K_i \in K$ mit $\epsilon^{(1)}, \epsilon^{(2)} \in K_i$ und folglich auch ein $PK_j \in PK$ mit $\epsilon^{(1)}, \epsilon^{(2)} \in PK_j$. Da nach Definition von f' für alle $(\delta_1, \ldots, \delta_p), (\delta'_1, \ldots, \delta'_p) \in PK_j$ gilt

$$f'_{x_1^{\delta_1} \ldots x_p^{\delta_p}} = f'_{x_1^{\delta'_1} \ldots x_p^{\delta'_p}},$$

ist insbesondere auch

$$f'_{x_1^{\epsilon_1}\ldots x_{j_1}^{\epsilon_{j_1}}\ldots x_{j_2}^{\epsilon_{j_2}}\ldots x_p^{\epsilon_p}} = f'_{x_1^{\epsilon_1}\ldots x_{j_1}^{\epsilon_{j_2}}\ldots x_{j_2}^{\epsilon_{j_1}}\ldots x_p^{\epsilon_p}}.$$

(f' ist so definiert, daß durch don't care–Belegung aus gleichen Zeilen der Zerlegungsmatrix $Z(X^{(1)})$ für f wieder gleiche Zeilen in der entsprechenden Zerlegungsmatrix für f' resultieren.)
f' ist also ebenfalls stark symmetrisch in x_{j_1}, x_{j_2}.

<u>2. Fall</u>: $\lambda_i \not\subseteq X^{(1)}$
Seien x_{j_1} und $x_{j_2} \in \lambda_i$.
Seien $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^n$ beliebig mit $\epsilon^{(1)} = (\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n)$ und $\epsilon^{(2)} = (\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n)$.
Da f stark symmetrisch ist in x_{j_1}, x_{j_2} gilt

$$\epsilon^{(1)}, \epsilon^{(2)} \notin D$$

oder

$$\epsilon^{(1)}, \epsilon^{(2)} \in D \text{ und } f(\epsilon^{(1)}) = f(\epsilon^{(2)}).$$

Zu zeigen ist, daß auch für f'

$$\epsilon^{(1)}, \epsilon^{(2)} \notin D'$$

oder

$$\epsilon^{(1)}, \epsilon^{(2)} \in D' \text{ und } f'(\epsilon^{(1)}) = f'(\epsilon^{(2)})$$

gilt.
Falls $\epsilon^{(1)}, \epsilon^{(2)} \in D$, dann gilt die Behauptung, da f' Erweiterung von f ist.
Sei $\epsilon^{(1)}, \epsilon^{(2)} \notin D$ und $\epsilon^{(1)} \in D'$ oder $\epsilon^{(2)} \in D'$. Sei o.B.d.A. $\epsilon^{(1)} \in D'$.
Sei $(\epsilon_1, \ldots, \epsilon_p) \in PK_i$.
Dann gibt es nach Definition von f' ein $(\delta_1, \ldots, \delta_p) \in PK_i$ mit $(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) \in D$. Es gilt

$$f'(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) = f(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n).$$

Da f stark symmetrisch ist in x_{j_1}, x_{j_2}, ist auch $(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) \in D$ und

$$f(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) = f(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n).$$

Nach Definition von f' ist dann $(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) \in D'$ und es gilt

$$f'(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) = f(\delta_1, \ldots, \delta_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n),$$

so daß insgesamt

$$f'(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) = f'(\epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n)$$

gilt.

□

Wendet man das in Abschnitt 3.2.3 beschriebene Verfahren zur Lösung des Problems EKM bei ROBDD–Darstellungen auf eine Funktion $f \in S(D)$ an, die stark symmetrisch ist in $\{\lambda_1, \dots, \lambda_k\}$ und bei der in der ROBDD–Darstellung zu $ext(f)$ die Variablen aus den Mengen λ_i in der Variablenordnung benachbart sind, so erhält man bei „Schnitt des ROBDD zwischen den Variablen aus λ_i und λ_{i+1}" gerade eine Erweiterung von f in der in Satz 3.12 angegebenen Form.

Will man die ROBDD–Größe einer partiellen Funktion f minimieren, so wird insgesamt folgendes Verfahren durchgeführt: Zunächst wird mit dem Verfahren aus Kapitel 2.2 eine möglichst kleine Partition $P = \{\lambda_1, \dots, \lambda_k\}$ bestimmt, so daß f symmetrisch ist in P. Es wird eine minimale Anzahl von don't cares belegt, so daß f *stark* symmetrisch wird in P. Ändere die Variablenordnung $<_{index}$ so, daß für alle $1 \leq i \leq k$

$$\lambda_i = \{x_{index(1+\sum_{j=1}^{i-1} |\lambda_j|)}, \dots, x_{index(|\lambda_i|+\sum_{j=1}^{i-1} |\lambda_j|)}\},$$

d.h. daß $<_{index}$ eine symmetrische Ordnung ist. Nun kann zur Verbesserung der Variablenordnung ein sifting–Verfahren [Rud93] angewendet werden, das die Blöcke symmetrischer Variablen als Ganzes verschiebt (symmetrisches sifting [MMD94, PSP94]). Zur Minimierung der Anzahl der ROBDD–Knoten wird nun ähnlich wie bei dem Verfahren von Chang/Marek–Sadowska [CCM94] die Anzahl der Verbindungsknoten unterhalb bestimmter Schnittlinien durch den ROBDD minimiert. Es werden aber nicht alle möglichen Schnittlinien genommen, sondern nur solche, die „zwischen benachbarten Blöcken symmetrischer Variablen" liegen, d.h. es wird nur an Schnittlinien nach den Variablen $x_{index(\sum_{j=1}^{i} |\lambda_j|)}$ $(1 \leq i \leq k-1)$ die Anzahl der Verbindungsknoten minimiert[10]. Abbildung 3.29 illustriert dieses Vorgehen. Gemäß Satz 3.12 wird die starke Symmetrie in P durch das Verfahren *nicht* zerstört. Daher kann nach der Minimierung der Verbindungsknoten nochmals „symmetrisches sifting" mit den gleichen Gruppen symmetrischer Variablen durchgeführt werden.

Experimente aus [SMHM96] anhand von Benchmarkfunktionen ergeben im Schnitt eine Verkleinerung der ROBDD–Darstellungen von über 70% bei diesem Verfahren im Vergleich zu reinem symmetrischem sifting. In Tabelle 3.2 sind die Resultate angegeben.

Zur Durchführung der Messungen wurden die Benchmarkschaltkreise (totaler Boolescher Funktionen) in eine zweistufige Darstellung gebracht. Aus dem Booleschen Polynom wurde nun eine partielle Boolesche Funktion bestimmt: Die ON–Mengen der einzelnen Monome wurden zufällig mit einer Wahrscheinlichkeit von 40% zur don't care–Menge hinzugefügt. Die drei letzten Funktionen in der Tabelle sind partielle Multiplizierer $partmult_n$. [11]

[10] Dabei wird natürlich jeweils vorausgesetzt, daß die zusätzliche Variable z in der Variablenordnung unterhalb von Variable $x_{index(\sum_{j=1}^{i} |\lambda_j|)}$ liegt.

[11] Die n^2 Eingänge sind die Bits der Partialprodukte und die $2n$ Ausgänge sind die Produktbits. Die don't care–Menge enthält alle Eingangsvektoren, die aufgrund der Tatsache nicht auftreten können, daß die Eingangsvariablen nicht unabhängig voneinander sind, sondern Konjunktionen $a_i b_j$ der Bits der Operanden $(a_1, \dots, a_n)$ und $(b_1, \dots, b_n)$ sind.

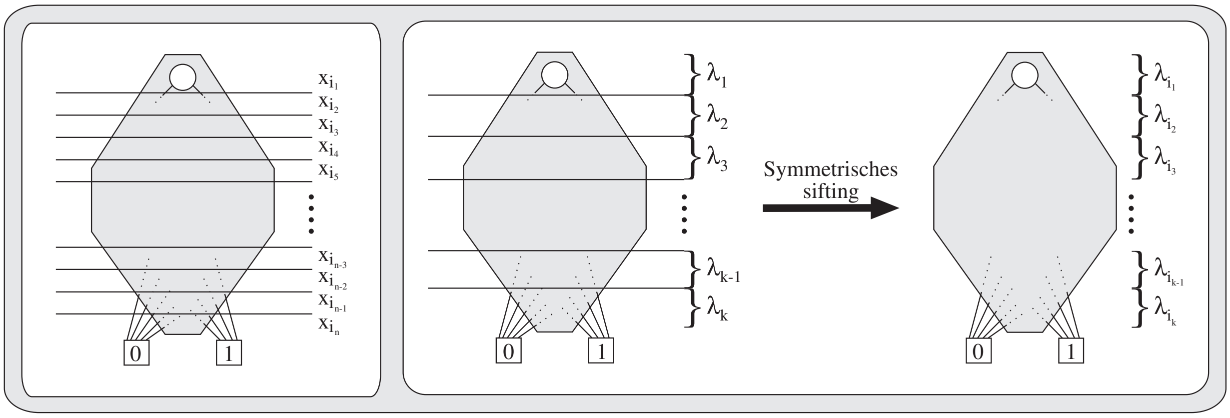


Abbildung 3.29: Es wird nicht wie im Originalverfahren nach jeder Variable geschnitten, sondern nur zwischen den Blöcken symmetrischer Variablen.

In einem ersten Experiment wurden ROBDDs, die die ON–Mengen repräsentierten, aufgebaut und die Variablenordnungen mit symmetrischem sifting optimiert. In Spalte „*sym_sift*“ von Tabelle 3.2 sind die entsprechenden Knotenanzahlen aufgeführt.
Die Spalte „*sym_group*“ enthält die Knotenzahlen nach Belegung von don't cares zur Erzeugung starker Symmetrien. Es ergibt sich eine Verbesserung von durchschnittlich 51%.
In Spalte „*sym_cover*“ sind schließlich die Knotenzahlen angegeben, die man erhält, wenn man das oben beschriebene Verfahren komplett anwendet (einschließlich einer Minimierung der Anzahl von Verbindungsknoten). Man erzielt gegenüber reinem symmetrischem sifting eine Verringerung der Knotenzahlen um über 70%.

3.6 Nichtdisjunkte Dekomposition

Schon in [RK62] wurden durch Roth und Karp nichtdisjunkte Zerlegungen erwähnt. Allerdings erfolgte (meines Wissens nach) keine systematische Untersuchung nichtdisjunkter Zerlegungen. Die Ergebnisse dieses Abschnitts zeigen, daß sich die Betrachtung nichtdisjunkter Zerlegungen ohne größere Probleme in die bisher entwickelten Syntheseverfahren integrieren läßt.

Nichtdisjunkte Zerlegungen sind folgendermaßen definiert:

Definition 3.12 (Einseitige nichtdisjunkte Zerlegung)
Eine **einseitige nichtdisjunkte Zerlegung** *einer Booleschen Funktion* $f \in B_n$ *mit den Eingangsvariablen* $x_1, \ldots, x_n$ *hinsichtlich des Paares* $(X^{(1)}, X^{(2)})$ *von Variablenteilmengen mit* $X^{(1)} = \{x_1, \ldots, x_p\}$, $X^{(2)} = \{x_h, \ldots, x_n\}$, $1 < p < n$ *und* $1 < h < p + 1$ *ist eine*

Schaltkreis	Anzahl Inputs	Anzahl Outputs	*sym_sift*	*sym_group*		*sym_cover*	
5xp1	7	10	67	66	(0.2 s)	53	(0.5 s)
9symml	9	1	108	25	(0.3 s)	25	(0.4 s)
alu2	10	6	201	201	(0.7 s)	152	(2.6 s)
apex6	135	99	1033	983	(267.6 s)	612	(459.7 s)
apex7	49	37	814	728	(27.7 s)	340	(52.2 s)
b9	41	21	256	185	(8.6 s)	122	(11.5 s)
c8	28	18	156	95	(1.7 s)	70	(3.2 s)
example2	85	66	491	484	(69.2 s)	416	(119.4 s)
mux	21	1	34	29	(0.6 s)	29	(0.7 s)
pcler8	27	17	78	73	(1.9 s)	72	(3.3 s)
rd73	7	3	76	34	(0.3 s)	27	(0.4 s)
rd84	8	4	144	42	(0.7 s)	42	(0.7 s)
sao2	10	4	104	104	(0.4 s)	70	(0.8 s)
x4	94	71	829	633	(121.9 s)	485	(203.4 s)
z4ml	7	4	51	32	(0.2 s)	17	(0.3 s)
partmult3	9	6	152	35	(1.0 s)	29	(1.2 s)
partmult4	16	8	971	222	(49.5 s)	114	(50.6 s)
partmult5	25	10	4574	998	(1540.4 s)	365	(1548.4 s)
total			10139	4969		3040	

Tabelle 3.2: Experimentelle Resultate. Die Tabelle zeigt die Knotenanzahlen der ROBDDs zu den einzelnen Funktionen. Die Zahlen in Klammern sind die benötigten CPU–Zeiten.

Darstellung von f in der Form

$$f(x_1,\ldots,x_n) = g(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_r(x_1,\ldots,x_p),x_h,\ldots,x_n).$$

Definition 3.13 (Zweiseitige nichtdisjunkte Zerlegung)
Eine **zweiseitige nichtdisjunkte Zerlegung** *einer Booleschen Funktion $f \in B_n$ mit den Eingangsvariablen $x_1,\ldots,x_n$ hinsichtlich des Paares $(X^{(1)},X^{(2)})$ von Variablenteilmengen mit $X^{(1)} = \{x_1,\ \ldots,x_p\}$, $X^{(2)} = \{x_h,\ \ldots,x_n\}$, $1 < p < n$ und $1 < h < p+1$ ist eine Darstellung von f in der Form*

$$\begin{aligned} &f(x_1,\ldots,x_n) = \\ &\quad g(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_r(x_1,\ldots,x_p),\beta_1(x_h,\ldots,x_n),\ldots,\beta_s(x_h,\ldots,x_n)) \end{aligned}$$

Anhand von Beispiel 3.20 soll demonstriert werden, wie man durch Verwendung nichtdisjunkter Zerlegungen Zerlegungsfunktionen einsparen kann:

Beispiel 3.20 Betrachte folgende Boolesche Funktion: $sel\_add_{2^k} : \{0,1\}^{2^{k+2}+1} \rightarrow \{0,1\}^{2^k}$, $sel\_add_{2^k}(os, a_{2^k-1}, \ldots, a_0, b_{2^k-1}, \ldots, b_0, c_{2^k-1}, \ldots, c_0, d_{2^k-1}, \ldots, d_0) = (r_{2^k-1}, \ldots, r_0)$, wobei

$$\sum_{i=0}^{2^k-1} r_i 2^i = \begin{cases} (\sum_{i=0}^{2^k-1} a_i 2^i + \sum_{i=0}^{2^k-1} b_i 2^i) \bmod 2^{2^k}, & falls\ os = 0 \\ (\sum_{i=0}^{2^k-1} c_i 2^i + \sum_{i=0}^{2^k-1} d_i 2^i) \bmod 2^{2^k}, & falls\ os = 1 \end{cases}$$

Die einzelnen Ausgangsfunktionen von $sel\_add_{2^k}$ werden mit $s_{2^k-1}, \ldots, s_0$ bezeichnet, also $sel\_add_{2^k} = (s_{2^k-1}, \ldots, s_0)$. Es handelt sich bei $sel\_add_{2^k}$ um die binäre Addition zweier 2^k–Bit–Zahlen. Der Eingang os bestimmt, welche beiden Zahlen addiert werden sollen: Ist $os = 0$, so werden die ersten beiden Operanden addiert ($(a_{2^k-1}, \ldots, a_0)$ und $(b_{2^k-1}, \ldots, b_0)$), ist $os = 1$, so werden die letzten beiden Operanden addiert ($(c_{2^k-1}, \ldots, c_0)$ und $(d_{2^k-1}, \ldots, d_0)$).

Abbildung 3.30 zeigt eine mögliche zweiseitige gleichmächtige und disjunkte Zerlegung aller s_i $(0 \leq i \leq 2^k - 1)$ hinsichtlich der Aufteilung $A = \{X^{(1)}, X^{(2)}\}^*$ mit

$$X^{(1)} = \{os, a_{2^k-1}, \ldots, a_{2^{k-1}}, b_{2^k-1}, \ldots, b_{2^{k-1}}, c_{2^k-1}, \ldots, c_{2^{k-1}}, d_{2^k-1}, \ldots, d_{2^{k-1}}\}$$

und

$$X^{(2)} = \{a_{2^{k-1}-1}, \ldots, a_0, b_{2^{k-1}-1}, \ldots, b_0, c_{2^{k-1}-1}, \ldots, c_0, d_{2^{k-1}-1}, \ldots, d_0\}.$$

Man erkennt, daß man bei dieser disjunkten Zerlegung für die Funktionen $s_0, \ldots, s_{2^{k-1}-1}$ jeweils 2 Zerlegungsfunktionen benötigt, die von $X^{(2)}$ abhängen, da der Wert von os bei der Berechnung der Zerlegungsfunktionen noch nicht bekannt ist. Es wird sowohl das entsprechende Summenbit der Addition von $(a_{2^{k-1}-1}, \ldots, a_0)$ und $(b_{2^{k-1}-1}, \ldots, b_0)$ als auch das Summenbit der Addition von $(c_{2^{k-1}-1}, \ldots, c_0)$ und $(d_{2^{k-1}-1}, \ldots, d_0)$ berechnet ($slab_i$ und $slcd_i$). Analog wird für die Funktionen $s_{2^{k-1}}, \ldots, s_{2^k-1}$ auf $X^{(2)}$ sowohl das Carrybit der Addition von $(a_{2^{k-1}-1}, \ldots, a_0)$ und $(b_{2^{k-1}-1}, \ldots, b_0)$ als auch das Carrybit der Addition von $(c_{2^{k-1}-1}, \ldots, c_0)$ und $(d_{2^{k-1}-1}, \ldots, d_0)$ berechnet ($carryab$ und $carrycd$).
Für die Zerlegung von $s_0, \ldots, s_{2^{k-1}-1}$ wird genau eine Zerlegungsfunktion benötigt, die von $X^{(1)}$ abhängt, nämlich os. Für die Zerlegung von $s_{2^{k-1}+2}, \ldots, s_{2^k-1}$ dagegen werden 3 Zerlegungsfunktionen benötigt, die von den Variablen aus $X^{(1)}$ abhängen: Das entsprechende Bit sh_i der Summe von $(a_{2^k-1}, \ldots, a_{2^{k-1}})$ und $(b_{2^k-1}, \ldots, b_{2^{k-1}})$ bzw. $(c_{2^k-1}, \ldots, c_{2^{k-1}})$ und $(d_{2^k-1}, \ldots, d_{2^{k-1}})$ (je nach Belegung von os) und das Bit $shp1_i$ der Summe plus 1 (für den Fall, daß die Addition der niederwertigen Bits der Operanden einen Übertrag erzeugt) und zusätzlich noch os, da in Abhängigkeit von os entweder $carryab$ oder $carrycd$ von der Addition der niederwertigen Bits ausgewählt werden muß[2].

Plaziert man jedoch die Variable os in *beide* Variablenmengen der Zerlegung, d.h. zerlegt man *nichtdisjunkt* hinsichtlich $X^{(1)}$ und $X^{(2)}$ mit

$$X^{(1)} = \{os, a_{2^k-1}, \ldots, a_{2^{k-1}}, b_{2^k-1}, \ldots, b_{2^{k-1}}, c_{2^k-1}, \ldots, c_{2^{k-1}}, d_{2^k-1}, \ldots, d_{2^{k-1}}\}$$

*Eine Realisierung der Funktion add_{2^k} (Addition zweier Operanden) ist in Kapitel 4 genauer beschrieben. Sie weist starke Parallelen zu der hier gezeigten Realisierung auf.

[2]Die Zerlegung von $s_{2^{k-1}}$ und $s_{2^{k-1}+1}$ stellt einen Sonderfall dar, da man außer os für $s_{2^{k-1}}$ nur noch eine Zerlegungsfunktion auf $X^{(1)}$ benötigt, die man auch in der Zerlegung von $s_{2^{k-1}+1}$ verwenden kann (vergleiche add_{2^k} aus Kapitel 4).

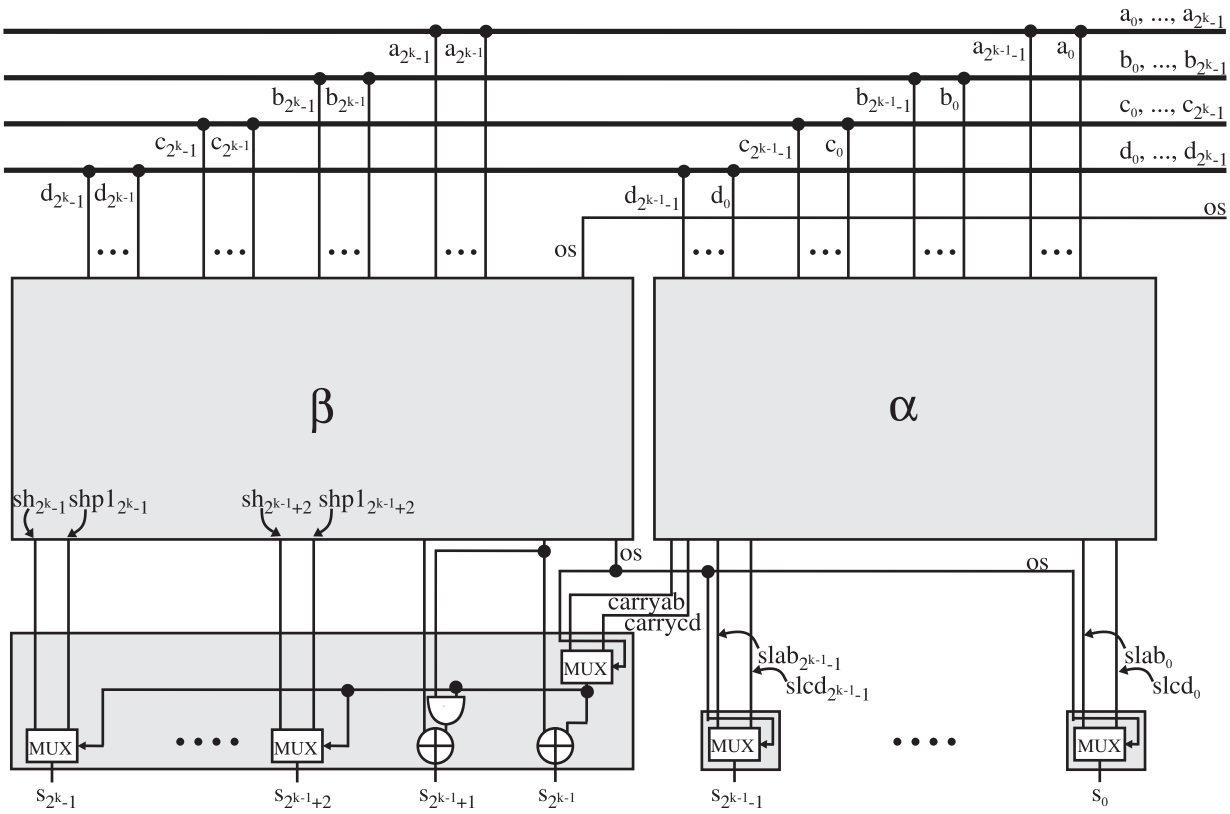


Abbildung 3.30: Disjunkte Zerlegung von $sel\_add_{2^k}$.

und

$$X^{(2)} = \{os, a_{2^{k-1}-1}, \ldots, a_0, b_{2^{k-1}-1}, \ldots, b_0, c_{2^{k-1}-1}, \ldots, c_0, d_{2^{k-1}-1}, \ldots, d_0\},$$

so kann man Zerlegungsfunktionen einsparen (siehe Abbildung 3.31).
Da man bei der Zerlegung von $s_0, \ldots, s_{2^{k-1}-1}$ hinsichtlich $X^{(2)}$ nun os schon kennt, genügt hier jeweils eine Zerlegungsfunktion, nämlich das entsprechende Ergebnisbit s_i.
Für die Funktionen $s_{2^{k-1}}, \ldots, s_{2^k-1}$ muß auf $X^{(2)}$ dann nur ein einziges Carrybit berechnet werden, da os in $X^{(2)}$ schon enthalten ist und daher bei der Berechnung auf den Variablen aus $X^{(2)}$ schon bekannt ist, ob die Operanden $(a_{2^{k-1}-1}, \ldots, a_0)$ und $(b_{2^{k-1}-1}, \ldots, b_0)$ oder $(c_{2^{k-1}-1}, \ldots, c_0)$ und $(d_{2^{k-1}-1}, \ldots, d_0)$ addiert werden müssen.
Ebenso entfällt dann bei der Zerlegung hinsichtlich $X^{(1)}$ für alle Ausgangsfunktionen os als Zerlegungsfunktion.

Bemerkung 3.9 *Seien $X^{(1)}$ und $X^{(2)}$ zwei disjunkte Variablenmengen mit $|X^{(2)}| = |X^{(1)}|$, c eine zusätzliche Variable $\notin X^{(1)} \cup X^{(2)}$. Dann kann man leicht Boolesche Funktionen $f \in B_n$ $(n = |X^{(1)}| + |X^{(2)}| + 1)$ mit Variablenmenge $X^{(1)} \cup X^{(2)} \cup \{c\}$ konstruieren, für die gilt:*

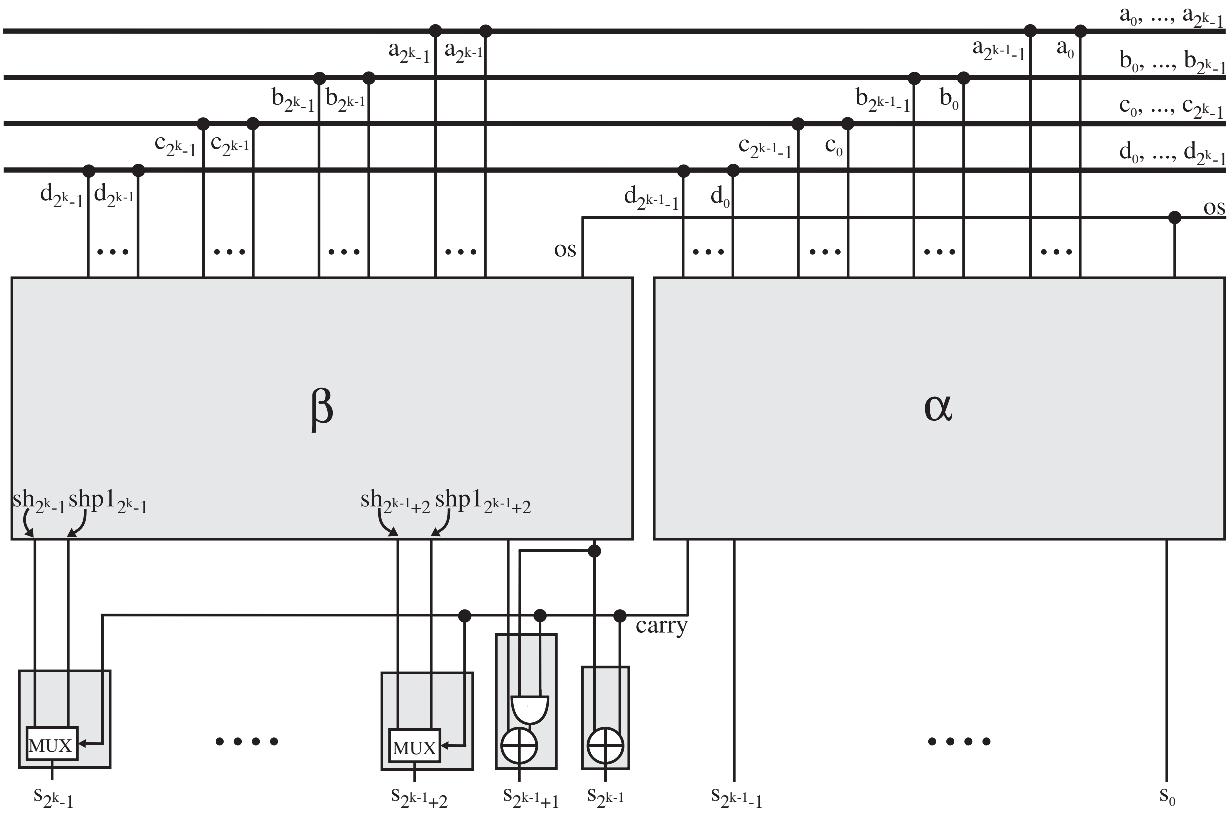


Abbildung 3.31: Nichtdisjunkte Zerlegung von $sel\_add_{2^k}$.

Bei einer disjunkten Zerlegung von f hinsichtlich $\{X^{(1)}, X^{(2)} \cup \{c\}\}$ benötigt man $|X^{(1)}|$ Zerlegungsfunktionen, die von $X^{(1)}$ abhängen, und $\frac{1}{2}|X^{(2)}|+1$ Zerlegungsfunktionen, die von $X^{(2)} \cup \{c\}$ abhängen. Bei einer disjunkten Zerlegung von f hinsichtlich $\{X^{(1)} \cup \{c\}, X^{(2)}\}$ benötigt man $|X^{(2)}|$ Zerlegungsfunktionen, die von $X^{(2)}$ abhängen, und $\frac{1}{2}|X^{(1)}| + 1$ Zerlegungsfunktionen, die von $X^{(1)} \cup \{c\}$ abhängen.
Bei einer zweiseitigen nichtdisjunkten Zerlegung von f hinsichtlich $X^{(1)} \cup \{c\}$ und $X^{(2)} \cup \{c\}$ benötigt man hingegen insgesamt nur $\frac{1}{2}|X^{(1)}| + \frac{1}{2}|X^{(2)}|$ Zerlegungsfunktionen.
Intuitiv kann man dies so interpretieren, daß c für die Auswertung von Zwischenergebnissen sowohl auf $X^{(1)}$ als auch auf $X^{(2)}$ „wesentlich“ ist: Entfernt man bei der nichtdisjunkten Zerlegung c aus $X^{(1)} \cup \{c\}$ bzw. aus $X^{(2)} \cup \{c\}$, so wird die Anzahl der Zerlegungsfunktionen, die auf $X^{(1)}$ bzw. $X^{(2)}$ berechnet werden muß, verdoppelt. Man muß sowohl für den Fall $c = 0$ als auch für den Fall $c = 1$ (getrennte) Zwischenergebnisse auf $X^{(1)}$ bzw. $X^{(2)}$ berechnen.

Bemerkung 3.9 zeigt, daß es Fälle geben kann, in denen es durchaus sinnvoll ist, nichtdisjunkt zu zerlegen.

Nichtdisjunkte Zerlegungen lassen sich leicht in das bisherige Schema zur Zerlegung Boolescher Funktionen integrieren:

- 1. Alternative:
  Auch bei der Berechnung gemeinsamer Zerlegungsfunktionen wurden für eine Ausgangsfunktion nicht alle Zerlegungsfunktionen gleichzeitig berechnet, sondern die Zerlegungsfunktionen wurden nach und nach unter Voraussetzung schon gegebener Zerlegungsfunktionen bestimmt. Einseitige nichtdisjunkte Zerlegungsfunktionen lassen sich darstellen als disjunkte Zerlegungen mit fester Vorgabe bestimmter Zerlegungsfunktionen. Eine einseitige nichtdisjunkte Zerlegung hinsichtlich des Paares $(X^{(1)}, X^{(2)})$ von Variablenteilmengen mit $X^{(1)} = \{x_1, \ldots, x_p\}$, $X^{(2)} = \{x_h, \ldots, x_n\}$, $0 < p < n$ und $1 < h < p+1$ läßt sich darstellen als einseitige disjunkte Zerlegung hinsichtlich $X^{(1)}$, bei der die Zerlegungsfunktionen $x_h, \ldots, x_p$ vorgegeben sind (vergleiche Abbildung 3.32).

- 2. Alternative:
  Eine andere Möglichkeit zur Integration nichtdisjunkter Zerlegungen ist die Ausnutzung von Zerlegungen partieller Funktionen. Hierbei werden bei der Zerlegung einer Funktion $f \in B_n$ hinsichtlich des Paares $(X^{(1)}, X^{(2)})$ mit $X^{(1)} = \{x_1, \ldots, x_p\}$ und $X^{(2)} = \{x_h, \ldots, x_n\}$ $(1 < p < n$ und $1 < h < p+1)$ die Eingangsvariablen, die sowohl in $X^{(1)}$ als auch in $X^{(2)}$ vorkommen, „verdoppelt“: Man führt zusätzlich zu $x_h, \ldots, x_p$ Variablen $x'_h, \ldots, x'_p$ ein und definiert eine Funktion f' durch

  $$f'(\epsilon_1, \ldots, \epsilon_{h-1}, \epsilon_h, \epsilon'_h, \ldots, \epsilon_p, \epsilon'_p, \epsilon_{p+1}, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_{h-1}, \epsilon_h, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_n)$$

  für alle $(\epsilon_1, \ldots, \epsilon_{h-1}, \epsilon_h, \epsilon'_h, \ldots, \epsilon_p, \epsilon'_p, \epsilon_{p+1}, \ldots, \epsilon_n) \in \{0,1\}^{n+p-h+1}$ mit $\epsilon_h = \epsilon'_h$, $\ldots$, $\epsilon_p = \epsilon'_p$. f' ist eine partielle Funktion, wobei die charakteristische Funktion der don't care–Menge durch

  $$dc(x_h, x'_h, \ldots, x_p, x'_p) = \bigvee_{j=h}^{p} x_j \oplus x'_j$$

  gegeben ist. Es ist klar, daß eine kommunikationsminimale zweiseitige disjunkte Zerlegung dieser partiellen Funktion f' hinsichtlich $\{Y^{(1)}, Y^{(2)}\}$ mit $Y^{(1)} = \{x_1, \ldots, x_p\}$ und $Y^{(2)} = \{x'_h, \ldots, x'_p, x_{p+1}, \ldots, x_n\}$ genau einer kommunikationsminimalen zweiseitigen nichtdisjunkten Zerlegung von f hinsichtlich $(X^{(1)}, X^{(2)})$ entspricht und daß eine kommunikationsminimale einseitige disjunkte Zerlegung von f' hinsichtlich $X^{(1)}$ einer kommunikationsminimalen einseitigen nichtdisjunkten Zerlegung von f hinsichtlich $(X^{(1)}, X^{(2)})$ entspricht (vergleiche Abbildung 3.33).

Es bleibt noch die Frage, wie man entscheidet, ob eine Variable c in beiden Mengen $X^{(1)}$ und $X^{(2)}$ vorkommen soll.

Ist $\{X^{(1)}, X^{(2)} \cup \{c\}\}$ eine Partition der Eingangsvariablen von f, so geht aus der 1. Alternative der obigen Aufzählung hervor, daß man die Anzahl der Zerlegungsfunktionen von f bei einer einseitigen nichtdisjunkten Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ durch

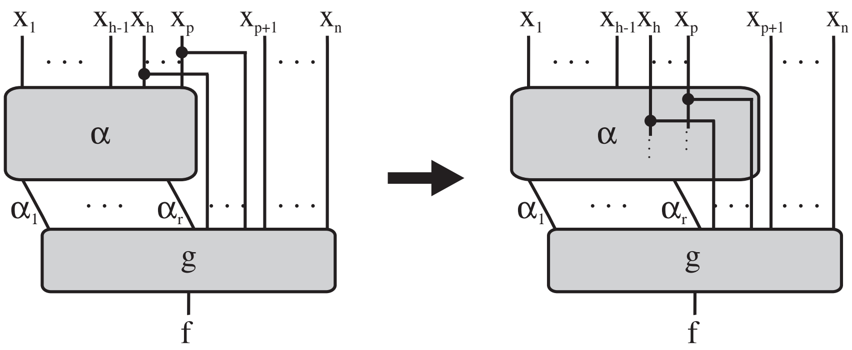


Abbildung 3.32: Interpretation einer einseitigen nichtdisjunkten Zerlegung als disjunkte Zerlegung mit vorgegebenen Zerlegungsfunktionen.

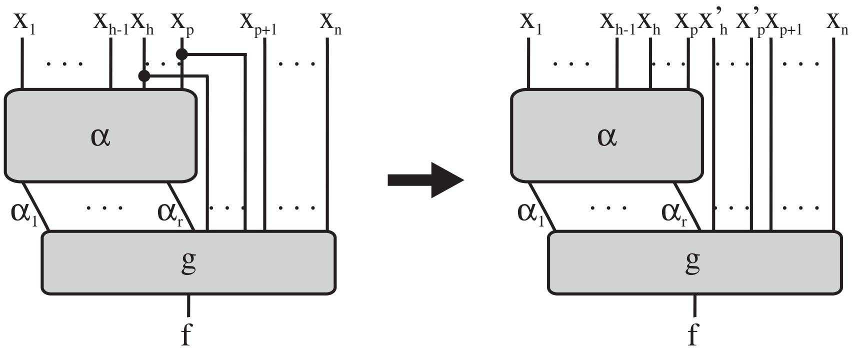


Abbildung 3.33: Durchführung einer nichtdisjunkten Zerlegung durch disjunkte Zerlegung mit Variablenverdopplung und don't care–Ausnutzung.

Berechnung von $\lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil$[3] erhält. $vz(X^{(1)} \cup \{c\}, f, c)$ läßt sich leicht mit Hilfe der Kofaktoren f_c und $f_{\overline{c}}$ bestimmen[4].

Lemma 3.14 *Sei $f \in B_n$, $\{X^{(1)}, X^{(2)} \cup \{c\}\}$ eine Partition der Eingangsvariablen von f. Dann gilt:*

$$vz(X^{(1)} \cup \{c\}, f, c) = \max(vz(X^{(1)}, f_{\overline{c}}), vz(X^{(1)}, f_c))$$

Beweis: Sei $X^{(1)} = \{x_1, \ldots, x_p\}$. Sei $Z(X^{(1)} \cup \{c\})$ die Zerlegungsmatrix von f hinsichtlich $X^{(1)} \cup \{c\}$. Dann ist nach Definition 3.10 $vz(X^{(1)} \cup \{c\}, f, 0)$ folgendermaßen bestimmt: Betrachte die Zeilen in $Z(X^{(1)} \cup \{c\})$, denen „die Funktion c den Wert 0 zuordnet", d.h. die Zeilen, für deren Index $(x_1, \ldots, x_p, c)$ $c = 0$ gilt. $vz(X^{(1)} \cup \{c\}, f, 0)$ ist dann die Anzahl der verschiedenen Zeilenmuster unter diesen Zeilen. Es ist klar daß diese betrachteten Zeilen genau mit den Zeilen der Zerlegungsmatrix $Z'(X^{(1)})$ des Kofaktors $f_{\overline{c}}$ übereinstimmen, so daß

$$vz(X^{(1)} \cup \{c\}, f, 0) = vz(X^{(1)}, f_{\overline{c}}).$$

Die Aussage des Lemmas ergibt sich aus der Tatsache, daß

$$vz(X^{(1)} \cup \{c\}, f, c) = \max(vz(X^{(1)} \cup \{c\}, f, 0), vz(X^{(1)} \cup \{c\}, f, 1)).$$

□

Allerdings kann die Berechnung von $\lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil$ noch keinen Hinweis darauf liefern, ob man c evtl. in beide Variablenteilmengen der Zerlegung aufnehmen sollte:

Bemerkung 3.10 *Man überlegt sich leicht, daß*

$$\lceil \log(vz(X^{(1)} \cup \{c\}, f)) \rceil - 1 \leq \lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil \leq \lceil \log(vz(X^{(1)} \cup \{c\}, f)) \rceil.$$

Folglich kann die Anzahl der Eingänge der Zusammensetzungsfunktion bei einer einseitigen nichtdisjunkten Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ nicht geringer sein als die Anzahl der Eingänge der Zusammensetzungsfunktion bei einer einseitigen disjunkten Zerlegung hinsichtlich $X^{(1)} \cup \{c\}$. Wählt man also einseitige Zerlegungen, so können nichtdisjunkte Zerlegungen lediglich im Hinblick auf die weitere (rekursive) Zerlegung der Zusammensetzungsfunktion einen Sinn machen.

Es ist dann sinnvoll, eine Variable c in beide Variablenteilmengen der Zerlegung aufzunehmen, wenn c „für die Auswertung von Zwischenergebnissen auf beiden Variablenteilmengen wesentlich ist". Daher bietet es sich an, auch bei der Überprüfung, ob eine einseitige nichtdisjunkte Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ sinnvoll ist, zu testen, ob *die*

[3] $vz(X^{(1)} \cup \{c\}, f, c) = \max(vz(X^{(1)} \cup \{c\}, f, 0), vz(X^{(1)} \cup \{c\}, f, 1))$ ist definiert durch Definition 3.10 (Seite 149).

[4] Der Einfachheit halber beschränkt sich die Darstellung hier auf die Betrachtung einer einzigen gemeinsamen Variable c der beiden Mengen der Zerlegung. Die Aussagen lassen sich problemlos auf mehrere gemeinsame Variablen ausdehnen.

entsprechende zweiseitige Zerlegung sinnvoll ist. Dazu wird einfach die Anzahl der Zerlegungsfunktionen bei einer zweiseitigen Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ verglichen mit der Anzahl der Zerlegungsfunktionen bei der zweiseitigen Zerlegung hinsichtlich $\{X^{(1)} \cup \{c\}, X^{(2)}\}$ bzw. $\{X^{(1)}, X^{(2)} \cup \{c\}\}$.

Da die exakte Bestimmung der Anzahl der benötigten Zerlegungsfunktionen bei einer nichtdisjunkten zweiseitigen Zerlegung einen gewissen Aufwand erfordert (Lösung des Problems ZKM aus Abschnitt 3.2 für die in der 2. Alternative (s.o.) konstruierte partielle Funktion), behilft man sich zum Test, ob die zweiseitige Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ sinnvoll ist, mit einer glücklicherweise recht genauen und einfach zu berechnenden Näherung für die Anzahl der Zerlegungsfunktionen[5]:

Lemma 3.15 *Sei $f \in B_n$, $\{X^{(1)}, X^{(2)} \cup \{c\}\}$ eine Partition der Eingangsvariablen von f. Dann gilt für die minimale Anzahl zf der Zerlegungsfunktionen einer zweiseitigen nichtdisjunkten Zerlegung hinsichtlich* $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$

$$\begin{aligned} zf &\geq \lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil + \lceil \log(vz(X^{(2)} \cup \{c\}, f, c)) \rceil \ \textit{und} \\ zf &\leq \lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil + \lceil \log(vz(X^{(2)} \cup \{c\}, f, c)) \rceil + 1. \end{aligned}$$

Beweis: (Skizze)
Wie weiter oben schon festgestellt wurde, stimmt die minimale Anzahl der Zerlegungsfunktionen einer zweiseitigen Zerlegung von f hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c\})$ überein mit der minimalen Anzahl von Zerlegungsfunktionen in der zweiseitigen *disjunkten* Zerlegung einer partiellen Funktion f' hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c'\})$ mit

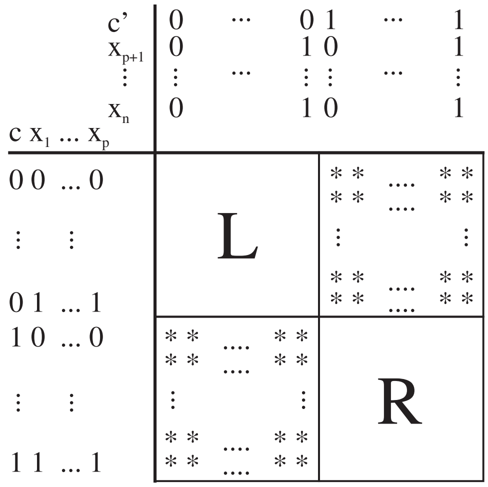
$$f'(\epsilon_c, \epsilon_1, \ldots, \epsilon_p, \epsilon_{c'}, \epsilon_{p+1}, \ldots, \epsilon_n) = f(\epsilon_c, \epsilon_1, \ldots, \epsilon_p, \epsilon_{p+1}, \ldots, \epsilon_n)$$

für alle $(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$, $\epsilon_c = \epsilon_{c'} \in \{0,1\}$.
$(X^{(1)} = \{x_1, \ldots, x_p\}$, $X^{(2)} = \{x_{p+1}, \ldots, x_n\}$, $c' \notin X^{(1)} \cup X^{(2)} \cup \{c\}$.)

$$DC(f') = \{(\epsilon_c, \epsilon_1, \ldots, \epsilon_p, \epsilon_{c'}, \epsilon_{p+1}, \ldots, \epsilon_n) \in \{0,1\}^{n+2} \mid \epsilon_c \neq \epsilon_{c'}\}.$$

Die Zerlegungsmatrix von f' hat folgendes Aussehen:

[5] Lemma 3.15 läßt sich leicht verallgemeinern auf mehrere gemeinsame Variablen in den beiden Variablenmengen der Zerlegung.



Z.z.: $zf \leq \lceil \log(vz(X^{(1)} \cup \{c\}, f, c))\rceil + \lceil \log(vz(X^{(2)} \cup \{c\}, f, c))\rceil + 1$.
Es wird eine Zerlegung von f' (bzw. einer beliebigen totalen Erweiterung von f') konstruiert mit $\lceil \log(vz(X^{(1)} \cup \{c\}, f, c))\rceil$ Zerlegungsfunktionen, die von $X^{(1)} \cup \{c\}$ abhängen und $\lceil \log(vz(X^{(2)} \cup \{c\}, f, c))\rceil + 1$ Zerlegungsfunktionen, die von $X^{(2)} \cup \{c'\}$ abhängen.
Dazu wird eine Kodierung der Zeilen der obigen Matrix gewählt, die Zeilen mit verschiedenem Zeilenmuster verschiedene Codewerte zuordnet. Aus der speziellen Anordnung der don't care–Stellen ($*$ in der Matrix) ergibt sich, daß die Anzahl der Codewerte, die man dazu benötigt, gerade dem Maximum aus der Anzahl der verschiedenen Zeilenmuster in L und der Anzahl der verschiedenen Zeilenmuster in R entspricht. Dieser Wert ist aber gleich $vz(X^{(1)} \cup \{c\}, f, c)$. Man benötigt $\lceil \log(vz(X^{(1)} \cup \{c\}, f, c))\rceil$ verschiedene Zerlegungsfunktionen, die diese Codewerte liefern.
Nun werden don't cares so belegt, daß Zeilen exakt gleich werden, die durch L und R verlaufen und denen der gleiche Code zugeordnet wird. Evtl. noch verbleibende don't cares werden z.B. mit 0 belegt. Eine wesentliche Beobachtung bei der Belegung der don't cares ist die Tatsache, daß Paare von Spalten, die beide durch L (bzw. beide durch R) verlaufen und vor don't care–Belegung gleich waren, auch nach der don't care–Belegung gleich bleiben.
Will man nun eine Kodierung der Spalten konstruieren, die Spalten aus L mit verschiedenem Spaltenmuster verschiedene Werte zuordnet und Spalten aus R mit verschiedenem Spaltenmuster ebenfalls verschiedene Werte zuordnet, so ist daher die Anzahl der verschiedenen Codes, die man dazu braucht, gerade das Maximum der verschiedenen Spaltenmuster in L und in R, also $vz(X^{(2)} \cup \{c\}, f, c)$. Hierzu sind $\lceil \log(vz(X^{(2)} \cup \{c\}, f, c))\rceil$ verschiedene Zerlegungsfunktionen nötig.
Um zu erreichen, daß insgesamt Spalten mit verschiedenem Spaltenmuster verschiedene Codewerte zugeordnet werden, braucht man noch höchstens eine Zerlegungsfunktion hinzuzufügen: die Funktion c', die den Spalten, die durch L verlaufen 0 zuordnet und den Spalten, die durch R verlaufen, 1 zuordnet.
Insgesamt wurde so eine totale Erweiterung von f' konstruiert, deren Zerlegung

$$\lceil \log(vz(X^{(1)} \cup \{c\}, f, c))\rceil + \lceil \log(vz(X^{(2)} \cup \{c\}, f, c))\rceil + 1$$

verschiedene Zerlegungsfunktionen aufweist.

Die untere Schranke

$$\lceil \log(vz(X^{(1)} \cup \{c\}, f, c)) \rceil + \lceil \log(vz(X^{(2)} \cup \{c\}, f, c)) \rceil$$

für die Anzahl der Zerlegungsfunktionen von f' ergibt sich einfach aus der Tatsache, daß bei der partiellen Funktion f'

$$\lceil \log(vkk(X^{(1)} \cup \{c\}, f')) \rceil + \lceil \log(vkk(X^{(2)} \cup \{c'\}, f')) \rceil$$

eine untere Schranke für die Anzahl der Zerlegungsfunktionen der zweiseitigen Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)} \cup \{c'\})$ ist und

$$vkk(X^{(1)} \cup \{c\}, f') = vz(X^{(1)} \cup \{c\}, f, c),$$

$$vkk(X^{(2)} \cup \{c'\}, f') = vz(X^{(2)} \cup \{c\}, f, c).$$

□

Anhand von Beispielen erkennt man, daß sowohl die Obergrenze als auch die Untergrenze in Lemma 3.15 scharf sind, d.h. daß es sowohl Boolesche Funktionen gibt, bei denen für die Anzahl der Zerlegungsfunktionen die Untergrenze angenommen wird, als auch Boolesche Funktionen, bei denen für die Anzahl der Zerlegungsfunktionen die Obergrenze angenommen wird.

Die Suche nach geeigneten nichtdisjunkten zweiseitigen Zerlegungen läßt sich dann einfach in die Suche nach guten disjunkten zweiseitigen Zerlegungen (vergleiche Abschnitte 3.1.4 und 3.4.1) integrieren:

Bei jeder Inputpartitionierung $A = \{X^{(1)}, X^{(2)}\}$, die man während des Verfahrens gefunden hat, bestimmt man für jede Eingangsvariable c gemäß der Obergrenze aus Lemma 3.15 die Anzahl der Zerlegungsfunktionen der nichtdisjunkten zweiseitigen Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X^{(2)})$ (für $c \in X^{(2)}$) bzw. hinsichtlich $(X^{(1)}, X^{(2)} \cup \{c\})$ (für $c \in X^{(1)}$). Auf dieser Grundlage berechnet man jeweils die geschätzten Kosten der betreffenden nichtdisjunkten Zerlegung und vergleicht sie mit mit den geschätzten Kosten bei der disjunkten Zerlegung $A = \{X^{(1)}, X^{(2)}\}$. Ausgewählt wird die Zerlegung mit den geringsten geschätzten Kosten. Ist dies eine nichtdisjunkte Zerlegung, so wird weiter getestet, ob man die Kosten durch Aufnahme einer weiteren Variable in beide Variablenteilmengen der Zerlegung noch weiter verringern kann.

Sucht man nach nichtdisjunkten einseitigen Zerlegungen, so geht man ähnlich vor:

Sei die Menge sämtlicher Variablen der zu zerlegenden Funktion mit X bezeichnet und das Verfahren zur Bestimmung einer Variablenteilmenge zur Zerlegung habe eine Variablenteilmenge $X^{(1)}$ gefunden. Dann wird für jede Variable c aus $X \setminus X^{(1)}$ die Anzahl der Zerlegungsfunktionen der *zweiseitigen* Zerlegung hinsichtlich $(X^{(1)} \cup \{c\}, X \setminus X^{(1)})$ abgeschätzt (vergleiche Bemerkung 3.10). Die geschätzten Kosten werden verglichen mit den geschätzten Kosten der zweiseitigen disjunkten Zerlegung hinsichtlich $\{X^{(1)}, X \setminus X^{(1)}\}$. Ausgewählt wird die Zerlegung (disjunkt oder nichtdisjunkt) mit den geringsten geschätzten Kosten. Ansonsten erfolgt die Behandlung einseitiger Zerlegungen völlig analog zur Behandlung zweiseitiger Zerlegungen.

3.7 Testen

Bei der Fertigung eines korrekt entworfenen Schaltkreises können physikalische Defekte (z.B. Kurzschlüsse, Leitungsabrisse etc.) auftreten. Aufgabe eines Fertigungstests ist es, fehlerhaft gefertigte Schaltkreise auszusortieren. Dabei scheidet bei mittleren und größeren Schaltungen eine vollständige Überprüfung des Schaltkreises für alle Eingangsbelegungen aus Zeit– und Kostengründen aus. Aus diesem Grund beschränkt man sich darauf, (in Abhängigkeit von der gewählten Technologie) *„wahrscheinliche“* Fehler zu modellieren und einen Test für diese Fehler zu berechnen. Ein Test ist hierbei eine Eingabe an den Schaltkreis, die den Fehlereffekt nach außen, d.h. an den Ausgängen, sichtbar macht.

Ein weit verbreitetes Fehlermodell ist das (Einzel–)Stuck–at–Fehlermodell für kombinatorische Defekte. Stuck–at–Fehler modellieren Defekte, die sich so äußern, daß ein Eingang oder Ausgang einer Grundzelle des Schaltkreises konstant auf einem logischen Wert (0 oder 1) liegt. (Aus diesem Grund ist das Stuck–at–Fehlermodell nur sinnvoll, wenn die Grundzellen möglichst klein gewählt sind. Als Alternative dazu bietet sich das mächtigere Zellenfehlermodell an: Hierbei wird angenommen, daß Grundzellen ein beliebiges kombinatorisches Fehlverhalten zeigen können, d.h. daß ein Fehler die Boolesche Funktion, die eine Grundzelle realisiert, beliebig abändern kann.) Das (Einzel–)Stuck–at–Fehlermodell geht weiterhin davon aus, daß im gesamten Schaltkreis nur ein einziger „Stuck–at–Fehler“ vorhanden ist, d.h. daß nur ein einziger Eingang/Ausgang einer Grundzelle konstant auf 0 oder 1 liegt. In der Regel werden nur solche „Einzelfehler“ betrachtet, da

- die Betrachtung von Mehrfachfehlern wegen ihrer enormen Zahl zu aufwendig ist, und
- der größte Teil der Mehrfachfehler schon durch Tests für Einzelfehler geprüft wird [AF81]. Defekte, die nicht lokal begrenzt sind, werden mit hoher Wahrscheinlichkeit durch Anlegen weniger zufälliger Muster direkt erkannt.

Im Rahmen der vorliegenden Arbeit soll auf eine formalere Einführung des Fehlermodells verzichtet werden. Genauere Definitionen können z.B. in [Sp91] nachgelesen werden.

In diesem Abschnitt soll gezeigt werden, wie man parallel zur Logiksynthese für eine Boolesche Funktion schon gleich einen vollständigen Test bzgl. des (Einzel–)Stuck-at-Fehlermodells mitberechnen kann. Die Betrachtung erfolgt hier für das (Einzel–)Stuck-at-Fehlermodell, man sieht aber leicht, daß die Resultate auch für das mächtigere Zellenfehlermodell gelten. Der Begriff des „vollständigen Tests“ ist hierbei folgendermaßen definiert:

Definition 3.14 *Sei F ein Fehler bzgl. eines gegebenen Fehlermodells $\mathcal{F}$. Sei S ein Schaltkreis und S_F der zugehörige Schaltkreis mit Fehler F.*

1. *Ein* Test für einen Fehler *F ist eine Eingabe, für die S und S_F verschiedene Ausgaben berechnen.*
2. *Der Fehler F heißt* redundant*, falls es keinen Test für F gibt.*
3. *Sei M eine Menge von Eingaben an den Schaltkreis.*
   M heißt vollständiger Test *von S bzgl. des Fehlermodells $\mathcal{F}$, falls M für jeden nicht redundanten Fehler in S (bzgl. des Fehlermodells $\mathcal{F}$) einen Test enthält.*

3.7.1 Berechnung eines vollständigen Tests

3.7.1.1 Freiheitsrelationen

Ein wesentlicher Begriff bei der Berechnung eines vollständigen Tests während des rekursiven Zerlegungsverfahrens zur Realisierung einer Booleschen Funktion f ist die Freiheitsrelation einer Teilschaltung der Gesamtschaltung für f. Eine Freiheitsrelation ist eine Boolesche Relation im Sinn von Cerny [CM77]:

Definition 3.15 (Boolesche Relation) *Eine Relation $F \subseteq \{0,1\}^n \times \{0,1\}^m$ heißt* Boolesche Relation *mit n Eingängen und m Ausgängen.*

Definition 3.16 (Vollständige Boolesche Relation)
Eine Boolesche Relation $F \subseteq \{0,1\}^n \times \{0,1\}^m$ heißt vollständig*, wenn es zu jedem $\epsilon \in \{0,1\}^n$ mindestens ein $\delta \in \{0,1\}^m$ gibt mit $(\epsilon, \delta) \in F$.*

Bezeichnung 3.11 *Sei $F \subseteq \{0,1\}^n \times \{0,1\}^m$ eine vollständige Boolesche Relation.*

1. *F heißt* totale Boolesche Funktion*, falls es zu jedem $\epsilon \in \{0,1\}^n$* genau ein *$\delta \in \{0,1\}^m$ gibt mit $(\epsilon, \delta) \in F$. F wird dann identifiziert mit $f \in B_{n,m}$ mit*
   $$f(\epsilon) = \delta \iff (\epsilon, \delta) \in F (\forall\, \epsilon \in \{0,1\}^n).$$
   F wird dann auch mit $R(f)$ bezeichnet.
2. *Ist $F' \subseteq F$ eine totale Boolesche Funktion, so sagt man auch: Die totale Funktion F' „liegt in der Booleschen Relation F“.*
3. *F heißt* partielle Boolesche Funktion *mit Definitionsmenge $D \subseteq \{0,1\}^n$, falls es für alle $\epsilon \in D$ genau ein $\delta \in \{0,1\}^m$ gibt mit $(\epsilon, \delta) \in F$ und falls für alle $\epsilon \in \{0,1\}^n \setminus D$ gilt: $(\epsilon, \delta) \in F$ für alle $\delta \in \{0,1\}^m$. F wird dann identifiziert mit $f \in BP_{n,m}$, $f : D \to \{0,1\}^m$ mit*
   $$\forall\, \epsilon \in D: \; f(\epsilon) = \delta \iff (\epsilon, \delta) \in F.$$

Bemerkung 3.11 *Alle vollständigen Booleschen Relationen* mit einem Ausgang *stellen partielle Boolesche Funktionen dar. Boolesche Relationen mit mehreren Ausgängen lassen sich im allgemeinen allerdings nicht als partielle Boolesche Funktionen beschreiben.*

Definition 3.17 (Freiheitsrelation) *Sei S ein Schaltkreis und T ein Teilschaltkreis von S mit n_T Eingängen und m_T Ausgängen. Sei FR_T die* maximale *(vollständige) Boolesche Relation mit n_T Eingängen und m_T Ausgängen, so daß für alle totalen Booleschen Funktionen f, die in der Relation FR_T liegen, gilt:*

> *Ersetzt man in S den Teilschaltkreis T durch eine Realisierung T' von f, so ändert sich die durch S berechnete Boolesche Funktion nicht.*

Dann heißt FR_T die Freiheitsrelation *von T bzgl. S.*

Die Freiheitsrelation eines Teilschaltkreises T von S gibt also an, wie man die durch T realisierte Funktion abändern kann, ohne daß sich dadurch die durch den Gesamtschaltkreis S berechnete Funktion ändert. Man kann für T jeden beliebigen Schaltkreis einsetzen, der eine totale Funktion realisiert, die in der Freiheitsrelation liegt. Da die Freiheitsrelation maximal gewählt ist, wird bei Einsetzung anderer Funktionen (die nicht in der Freiheitsrelation liegen) aber auf jeden Fall die durch den Gesamtschaltkreis S realisierte Funktion verändert.

Somit ist auch der Zusammenhang zwischen der Freiheitsrelation des Teilschaltkreises T und redundanten Fehlern in T klar:

Lemma 3.16 *Sei S ein Schaltkreis und T ein Teilschaltkreis von S. Sei F ein Fehler in T und T_F der Schaltkreis, der entsteht, wenn man „den Fehler F in T einbaut“. Der Fehler F ist genau dann redundant (d.h. es gibt keine Belegung der Eingänge von S, die den Fehler an den Ausgängen sichtbar macht), wenn die durch T_F realisierte Funktion in der Freiheitsrelation von T bzgl. S liegt.*

Beweis: Falls die durch T_F realisierte Funktion f_{T_F} in der Freiheitsrelation von T bzgl. S liegt, ändert sich nach Definition bei Ersetzung von T durch T_F die durch S berechnete Funktion nicht. Folglich kann es keine Eingabe an S geben, die den Fehler an den Ausgängen sichtbar macht.
Falls f_{T_F} *nicht* in der Freiheitsrelation von T bzgl. S liegt, so ändert sich bei Ersetzung von T durch T_F die durch den entstandenen Schaltkreis S_F berechnete Funktion (FR_T ist maximal!). Also gibt es auch eine Eingabe an S bzw. S_F, so daß sich die Ausgaben von S und S_F unterscheiden; F ist *nicht* redundant. □

Bemerkung 3.12 *Sei S ein Schaltkreis, T ein Teilschaltkreis von S, $S \backslash T$ der Schaltkreis, der aus S hervorgeht, wenn man T entfernt. (Dabei werden die Gatter aus T in S entfernt.*

Eingänge von T werden zu zusätzlichen Ausgängen von $S \setminus T$ und Ausgänge von T zu zusätzlichen Eingängen von $S \setminus T$.) f_S, f_T bzw. $f_{S\setminus T}$ seien die durch S, T bzw. $S \setminus T$ realisierten Funktionen. Da für die Bestimmung der Freiheitsrelation FR_T lediglich die durch $S\setminus T$ berechnete Funktion $f_{S\setminus T}$ relevant ist, spricht man auch von der Freiheitsrelation FR_{f_T} von f_T bzgl. $f_{S\setminus T}$ statt von der Freiheitsrelation von T bzgl. S.

Boolesche Relationen lassen sich durch ROBDDs für ihre charakteristischen Funktionen darstellen. Zu jeder Booleschen Relation $F \subseteq \{0,1\}^n \times \{0,1\}^m$ gehört eine charakteristische Funktion χ_F mit $\chi_F(\epsilon, \delta) = 1$ genau dann, wenn $(\epsilon, \delta) \in F$ ($\epsilon \in \{0,1\}^n$, $\delta \in \{0,1\}^m$).
Seien den Ausgängen einer totalen Booleschen Funktion $f \in B_{n,m}$ mit den Eingangsvariablen $x_1, \ldots, x_n$ die Variablen $o_1, \ldots, o_m$ zugeordnet. Dann erhält man die charakteristische Funktion $\chi_{R(f)}$ zu $R(f)$ durch

$$\chi_{R(f)}(x_1, \ldots, x_n, o_1, \ldots, o_m) = \bigwedge_{i=1}^{m} \overline{o_i \oplus f_i(x_1, \ldots, x_n)}.$$

3.7.1.2 Rekursive Testberechnung

Es soll nun ein Verfahren angegeben werden, das parallel zur rekursiven Zerlegung einer totalen Funktion aus $B_{n,m}$ einen vollständigen Test berechnet. Dabei werden für die einzelnen Blöcke, für die rekursive Aufrufe erfolgen, jeweils die zugehörigen Freiheitsrelationen (immer bzgl. der Gesamtschaltung) berechnet und den entsprechenden rekursiven Aufrufen mitgegeben.

Auf der obersten Rekursionsstufe ist die Freiheitsrelation für f einfach durch $R(f)$ gegeben. Allerdings können im Verlauf des Verfahrens wesentlich komplexere Freiheitsrelationen entstehen: Angenommen, bei der Zerlegung der Funktion $f_i \in B_n$ der Form

$$f_i(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}) = g_i(\alpha_1^{(i)}(\mathbf{x^{(1)}}), \ldots, \alpha_{r_i}^{(i)}(\mathbf{x^{(1)}}), \mathbf{x^{(2)}})$$

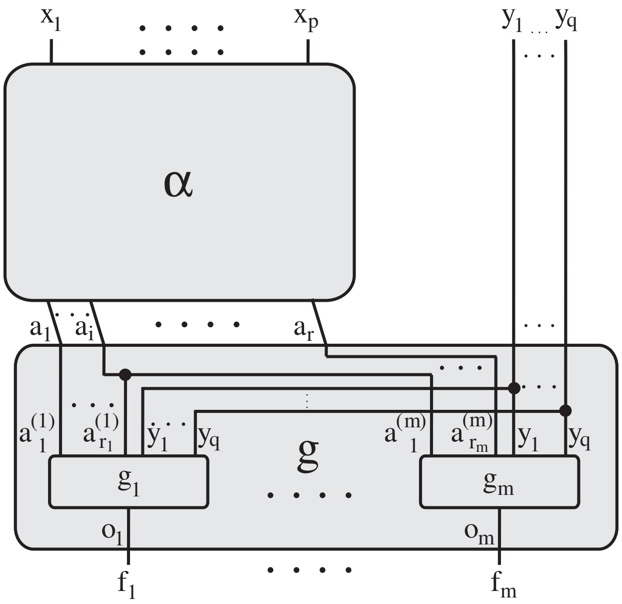
treten gewisse Codewerte $(a_1^{(i)}, \ldots, a_{r_i}^{(i)})$ im Bild von $(\alpha_1^{(i)}, \ldots, \alpha_{r_i}^{(i)})$ nicht auf. Dann ist die Zusammensetzungsfunktion g_i auf jeden Fall eine partielle Funktion.
Bei der weiteren Zerlegung von Funktionen, deren Freiheitsrelation eine partielle Funktion ist, können dann Situationen auftreten, in denen sich die Freiheitsrelationen *nicht* mehr als partielle Funktionen darstellen lassen:
Betrachte folgenden Ausschnitt aus der Zerlegungsmatrix zu einer Funktion g mit einem Ausgang:

$$
\begin{array}{cc|ccc|cccc}
 & & & & x_4 & 0 & 0 & 1 & 1 \\
 & & & & x_5 & 0 & 1 & 0 & 1 \\
\beta_1 & \beta_2 & & x_1x_2x_3 & & & & & \\
\hline
 & & & \vdots & & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \Longleftarrow & 0\,1\,1 & & \star & 1 & 0 & 0 \\
 & & & \vdots & & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \Longleftarrow & 1\,0\,1 & & 0 & \star & 0 & \star \\
1 & 1 & \Longleftarrow & 1\,1\,0 & & 1 & \star & 0 & \star \\
 & & & \vdots & & \vdots & \vdots & \vdots & \vdots
\end{array}
$$

Matrixeinträge mit $\star$ drücken aus, daß die entsprechende Belegung nicht als Eingabe an g vorkommen kann. Sei für die Zerlegungsfunktion β $\beta(011) = (00)$, $\beta(101) = (00)$ und $\beta(110) = (11)$. Gilt nun für die Zusammensetzungsfunktion h mit den Eingangsvariablen b_1 und b_2 (für die Ausgänge von β_1 und β_2) und x_4, x_5 $h(1,1,0,1) = 1$ und $h(1,1,1,1) = 0$, dann ist es aber auch möglich, $(0,1,1)$ durch (β_1, β_2) mit $(1,1)$ zu kodieren, ohne g an den definierten Stellen (d.h. an den Stellen ohne $\star$) zu ändern. Die Freiheitsrelation von $\beta = (\beta_1, \beta_2)$ enthält dann also außer $(0,1,1,0,0)$ auch $(0,1,1,1,1)$, da eine Änderung von $\beta(0,1,1) = (0,0)$ auf $\beta(0,1,1) = (1,1)$ die Gesamtfunktion nicht verändern würde.
Würde ein Fehler in der Realisierung von β die Ausgabe von β bei Eingabe von $(0,1,1)$ von $(0,0)$ auf $(1,1)$ ändern, so könnte dieser Fehlereffekt an den Ausgängen der Gesamtfunktion nicht beobachtet werden.

Im Rahmen des rekursiven Zerlegungsverfahrens kann man annehmen, daß man außer der gerade zu realisierenden totalen Funktion $f \in B_{n,m}$ auch die Freiheitsrelation FR_f zur Verfügung hat, die sich aus der Einbettung der Realisierung von f in eine Gesamtschaltung ergibt. Für die oberste Rekursionsstufe ist dies klar, für die weiteren Rekursionsstufen wird im folgenden gezeigt, wie man aus der Freiheitsrelation FR_f von f, der totalen Zerlegungsfunktion α und der totalen Zusammensetzungsfunktion g die Freiheitsrelationen von α und g berechnen kann.

Die Zerlegung erfolge hinsichtlich der Variablenteilmenge $\{x_1, \ldots, x_p\}$ wie im untenstehenden Bild angegeben:



Die Zerlegungsfunktionen der Funktion f_i sind mit $\alpha_1^{(i)}, \ldots, \alpha_{r_i}^{(i)}$ bezeichnet, die Gesamtmenge der Zerlegungsfunktionen sei $\{\alpha_1, \ldots, \alpha_r\} = \bigcup_{i=1}^{m} \{\alpha_1^{(i)}, \ldots \alpha_{r_i}^{(i)}\}$. Die Eingänge der zugehörigen Zusammensetzungsfunktionen g_i sind entsprechend mit $a_1^{(i)}, \ldots, a_{r_i}^{(i)}$ bezeichnet, die Eingänge von $g = (g_1, \ldots, g_m)$ mit $a_1, \ldots, a_r$. Die Ausgänge von $f_1, \ldots, f_m$ sind mit den Variablen $o_1, \ldots, o_m$ bezeichnet.

Bei der Bestimmung von FR_α und FR_g im allgemeinen Fall werden Existenz– und Allquantoren Boolescher Funktionen verwendet. Der Existenzquantor einer Booleschen Funktion f mit den Eingangsvariablen $x_1, \ldots, x_p$, $y_1, \ldots, y_q$ hinsichtlich der Variablen $x_1, \ldots, x_p$ ist hierbei definiert als

$$\exists_{x_1,\ldots,x_p} f = \bigvee_{(\epsilon_1,\ldots,\epsilon_p)=(0,\ldots,0)}^{(1,\ldots,1)} f_{x_1^{\epsilon_1},\ldots,x_p^{\epsilon_p}}.$$

Der Allquantor hinsichtlich $x_1, \ldots, x_p$ ist definiert durch

$$\forall_{x_1,\ldots,x_p} f = \bigwedge_{(\epsilon_1,\ldots,\epsilon_p)=(0,\ldots,0)}^{(1,\ldots,1)} f_{x_1^{\epsilon_1},\ldots,x_p^{\epsilon_p}}.$$

Sind Funktionen f durch ROBDDs gegeben, so beobachtet man anhand von praktischen Beispielen, daß Existenz– und Allquantor von f oft sehr effizient berechnet werden können [HDB96].

Seien nun die totale Zerlegungsfunktion α und die totale Zusammensetzungsfunktion g von f gegeben (m.H. von Verfahren aus den Abschnitten 3.1 und 3.4).

Bestimmung von FR_α Allgemein erhält man aus den ROBDDs für χ_{FR_f} und $\chi_{R(g)}$ den ROBDD zur Freiheitsrelation χ_{FR_α} nach der folgenden Formel[6]:

$$\chi_{FR_\alpha}(\mathbf{x},\mathbf{a}) = \forall_{\mathbf{y}}(\exists_{\mathbf{o}}(\chi_{FR_f}(\mathbf{x},\mathbf{y},\mathbf{o}) \cdot \chi_{R(g)}(\mathbf{a},\mathbf{y},\mathbf{o}))). \qquad (\star)$$

Beweis: Sei R_{rgl} die Relation, deren charakteristische Funktion durch die rechte Seite von Gleichung $(\star)$ beschrieben wird.

$FR_\alpha \subseteq R_{rgl}$: Sei $(\tilde{\mathbf{x}},\tilde{\mathbf{a}}) \in FR_\alpha$.
Angenommen $(\tilde{\mathbf{x}},\tilde{\mathbf{a}}) \notin R_{rgl}$.
Dann gäbe es ein $\tilde{\mathbf{y}}$, so daß für alle $\tilde{\mathbf{o}}$ $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin FR_f$ oder $(\tilde{\mathbf{a}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin R(g)$ gilt, d.h. speziell für $\tilde{\mathbf{o}} = g(\tilde{\mathbf{a}},\tilde{\mathbf{y}})$ würde $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin FR_f$ gelten, so daß sich ein Widerspruch zu $(\tilde{\mathbf{x}},\tilde{\mathbf{a}}) \in FR_\alpha$ ergeben würde.

$R_{rgl} \subseteq FR_\alpha$: Sei $(\tilde{\mathbf{x}},\tilde{\mathbf{a}}) \in R_{rgl}$.
Angenommen $(\tilde{\mathbf{x}},\tilde{\mathbf{a}}) \notin FR_\alpha$.
Dann muß es ein $\tilde{\mathbf{y}}$ geben, so daß $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},g(\tilde{\mathbf{a}},\tilde{\mathbf{y}})) \notin FR_f$. Da g eine totale Funktion ist, kann es dann aber kein $\tilde{\mathbf{o}}$ geben, so daß $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \in FR_f$ *und* $(\tilde{\mathbf{a}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \in R(g)$.

□

Alternative Berechnungsmöglichkeiten für FR_α ergeben sich für die Sonderfälle, daß $m = 1$ oder FR_f eine totale Funktion ist ($FR_f = R(f)$) (siehe Anhang C.1.1).

Bestimmung von FR_g Aus den ROBDDs für χ_{FR_f} und $\chi_{R(\alpha)}$ erhält man den ROBDD zur Freiheitsrelation χ_{FR_g}:

$$\chi_{FR_g}(\mathbf{a},\mathbf{y},\mathbf{o}) = \forall_{\mathbf{x}}(\chi_{R(\alpha)}(\mathbf{x},\mathbf{a}) \Longrightarrow \chi_{FR_f}(\mathbf{x},\mathbf{y},\mathbf{o})) = \forall_{\mathbf{x}}(\overline{\chi_{R(\alpha)}(\mathbf{x},\mathbf{a})} \vee \chi_{FR_f}(\mathbf{x},\mathbf{y},\mathbf{o})) \qquad (\star\star)$$

Beweis: Sei R_{rgl} die Relation, deren charakteristische Funktion durch die rechte Seite von Gleichung $(\star\star)$ beschrieben wird.

$FR_g \subseteq R_{rgl}$: Sei $(\tilde{\mathbf{a}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \in FR_g$.
Angenommen $(\tilde{\mathbf{a}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin R_{rgl}$.
Dann gäbe es ein $\tilde{\mathbf{x}}$, so daß $\alpha(\tilde{\mathbf{x}}) = \tilde{\mathbf{a}}$ und $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin FR_f$.
Dann gilt $(\tilde{\mathbf{a}},\tilde{\mathbf{y}},\tilde{\mathbf{o}}) \notin FR_g$, so daß sich ein Widerspruch zur Annahme ergibt.

[6] $\mathbf{x}$ steht abkürzend für $x_1,\ldots,x_p$, $\mathbf{a}$ für $a_1,\ldots,a_r$, $\mathbf{y}$ für $y_1,\ldots,y_q$ und $\mathbf{o}$ für $o_1,\ldots,o_m$.

$R_{rgl} \subseteq FR_g$: Sei $(\tilde{\mathbf{a}}, \tilde{\mathbf{y}}, \tilde{\mathbf{o}}) \in R_{rgl}$.
Angenommen $(\tilde{\mathbf{a}}, \tilde{\mathbf{y}}, \tilde{\mathbf{o}}) \notin FR_g$.
Dann muß es nach Definition der Freiheitsrelation FR_g ein $\tilde{\mathbf{x}}$ geben, so daß $\alpha(\tilde{\mathbf{x}}) = \tilde{\mathbf{a}}$ und $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{o}}) \notin FR_f$.
$\Longrightarrow (\tilde{\mathbf{a}}, \tilde{\mathbf{y}}, \tilde{\mathbf{o}}) \notin R_{rgl}$, Widerspruch zur Annahme.

□

Wenn das Zerlegungsverfahren nicht rekursiv auf die gesamte Funktion $g = (g_1, \ldots, g_m)$ angewendet wird, sondern nur auf einzelne Ausgangsfunktionen g_i oder auf Funktionen $h = (g_{i_1}, \ldots, g_{i_s})$ $(\cup_{l=1}^{s}\{i_l\} \subseteq \{1, \ldots, m\})$ (vergleiche Abschnitt 3.4.4), so muß man die Relationen FR_{g_i} bzw. FR_h berechnen.
Hierzu kann man eine leicht abgeänderte Version von Gleichung $(\star\star)$ benutzen: Um der Tatsache Rechnung zu tragen, daß bei der Betrachtung der Freiheitsrelation von h die Ausgangsfunktionen f_i mit $i \notin \{i_1, \ldots, i_s\}$ fest bleiben, muß $\chi_{FR_f}(\mathbf{x}, \mathbf{y}, \mathbf{o})$ ersetzt werden durch $\exists_{\mathbf{o2}}(\chi_{FR_f}(\mathbf{x}, \mathbf{y}, \mathbf{o}) \cdot \chi_{R(f\_ohne\_h)}(\mathbf{x}, \mathbf{y}, \mathbf{o2}))$, wobei $f\_ohne\_h = (f_{j_1}, \ldots, f_{j_{m-s}})$ mit $\cup_{l=1}^{m-s}\{j_l\} = \{1, \ldots, m\} \setminus (\cup_{l=1}^{s}\{i_l\})$ und $\mathbf{o2} = (o_{j_1}, \ldots, o_{j_{m-s}})$. Es ergibt sich also (mit der Bezeichnung $\mathbf{o1} = (o_{i_1}, \ldots, o_{i_s})$)

$$\chi_{FR_h}(\mathbf{a}, \mathbf{y}, \mathbf{o1}) = \forall_{\mathbf{x}} \left(\overline{\chi_{R(\alpha)}(\mathbf{x}, \mathbf{a})} \vee \left(\exists_{\mathbf{o2}}(\chi_{FR_f}(\mathbf{x}, \mathbf{y}, \mathbf{o}) \cdot \chi_{R(f\_ohne\_h)}(\mathbf{x}, \mathbf{y}, \mathbf{o2}))\right)\right).$$

Wie bei FR_α werden auch hier in Anhang C.1.2 Sonderfälle zur Berechnung von FR_g für $m = 1$ und $FR_f = R(f)$ betrachtet.

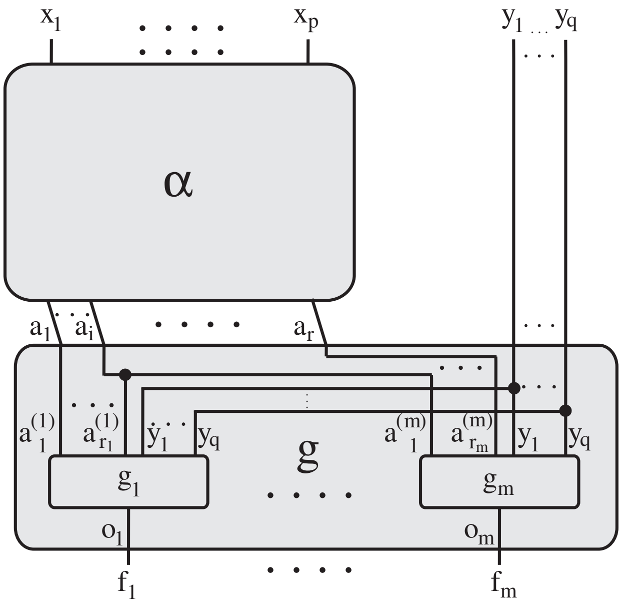
Es ist nun geklärt, daß man bei einem rekursiven Aufruf des Zerlegungsverfahrens annehmen kann, daß zusätzlich zu der zu realisierenden totalen Booleschen Funktion $f \in B_{n,m}$ auch die Freiheitsrelation FR_f von f gegeben ist.

Es bleibt die Aufgabe, rekursiv für die modellierten Fehler einen Test zu berechnen, d.h. eine Belegung der primären Inputs, die den Fehlereffekt an den primären Outputs sichtbar macht. Ein rekursiver Aufruf für eine Funktion f soll dazu nicht nur eine Realisierung für f zurückliefern, sondern auch eine Liste der nichtredundanten[7] Fehler in der Realisierung von f und zu jedem nichtredundanten Fehler eine Belegung $(\tilde{i_1}, \ldots, \tilde{i_n})$ der Inputs von f, so daß bei Einpflanzen dieses Fehlers eine Ausgabe $(\tilde{o_1}, \ldots, \tilde{o_m})$ von f entsteht, für die $(\tilde{i_1}, \ldots, \tilde{i_n}, \tilde{o_1}, \ldots, \tilde{o_m})$ nicht in der Freiheitsrelation $FR(f)$ enthalten ist. Somit muß es eine Belegung der primären Inputs der Gesamtschaltung geben, so daß der Fehlereffekt an den Outputs der Gesamtschaltung beobachtet werden kann.

Ist n genügend klein (z.B. wenn auch das rekursive Zerlegungsverfahren abgebrochen wird), so wird durch Anlegen sämtlicher Eingaben an die Realisierung von f und Vergleichen mit der Freiheitsrelation FR_f von f für jeden nichtredundanten modellierten Fehler ein solches Paar von Eingabe an f und gegebener (fehlerhafter) Ausgabe von f bestimmt.

Ansonsten wird f durch Zerlegung realisiert wie in der folgenden Abbildung angegeben (siehe auch oben):

[7]„Nichtredundant" bedeutet hier nichtredundant bzgl. der *Gesamtschaltung* und nicht nichtredundant bzgl. der Realisierung der Teilfunktion f.



Der rekursive Aufruf des Zerlegungsverfahrens liefert für alle nichtredundanten Fehler in der Realisierung von α sowohl eine Belegung für $x_1, \ldots, x_p$ als auch eine dazugehörige fehlerhafte Ausgabe von α. Ebenso hat man für alle nichtredundanten Fehler in der Realisierung von g eine Belegung für $(a_1, \ldots, a_r, y_1, \ldots, y_q)$ und eine dazugehörige fehlerhafte Ausgabe von g. Zu jedem nichtredundanten Fehler sind nun sowohl eine Belegung $(\tilde{i_1}, \ldots, \tilde{i_n})$ von $(x_1, \ldots, x_p, y_1, \ldots, y_q)$ als auch eine fehlerhafte Ausgabe $(\tilde{o_1}, \ldots, \tilde{o_m})$ von f zu berechnen, so daß $(\tilde{i_1}, \ldots, \tilde{i_n}, \tilde{o_1}, \ldots, \tilde{o_m}) \notin FR_f$.

<u>1. Fall</u>: Nichtredundanter Fehler in α.
Gegeben sind ein $(\tilde{x_1}, \ldots, \tilde{x_p}) \in \{0,1\}^p$ und eine fehlerhafte Ausgabe $(\tilde{a}_1, \ldots, \tilde{a}_r) \in \{0,1\}^r$ bei Einpflanzen des Fehlers, $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{a}_1, \ldots, \tilde{a}_r) \notin FR_\alpha$.
Berechne

$$T = \overline{(\chi_{FR_f})_{x_1^{\tilde{x_1}} \ldots x_p^{\tilde{x_p}}}} \cdot (\chi_{R(g)})_{a_1^{\tilde{a_1}} \ldots a_r^{\tilde{a_r}}}.$$

Wegen $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{a}_1, \ldots, \tilde{a}_r) \notin FR_\alpha$ gilt auf jeden Fall $T \neq 0$.
Wähle dann $(\tilde{y_1}, \ldots, \tilde{y_q}, \tilde{o_1}, \ldots, \tilde{o_m}) \in ON(T)$.
Dann liefert die Realisierung von f bei Einpflanzen des Fehlers unter Eingabe von $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{y_1}, \ldots, \tilde{y_q})$ die fehlerhafte Ausgabe $(\tilde{o_1}, \ldots, \tilde{o_m})$ und $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{y_1}, \ldots, \tilde{y_q}, \tilde{o_1}, \ldots, \tilde{o_m}) \notin FR_f$.

<u>2. Fall</u>: Nichtredundanter Fehler in g.
Gegeben sind ein $(\tilde{a}_1, \ldots, \tilde{a}_r, \tilde{y}_1, \ldots, \tilde{y}_q) \in \{0,1\}^{r+q}$ und eine fehlerhafte Ausgabe $(\tilde{o_1}, \ldots, \tilde{o_m}) \in \{0,1\}^m$ bei Einpflanzen des Fehlers, $(\tilde{a}_1, \ldots, \tilde{a}_r, \tilde{y}_1, \ldots, \tilde{y}_q, \tilde{o_1}, \ldots, \tilde{o_m}) \notin FR_g$.
Berechne

$$T = \overline{(\chi_{FR_f})_{y_1^{\tilde{y_1}} \ldots y_q^{\tilde{y_q}} o_1^{\tilde{o_1}} \ldots o_m^{\tilde{o_m}}}} \cdot (\chi_{R(\alpha)})_{a_1^{\tilde{a_1}} \ldots a_r^{\tilde{a_r}}}.$$

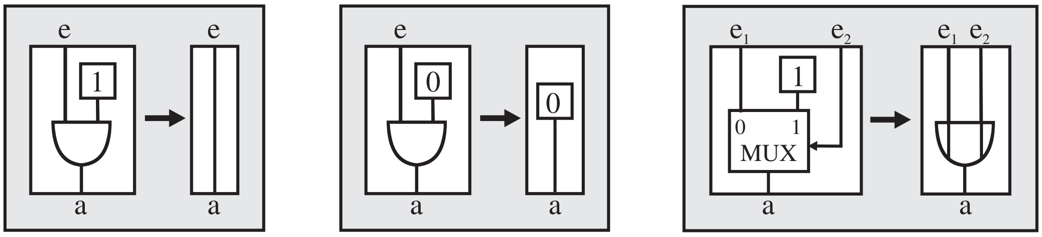


Abbildung 3.34: Beispiele für Schaltkreisvereinfachung durch „Konstantenpropagation“.

Wegen $(\tilde{a_1}, \ldots, \tilde{a_r}, \tilde{y_1}, \ldots, \tilde{y_q}, \tilde{o_1}, \ldots, \tilde{o_m}) \notin FR_g$ gilt auf jeden Fall $T \neq 0$.
Wähle dann $(\tilde{x_1}, \ldots, \tilde{x_p}) \in ON(T)$.
Dann liefert die Realisierung von f bei Einpflanzen des Fehlers unter Eingabe von $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{y_1}, \ldots, \tilde{y_q})$ die fehlerhafte Ausgabe $(\tilde{o_1}, \ldots, \tilde{o_m})$ und $(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{y_1}, \ldots, \tilde{y_q}, \tilde{o_1}, \ldots, \tilde{o_m}) \notin FR_f$.

3.7.2 Redundanzen

Die bisherigen Ausführungen zur Berechnung von Tests legen eine einfache Frage nahe: Wenn es möglich ist, redundante Fehler zu erkennen, ist man dann nicht auch in der Lage, während des Logiksyntheseverfahrens diese Tatsache auszunutzen und evtl. Schaltkreise ohne redundante Fehler zu erzeugen?

Diese Frage soll in diesem Abschnitt anhand des (Einzel–)Stuck–at–Fehlermodells untersucht werden.
Ein Stuck–at–0–Fehler (s-a-0–Fehler) an einem Gattereingang bzw. –ausgang äußert sich so, daß der Eingang bzw. Ausgang des Gatters konstant auf dem logischen Wert 0 liegt, ein Stuck–at–1–Fehler (s-a-1–Fehler) äußert sich entsprechend so, daß der Eingang bzw. Ausgang konstant auf 1 liegt. Ist ein Stuck–at–Fehler in einem Schaltkreis S redundant, so läßt sich S im allgemeinen vereinfachen: Ist der s-a-ϵ–Fehler ($\epsilon \in \{0, 1\}$) an einem Gattereingang redundant, so kann man diesen Eingang direkt mit der Konstanten ϵ verbinden. Ist an einem Gatterausgang s-a-ϵ redundant, so kann man das Gatter durch die Konstante ϵ ersetzen. In beiden Fällen besteht noch die Möglichkeit, den Schaltkreis weiter zu vereinfachen, indem man die eingesetzten Konstanten „weiterpropagiert“, d.h. indem man die auf die Konstante folgenden Gatter vereinfacht (vgl. Abbildung 3.34).

Ist eine totale Funktion $f \in B_{n,m}$ und *die Freiheitsrelation* FR_f gegeben und wird f zerlegt mit Zerlegungsfunktion α und Zusammensetzungsfunktion g (vgl. Abbildung auf Seite 212), so kann man verhindern, daß redundante Stuck–at–Fehler an den Ausgängen von α

bzw. an den Eingängen von g entstehen: Zerlegt man beispielsweise auf ROBDD–Basis, so hat man durch die Zerlegung ROBDDs zu $\alpha_1, \ldots, \alpha_r$ und $g_1, \ldots, g_m$ zur Verfügung. Will man überprüfen, ob der s-a-ϵ–Fehler an Ausgang Nr. i von α redundant ist, so kann man dies tun, indem man α_i durch die Konstante ϵ ersetzt und dann mit Hilfe von *compose*–Operationen [Bry86] die ROBDDs zu $f'_1, \ldots, f'_m$ berechnet, die sich bei dieser Ersetzung aus $f_1, \ldots, f_m$ ergeben. Der s-a-ϵ–Fehler ist am i. Ausgang von α ist genau dann redundant, wenn $R(f'_1, \ldots, f'_m) \subseteq FR_f$ (bzw. wenn $\chi_{R(f')} \cdot \overline{\chi_{FR_f}} = 0$). Ist dieser s-a-$\epsilon$–Fehler redundant, so kann α_i weggelassen werden (bzw. durch ϵ ersetzt werden). Es ergibt sich eine Zerlegung zu einer anderen totalen Funktion f' mit weniger Zerlegungsfunktionen. Da f' aber weiterhin in der Freiheitsrelation $FR_f = FR_{f'}$ liegt, ändert sich das Verhalten der Gesamtschaltung nicht.
Schließlich erhält man eine Zerlegung einer (evtl. veränderten) Funktion $\tilde{f}$ mit $R(\tilde{f}) \subseteq FR_f$ (mit Zerlegungsfunktion $\tilde{\alpha}$ und Zusammensetzungsfunktion $\tilde{g}$), bei der keine redundanten s-a–Fehler an den Ausgängen von $\tilde{\alpha}$ bzw. den Eingängen von $\tilde{g}$ auftreten.

Man könnte nun auf den ersten Blick annehmen, daß das erwähnte Verfahren in der Lage ist, Schaltkreise zu erzeugen, die vollständig frei sind von redundanten s-a–Fehlern. Es stellt sich aber heraus, daß das Verfahren zwar in der Lage ist, die Anzahl der Redundanzen zu minimieren, aber keine garantiert s-a–fehlerfreien Schaltkreise erzeugt. Dies liegt an der Tatsache, daß *gerade das Entfernen redundanter s-a–Fehler verhindert, daß man zum Zeitpunkt der Synthese eines Teilschaltkreises die Freiheitsrelation zu diesem Teilschaltkreis genau kennen kann*: Angenommen f ist zu zerlegen und nach Entfernen redundanter s-a–Fehler an den Ausgängen der Zerlegungsfunktion α bzw. den Eingängen der Zusammensetzungsfunktion g erhält man die Zerlegungsfunktion $\tilde{\alpha}$ und die Zusammensetzungsfunktion $\tilde{g}$ zu einer Funktion $\tilde{f}$ mit $R(\tilde{f}) \subseteq FR_f$. Nun wird die Freiheitsrelation zu $\tilde{g}$ berechnet (aus FR_f und der totalen Funktion $\tilde{\alpha}$) und es wird (rekursiv) eine Realisierung zu $\tilde{g}$ berechnet. Dabei wird die Freiheitsrelation $FR_{\tilde{g}}$ ausgenutzt, um redundante s-a–Fehler in der Realisierung von $\tilde{g}$ zu vermeiden. Anschließend wird die Freiheitsrelation zu $\tilde{\alpha}$ berechnet (aus FR_f und der realisierten Zusammensetzungsfunktion $\tilde{\tilde{g}}$[††]) und $\tilde{\alpha}$ wird realisiert. Der springende Punkt ist nun, daß bei der Realisierung von $\tilde{\alpha}$ evtl. die Freiheitsrelation $FR_{\tilde{\alpha}}$ ausgenutzt wird, um redundante s-a–Fehler zu vermeiden, so daß eine geänderte totale Funktion $\tilde{\tilde{\alpha}}$ realisiert wird (mit $R(\tilde{\tilde{\alpha}}) \subseteq FR_{\tilde{\alpha}}$). Durch Änderung von $\tilde{\alpha}$ zu $\tilde{\tilde{\alpha}}$ kann sich jedoch die Freiheitsrelation von $\tilde{\tilde{g}}$ ändern! Bei der Vermeidung redundanter s-a–Fehler in der Realisierung von $\tilde{g}$ wurde dann evtl. eine andere Freiheitsrelation vorausgesetzt. Es ist möglich, daß Fehler, die hinsichtlich der alten Freiheitsrelation testbar waren (d.h. unter Voraussetzung von $\tilde{\alpha}$), jetzt (unter Voraussetzung von $\tilde{\tilde{\alpha}}$) redundant werden[‡‡]. Im Beispiel aus Anhang C.2 wurde eine solche Situation konstruiert.

Das Beispiel zeigt, daß das obige Verfahren zwar dazu benutzt werden kann, die Anzahl der redundanten Stuck–at–Fehler zu verringern. Es kann aber nicht garantiert werden, daß das Resultat völlig frei von redundanten Stuck–at–Fehlern ist.

[††] $\tilde{g}$ wurde evtl. durch Entfernen redundanter s-a–Fehler zu $\tilde{\tilde{g}}$ verändert.
[‡‡] Eine analoge Situation ergibt sich natürlich auch, wenn man zuerst $\tilde{\alpha}$ und dann erst $\tilde{g}$ realisiert.

Kapitel 4

Experimentelle Resultate

Die beschriebenen Verfahren zur Zerlegung Boolescher Funktionen mit mehreren Ausgängen wurden implementiert und es wurden Schaltkreise für Beispielfunktionen damit entworfen. Das Kapitel beginnt mit der Beschreibung einiger kleinerer Beispiele. Danach werden automatisch generierte Realisierungen für einen Addierer und einen partiellen Multiplizierer etwas ausführlicher betrachtet. Anhand des Entwurfs eines fehlererkennenden Dividierers [TY88] wird das Zusammenspiel des implementierten Logiksynthesewerkzeuges mit CADIC [BHK+87], einem im Rahmen des Sonderforschungsbereichs 124 „VLSI–Entwurfsmethoden und Parallelität“ entwickelten Entwurfssystem, demonstriert. Schließlich werden Resultate der Logiksynthese für einige Benchmarkschaltungen angegeben.

Zunächst soll anhand zweier einfacher Beispiele die Wirkungsweise des Algorithmus gezeigt werden (es wurden jeweils zweiseitige Zerlegungen durchgeführt):

Beispiel 4.1 Beim ersten Beispiel handelt es sich um eine $exor$–Funktion mit 8 Eingängen.

Wie in Abbildung 4.1 ersichtlich, handelt es sich bei dem durch den Algorithmus berechneten Schaltkreis um einen balancierten Baum aus 7 $exor_2$–Gattern. In der ersten Rekursionsstufe wurde eine Variablenaufteilung in die Mengen $\{x_1, \ldots, x_4\}$ und $\{x_5, \ldots, x_8\}$ berechnet. Sowohl bei der Zerlegungsfunktion auf $\{x_1, \ldots, x_4\}$ als auch bei der Zerlegungsfunktion auf $\{x_5, \ldots, x_8\}$ handelt es sich um die $exor_4$–Funktion, die Zusammensetzungsfunktion ist die $exor_2$–Funktion. Für die Zerlegungsfunktionen wird rekursiv wiederum eine Zerlegung durchgeführt. Man sieht leicht, daß die gefundene Realisierung B_2–optimal ist. Eine Berechnung gemeinsamer Zerlegungsfunktionen erübrigt sich in diesem Beispiel, da die zu realisierenden Funktionen alle einen Ausgang haben.

Beispiel 4.2 Das nächste Beispiel ist eine Schwellenfunktion s_4^6, die folgendermaßen definiert ist:

$$s_4^6(x_0, \ldots, x_5) = 1 \iff \sum_{i=0}^{5} x_i \geq 4.$$

Der resultierende Schaltkreis ist in Abbildung 4.2 angegeben.

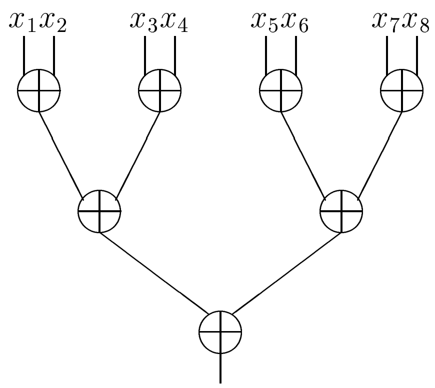


Abbildung 4.1: Realisierung zu $exor_8$

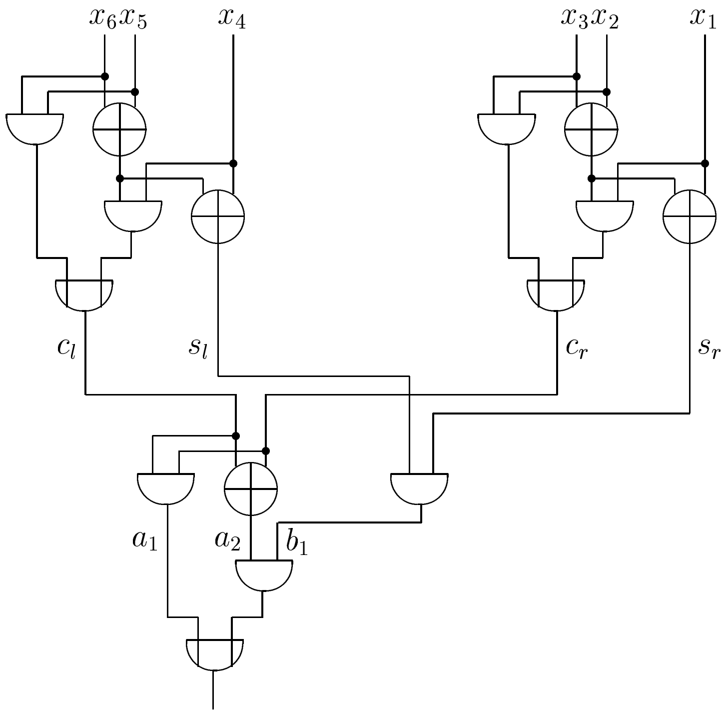
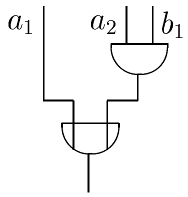


Abbildung 4.2: Realisierung zu s_4^6

Im ersten Schritt wurde eine Variablenaufteilung in die Mengen $\{x_1, x_2, x_3\}$ und $\{x_4, x_5, x_6\}$ gefunden. Auf $\{x_1, x_2, x_3\}$ werden 2 Zerlegungsfunktionen c_r und s_r berechnet. Es handelt sich gerade um Übertrags– und Summenbit der binären Addition von x_1, x_2 und x_3. Ebenso sind die Zerlegungsfunktionen c_l und s_l auf den Variablen $\{x_4, x_5, x_6\}$ Übertrags– und Summenbit der binären Addition von x_4, x_5 und x_6. Betrachtet man die gefundene Realisierung für (c_r, s_r), so erkennt man, daß es sich um einen Volladdierer handelt. Der Volladdierer ergibt sich bei der Zerlegung von c_r und s_r hinsichtlich $\{\{x_1\}, \{x_2, x_3\}\}$. Hierbei wird $x_2 \oplus x_3$ bei der Zerlegung von c_r und s_r gemeinsam benutzt. Analog wird (c_l, s_l) durch einen Volladdierer realisiert.

Die Zusammensetzungsfunktion auf der 1. Rekursionsstufe hat 4 Eingänge c_l, s_l, c_r und s_r. Bei der rekursiven Behandlung dieser Funktion wird die Variablenaufteilung $\{\{c_l, c_r\}, \{s_l, s_r\}\}$ berechnet. Auf c_l und c_r werden zwei Zerlegungsfunktionen a_1 und a_2, auf s_l und s_r wird eine Zerlegungsfunktion b_1 berechnet.

Die zugehörige Zusammensetzungsfunktion mit 3 Eingängen a_1, a_2 und b_1 wird wiederum rekursiv zerlegt. An dieser Stelle zeigt sich die Bedeutung der Behandlung von don't cares: Es handelt sich um eine partielle Funktion, da a_1 und a_2 nie gleichzeitig den Wert 1 annehmen können. Würde man die don't care–Menge einfach der OFF–Menge zuschlagen, d.h. die Funktionswerte für $(1, 1, 0)$ und $(1, 1, 1)$ auf 0 festlegen, so wäre die resultierende Funktion nicht nichttrivial zerlegbar. Die Funktion ist nämlich nur dann nichttrivial zerlegbar, wenn der Funktionswert für $(1, 1, 0)$ auf 1 festgelegt wird. Der implementierte Algorithmus legt die Funktionswerte für $(1, 1, 0)$ und $(1, 1, 1)$ auf 1 fest und erreicht somit eine nichttriviale Zerlegung mit Aufteilung der Variablen in $\{a_1\}$ und $\{a_2, b_1\}$. Die Funktion wird dann durch folgende einfache Teilschaltung realisiert:



Wählt man als Kostenmaß die R_2–Komplexität, wobei $R_2 = B_2 \setminus \{exor, equiv\}$, so ergeben sich als Kosten der berechneten Realisierung S für s_4^6

$$C_{R_2}(S) = 7C_{R_2}(and) + 3C_{R_2}(or) + 5C_{R_2}(exor) = 7 + 3 + 5 \cdot 3 = 25.$$

Realisiert man diese Schwellenfunktion zum Vergleich durch ihr eindeutiges Minimalpolynom, so erhält man

$$P = \bigvee_{0 \leq i_1 \leq \ldots \leq i_4 \leq 5} x_{i_1} \ldots x_{i_4}.$$

Das Minimalpolynom enthält $\binom{6}{4} = 15$ Monome der Länge 4. Die Kosten der Minimalpolynomrealisierung betragen also

$$C_{R_2}(P) = 15C_{R_2}(and_4) + C_{R_2}(or_{15}) = 15 \cdot 3 + 14 = 59.$$

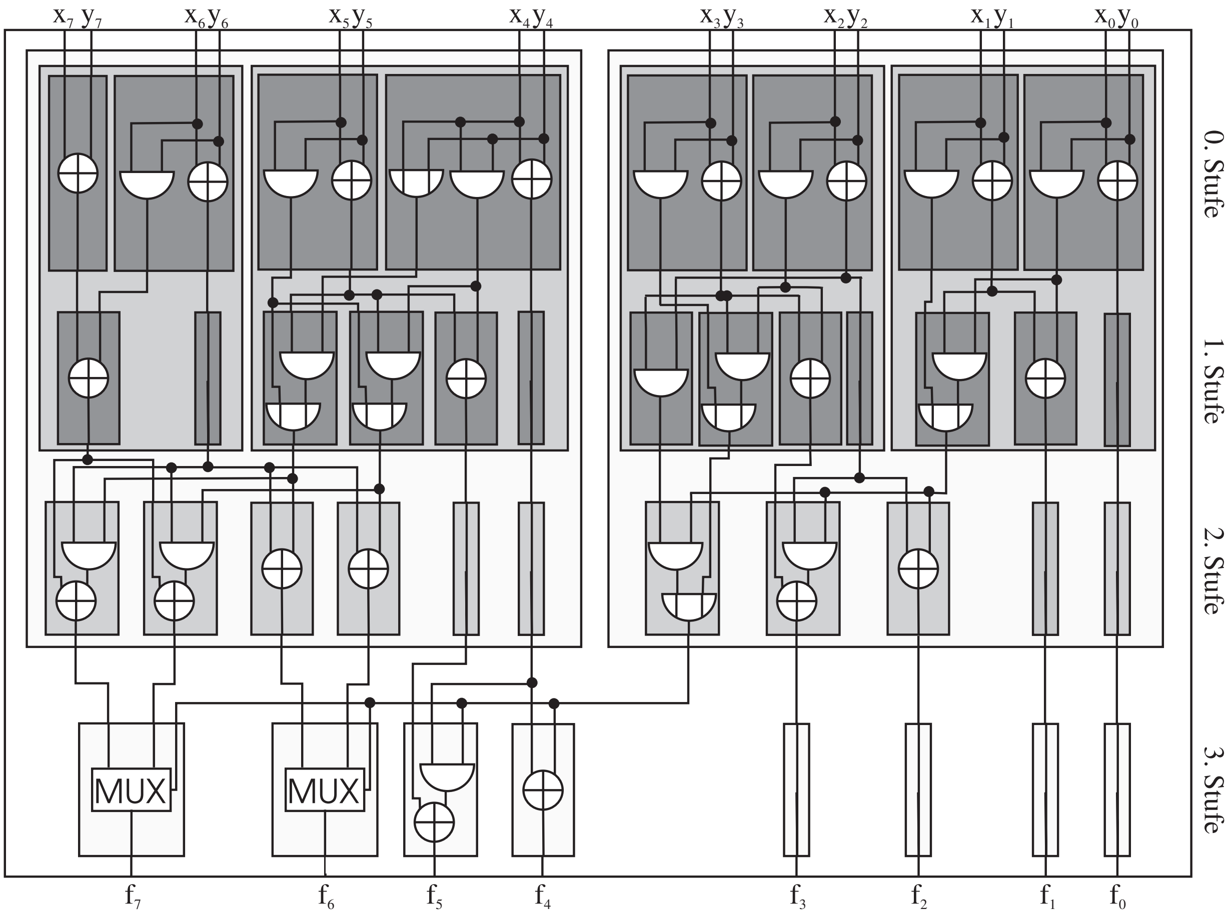


Abbildung 4.3: Automatisch generierter Schaltkreis zu einem 8–Bit–Addierer.

Realisierung eines Addierers

Läßt man den Algorithmus mit einem 8–Bit–Addierer als Eingabe (und der Vorgabe, zweiseitige Zerlegungen durchzuführen) ablaufen, so ergibt sich die Realisierung in Bild 4.3. Grau schattierte Boxen in der Abbildung stehen jeweils für die Ergebnisse rekursiver Aufrufe des Zerlegungsverfahrens.

Betrachtet man die Realisierung genauer, so erkennt man, daß sie die gleiche Struktur wie der Conditional–Sum–Addierer hat. (Zum Vergleich ist in Abbildung 4.4 der Conditional–Sum–Addierer für die Addition zweier 8–Bit–Zahlen angegeben.)

Auf der ersten Rekursionsstufe wurde die Variablenaufteilung in die beiden Mengen

$$\{x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3\} \text{ und } \{x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7\}$$

berechnet. Bei dieser Zerlegung wird die auf $\{x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3\}$ berechnete Zerle-

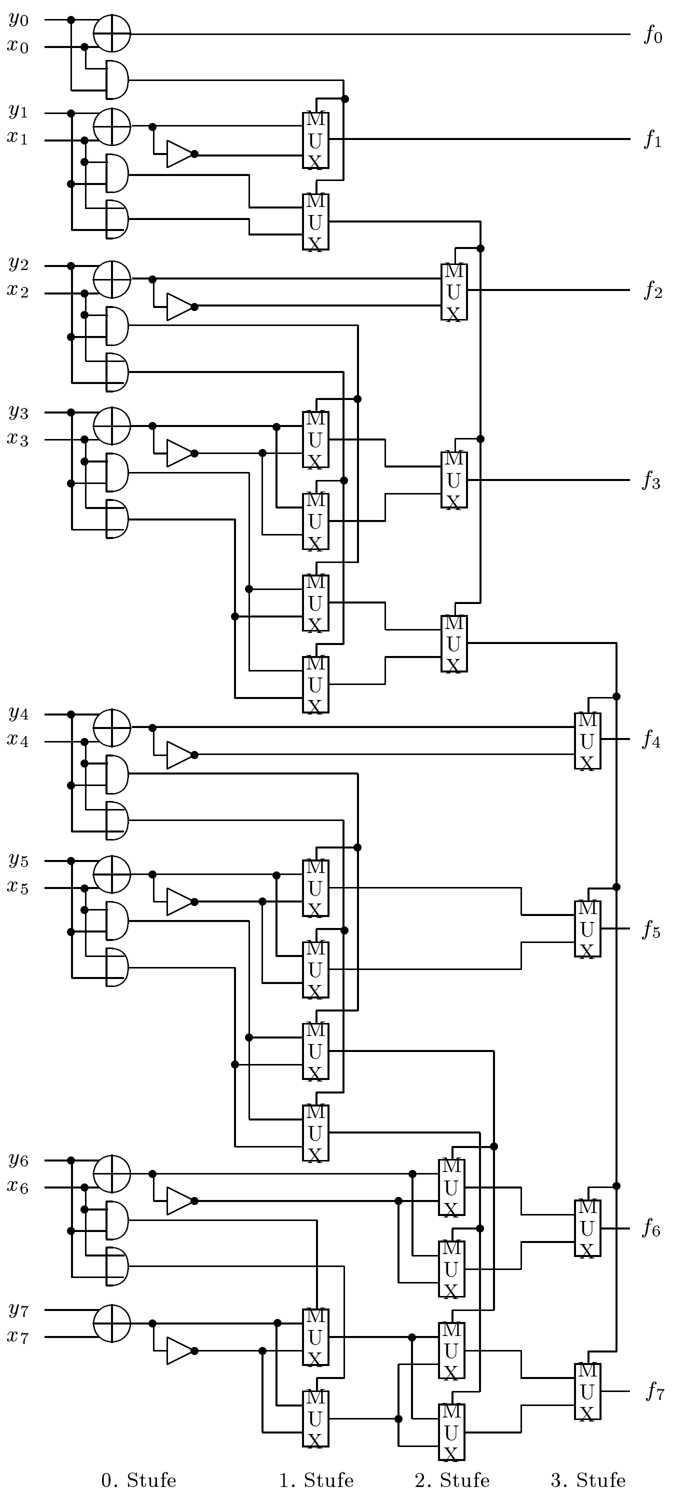


Abbildung 4.4: Conditional–Sum–Addierer mit Bitbreite 8.

gungsfunktion (das Carry–Bit der Addition von $(x_3x_2x_1x_0)$ und $(y_3y_2y_1y_0)$) für die Zerlegung von f_4, f_5, f_6 und f_7 *gemeinsam* benutzt.

Der Unterschied dieser Realisierung zum Conditional–Sum–Addierer liegt einerseits darin, daß grundsätzlich die Zahl der Zerlegungsfunktionen minimal gewählt wird und andererseits darin, daß nach Möglichkeit immer *gemeinsame* Zerlegungsfunktionen gewählt werden.

Dies führt dazu, daß bei der Zerlegung von f_4 auf der 1. Rekursionsstufe des Algorithmus (im Bild „3. Stufe") nur *eine* Zerlegungsfunktion auf $\{x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7\}$ berechnet wird, während der Conditional–Sum–Addierer an dieser Stelle 2 Funktionen benutzt, nämlich das letzte Bit der Summe von $(x_7x_6x_5x_4)$ und $(y_7y_6y_5y_4)$ und das letzte Bit der Summe von $(x_7x_6x_5x_4)$, $(y_7y_6y_5y_4)$ und 1.

Auf der Variablenmenge $\{x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7\}$ werden auf der 1. Rekursionsstufe für die Funktionen f_5, f_6 und f_7 sowohl bei der Realisierung aus Bild 4.3 als auch beim Conditional–Sum–Addierer 4 Informationen berechnet, die durch 2 Bit kodiert werden. Der Conditional–Sum–Addierer kodiert diese 4 Informationen jeweils fest durch die entsprechenden Bits der Summe und der Summe + 1. Bei Funktion f_5 wird in Bild 4.3 allerdings eine andere Kodierung dieser 4 Informationen gewählt. Der Grund für diese Wahl liegt darin, daß in diesem Fall die Zerlegungsfunktion von f_4 auf den Variablen $\{x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7\}$ von Funktion f_5 „mitverwendet" werden kann.

Diese Unterschiede bewirken, daß der durch den vorliegenden Algorithmus gefundene Addierer eine R_2–Komplexität von 89 hat, während der entsprechende Conditional–Sum–Addierer eine R_2–Komplexität von 106 hat. Die Tiefe ist in beiden Fällen die gleiche (bei einem parametrisierten Entwurf logarithmisch in der Anzahl der Eingänge).

Der enorme Vorteil der Verwendung von ROBDDs bei der Logiksynthese läßt sich anhand der Tabelle 4.1 ablesen. Sie enthält einen Vergleich zwischen den Laufzeiten des Synthesewerkzeuges in einer früheren Version, die auf der Bearbeitung von Funktionstabellen und Zerlegungsmatrizen basierte, und der aktuellen, ROBDD–basierten Version. Die sehr viel höheren Laufzeiten der tabellenbasierten Version lassen sich einfach dadurch erklären, daß wesentlich größere Datenmengen zu bearbeiten sind. Beispielsweise beim 16–Bit–Addierer würde die Größe einer Funktionstabelle (bei 32 Eingängen, 16 Ausgängen und einem Bit pro Tabelleneintrag) schon 8 Gigabyte betragen, beim 32–Bit–Addierer ca. 69 Milliarden Gigabyte. Dies macht schon deutlich, daß Probleme dieser Größenordnung von der tabellenbasierten Version allein wegen der Größe der benötigten Repräsentation nicht mehr bearbeitet werden können.

Mit der ROBDD–basierten Version konnten dagegen sogar Addierer bis zu einer Bitbreite von 64 Bit (128 Eingänge und 64 Ausgänge) generiert werden.

Der Vorteil der ROBDD–basierten Version ist darauf zurückzuführen, daß die entsprechenden Booleschen Funktionen durch ROBDDs *sehr viel* kompakter repräsentiert werden konnten. Die Spezifikation der Addierer konnte hierbei natürlich nicht mehr durch Funktionstabellen erfolgen, sondern mit einem Carry–Ripple–Addierer, einem einfachen Schaltkreis,

Schaltkreis	Laufzeit der Tabellenversion	Laufzeit der ROBDD–Version
$adder_4$	1,3 s	0,2 s
$adder_8$	18 h 38 min	1,15 s
$adder_{16}$	—	11,55 s
$adder_{32}$	—	3 min 8 s
$adder_{64}$	—	31 min 16 s

Tabelle 4.1: Laufzeiten für die Synthese von n–Bit–Addierern $adder_n$.

der die Addition nach der Schulmethode realisiert. Aus dieser Schaltkreisdarstellung wurden ROBDDs für die einzelnen Ausgangsfunktionen aufgebaut, die dann als Eingabe für das Logiksynthesewerkzeug dienten.

Realisierung eines partiellen Multiplizierers

Als nächstes wird die Synthese eines partiellen Multiplizierers betrachtet.

Ein partieller Multiplizierer der Bitbreite n ist eine Boolesche Funktion $pm_n : \{0,1\}^{n^2} \to \{0,1\}^{2n}$. Die Eingänge sind gegeben durch die Partialprodukte eines n–Bit–Multiplizierers, die Ausgänge entsprechen den Produktbits des n–Bit–Multiplizierers.
Sind die beiden zu multiplizierenden Operanden die Binärzahlen $(a_{n-1}, \ldots, a_0)$ und $(b_{n-1}, \ldots, b_0)$, und das Ergebnis der Multiplikation die Binärzahl $(r_{2n-1}, \ldots, r_0)$, so ergibt sich $(r_{2n-1}, \ldots, r_0)$ als Summe der n Partialprodukte $2^j \cdot (a_{n-1}b_j, \ldots, a_0 b_j)$ $(0 \leq j \leq n-1)$. Die Eingänge des partiellen Multiplizierers sind dann die Bits $p_{i,j} = a_i b_j$ $(0 \leq i, j \leq n-1)$. Dann gilt:

$$pm_n : \{0,1\}^{n^2} \to \{0,1\}^{2n}, pm_n(p_{n-1,n-1}, \ldots, p_{0,n-1}, \ldots, p_{n-1,0}, \ldots, p_{0,0}) = (r_{2n-1}, \ldots, r_0),$$

wobei

$$\sum_{i=0}^{2n-1} r_i 2^i = \sum_{j=0}^{n-1} \left[\left(\sum_{i=0}^{n-1} p_{i,j} \cdot 2^i \right) \cdot 2^j \right].$$

Bei der Synthese des partiellen Multiplizierers werden l–seitige Zerlegungen benutzt, wobei l die Anzahl der „Symmetriemengen“ der Variablenpartition $P_{\sim_{esym}} = \{\mu_1, \ldots, \mu_l\}$ ist, in der die zu realisierende Funktion symmetrisch ist. Sind keine Symmetriemengen mit Mächtigkeit größer als 2 vorhanden (d.h. hätte bei Zerlegung hinsichtlich $P_{\sim_{esym}}$ die Zusammensetzungsfunktion evtl. ebensoviele Eingänge wie die Ausgangsfunktion) oder ist nur eine einzige Symmetriemenge vorhanden, so wird zweiseitig zerlegt.

In Abbildung 4.5 ist exemplarisch der generierte Schaltkreis zu einem partiellen 4–Bit–Multiplizierer dargestellt.

Der Schaltkreis läßt sich folgendermaßen interpretieren:

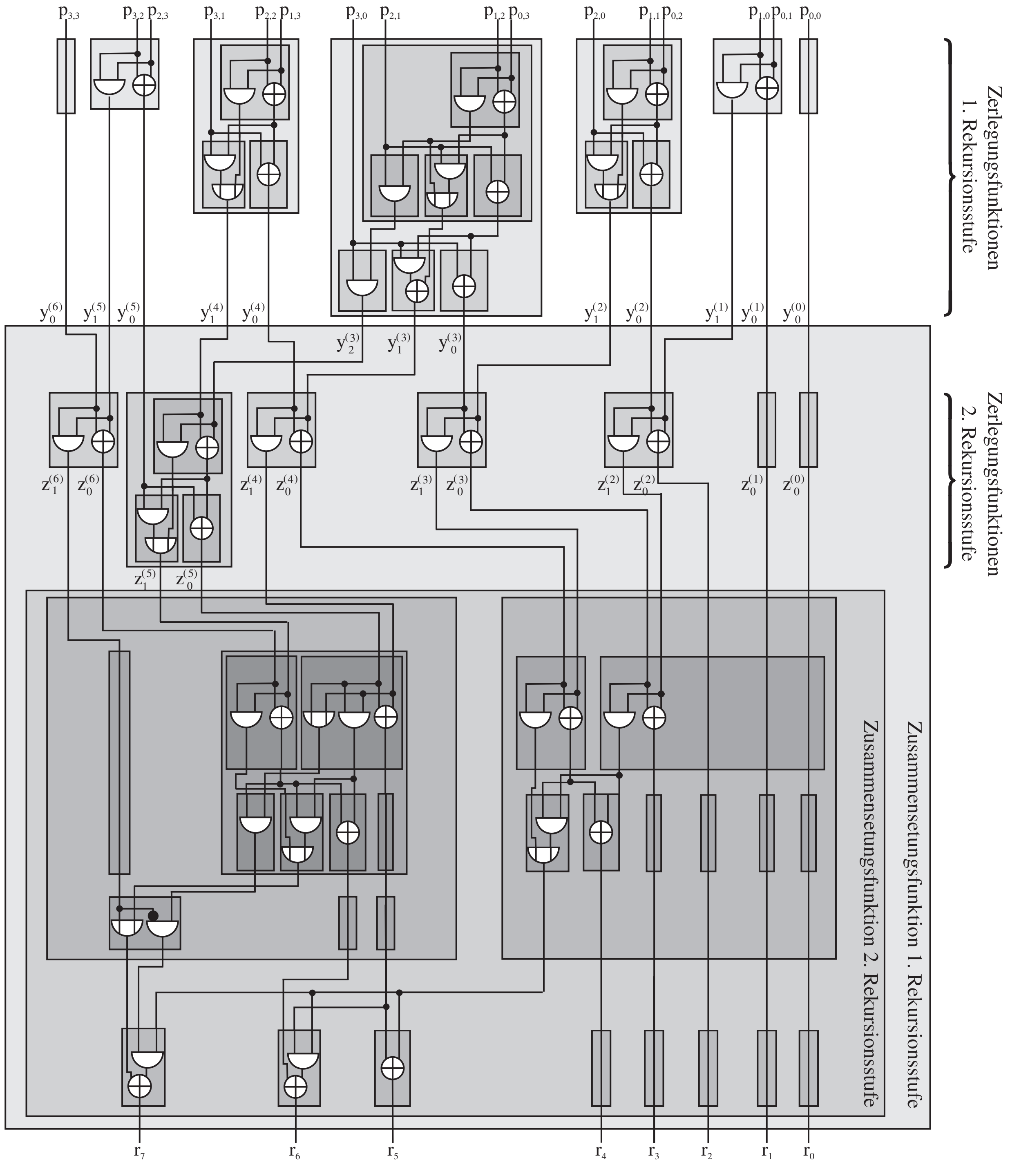


Abbildung 4.5: Automatisch generierter Schaltkreis zu einem partiellen 4–Bit–Multiplizierer.

Die Eingänge der Schaltung sind die Bits der Partialprodukte $p_{3,3}, p_{3,2}, \ldots, p_{0,0}$. Auf der ersten Rekursionsstufe erfolgt eine Zerlegung von pm_4 hinsichtlich der Symmetriemengen $\{p_{3,3}\}$, $\{p_{3,2}, p_{2,3}\}$, $\{p_{3,1}, p_{2,2}, p_{1,3}\}$, $\{p_{3,0}, p_{2,1}, p_{1,2}, p_{0,3}\}$, $\{p_{2,0}, p_{1,1}, p_{0,2}\}$, $\{p_{1,0}, p_{1,0}\}$ und $\{p_{0,0}\}$.

Die Zerlegungsfunktionen, die auf den einzelnen Symmetriemengen berechnet werden, (siehe Abbildung 4.5, „Zerlegungsfunktionen 1. Rekursionsstufe") kann man jeweils als Summe der Bits aus den Symmetriemengen interpretieren. (Beispielsweise werden $p_{3,1}$, $p_{2,2}$ und $p_{1,3}$ durch einen Volladdierer zu der 2–Bit–Zahl $(y_1^{(4)}, y_0^{(4)})$ addiert usw., siehe Abbildung 4.5.) Folglich ergeben sich nun 7 verschiedene Zahlen, deren Summe das Endergebnis der Multiplikation $(r_7, \ldots, r_0)$ darstellt. Folgende Addition wurde also auf der 1. Rekursionsstufe durchgeführt:

$$\begin{array}{cccccccc}
 & & & & p_{3,0} & p_{2,0} & p_{1,0} & p_{0,0} \\
 & & & p_{3,1} & p_{2,1} & p_{1,1} & p_{0,1} & \\
 & & p_{3,2} & p_{2,2} & p_{1,2} & p_{0,2} & & \\
 & p_{3,3} & p_{2,3} & p_{1,3} & p_{0,3} & & & \\
\hline
 & & & & & & & y_0^{(0)} \\
 & & & & & y_1^{(1)} & y_0^{(1)} & \\
 & & & & y_1^{(2)} & y_0^{(2)} & & \\
 & & y_2^{(3)} & y_1^{(3)} & y_0^{(3)} & & & \\
 & & y_1^{(4)} & y_0^{(4)} & & & & \\
 & y_1^{(5)} & y_0^{(5)} & & & & & \\
 & y_0^{(6)} & & & & & & \\
\hline
r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0
\end{array}$$

Die verbleibende Aufgabe der Addition dieser Zahlen (realisiert durch die Zusammensetzungsfunktion, siehe Abbildung 4.5, „Zusammensetzungsfunktion 1. Rekursionsstufe") läßt sich nach „Verschieben" von Bits gleicher Stelligkeit aber ebenso als Addition dreier Zahlen beschreiben:

$$\begin{array}{cccccccc}
 & y_1^{(5)} & y_2^{(3)} & y_1^{(3)} & y_1^{(2)} & y_1^{(1)} & y_0^{(1)} & y_0^{(0)} \\
 & y_0^{(6)} & y_1^{(4)} & y_0^{(4)} & y_0^{(3)} & y_0^{(2)} & & \\
 & & y_0^{(5)} & & & & & \\
\hline
r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0
\end{array}$$

Die Zusammensetzungsfunktion der 1. Rekursionsstufe wird hinsichtlich der Symmetriemengen $\{y_1^{(5)}, y_0^{(6)}\}$, $\{y_2^{(3)}, y_1^{(4)}, y_0^{(5)}\}$, $\{y_1^{(3)}, y_0^{(4)}\}$, $\{y_1^{(2)}, y_0^{(3)}\}$, $\{y_1^{(1)}, y_0^{(2)}\}$, $\{y_0^{(1)}\}$ und $\{y_0^{(0)}\}$ weiter zerlegt. Auch hier kann man die Zerlegungsfunktionen, die auf den einzelnen Symmetriemengen berechnet werden, (siehe Abbildung 4.5, „Zerlegungsfunktionen 2. Rekursionsstufe") jeweils als Summe der Bits aus den Symmetriemengen interpretieren. (Beispielsweise werden $y_2^{(3)}$, $y_1^{(4)}$ und $y_0^{(5)}$ durch einen Volladdierer zu der 2–Bit–Zahl $(z_1^{(5)}, z_0^{(5)})$ addiert.) Es ergeben sich wiederum 7 verschiedene Zahlen, deren Summe das Endergebnis der Multiplikation $(r_7, \ldots, r_0)$ darstellt. Auf der 2. Rekursionsstufe wurde folgende Addition durchgeführt:

$$
\begin{array}{cccccccc}
 & y_1^{(5)} & y_2^{(3)} & y_1^{(3)} & y_1^{(2)} & y_1^{(1)} & y_0^{(1)} & y_0^{(0)} \\
 & y_0^{(6)} & y_1^{(4)} & y_0^{(4)} & y_0^{(3)} & y_0^{(2)} & & \\
 & & y_0^{(5)} & & & & & \\
\hline
 & & & & & & & z_0^{(0)} \\
 & & & & & & z_0^{(1)} & \\
 & & & & z_1^{(2)} & z_0^{(2)} & & \\
 & & & z_1^{(3)} & z_0^{(3)} & & & \\
 & & z_1^{(4)} & z_0^{(4)} & & & & \\
 & z_1^{(5)} & z_0^{(5)} & & & & & \\
z_1^{(6)} & z_0^{(6)} & & & & & & \\
\hline
r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0
\end{array}
$$

Wieder läßt sich durch „Verschieben“ von Bits gleicher Stelligkeit die verbleibende Aufgabe als Addition zweier Zahlen beschreiben:

$$
\begin{array}{cccccccc}
z_1^{(6)} & z_1^{(5)} & z_1^{(4)} & z_1^{(3)} & z_1^{(2)} & z_0^{(2)} & z_0^{(1)} & z_0^{(0)} \\
 & z_0^{(6)} & z_0^{(5)} & z_0^{(4)} & z_0^{(3)} & & & \\
\hline
r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0
\end{array}
$$

Die Addition dieser beiden Zahlen wird durch die Zusammensetzungsfunktion der 2. Rekursionsstufe aus Abbildung 4.5 realisiert. Da hier keine Symmetriemengen der Mächtigkeit größer als 2 auftreten, erfolgt eine zweiseitige Zerlegung. Die Zusammensetzungsfunktion der 2. Rekursionsstufe wird zunächst hinsichtlich $\{\{z_1^{(6)}, z_0^{(6)}, z_1^{(5)}, z_0^{(5)}, z_1^{(4)}\}, \{z_0^{(4)}, z_1^{(3)}, z_0^{(3)}, z_1^{(2)}, z_0^{(2)}, z_0^{(1)}, z_0^{(0)}\}\}$ zerlegt. Die Realisierung der Zusammensetzungsfunktion der 2. Rekursionsstufe entspricht *im wesentlichen* der eines Addierers nach dem Conditional–Sum–Prinzip.

Ein entscheidender Punkt bei der Synthese des partiellen Multiplizierers wurde bisher noch nicht erwähnt:
Die auftretenden Zusammensetzungsfunktionen sind *partielle* Funktionen. Beispielsweise treten bei der Addition von $p_{3,0}$, $p_{2,1}$, $p_{1,2}$ und $p_{0,3}$ auf der ersten Rekursionsstufe nur nur 5 mögliche Summenwerte auf. Es werden jedoch 3 Zerlegungsfunktionen benötigt, so daß bei der Addition 3 Ausgabevektoren nicht vorkommen können und bei der Zusammensetzungsfunktion folglich don't cares auftreten.
Anhand dieses Beispiels läßt sich sehr gut die Bedeutung der in den Kapiteln 2 und 3 entwickelten Verfahren zur Belegung von don't cares demonstrieren. Die Güte des Ergebnisses hängt *entscheidend* von der Belegung der don't cares ab.
Anzahl und Größe der Symmetriemengen hängen wesentlich davon ab, wie die vorhandenen don't cares belegt werden.
Auch der Einsatz der Verfahren zur don't care–Belegung aus Kapitel 3 für die Zusammensetzungsfunktion der 2. Rekursionsstufe ist in hohem Maße für die Güte des generierten Schaltkreises verantwortlich.

Verzichtet man auf eine gezielte Ausnutzung von don't cares und belegt beispielsweise sämtliche don't care–Stellen mit 0, dann ergibt sich anstatt der hier gefundenen Realisierung mit einer R_2–Komplexität von 110 eine Realisierung mit R_2–Komplexität von 194.
Die oben angegebene Interpretation der Zusammensetzungsfunktion der 2. Rekursionsstufe als Addition einer 7–Bit– und einer 4–Bit–Zahl ist nur *eine* Möglichkeit mit einer ganz speziellen Belegung der auftretenden don't cares. Bei näherer Betrachtung der Realisierung aus Abbildung 4.5 wird man auch erkennen, daß die Funktion, die aus der vom Verfahren gewählten don't care–Belegung hervorgeht, nicht exakt mit dem Addierer übereinstimmt. Die Funktion r_7, die das höchstwertigste Bit berechnet, entspricht nicht exakt der entsprechenden Funktion des Addierers. Dieser Unterschied führt sogar zu einer Kostenverringerung im Vergleich zum Addierer: Die gefundene Realisierung für die Zusammensetzungsfunktion hat eine R_2–Komplexität von 41. Würden die don't cares so belegt, daß die Zusammensetzungsfunktion der 2. Rekursionsstufe mit dem oben erwähnten Addierer übereinstimmt, so würde das Zerlegungsverfahren eine nach dem Conditional–Sum–Prinzip aufgebaute Realisierung mit R_2–Komplexität 45 berechnen.

Der automatische Entwurf partieller Multiplizierer fester Bitbreite durch das entwickelte Logiksynthesewerkzeug und die Analyse der gefundenen Realisierungen legt nun ein allgemeines Konstruktionsprinzip für Multiplizierer variabler Bitbreiten nahe:

Verallgemeinerung auf variable Bitbreiten Ein partieller n–Bit–Multiplizierer bildet die Summe von n Partialprodukten $2^j \cdot (p_{n-1,j}, \ldots, p_{0,j})$ $(0 \leq j \leq n-1)$. Es ist also folgende Addition durchzuführen:

$$\begin{array}{cccccccccc}
 & & & & & p_{n-1,0} & p_{n-2,0} & \cdots & p_{1,0} & p_{0,0} \\
 & & & & p_{n-1,1} & p_{n-2,1} & p_{n-3,1} & \cdots & p_{0,1} & \\
 & & & & \vdots & \vdots & \vdots & & & \\
 & & p_{n-1,n-2} & \cdots & p_{2,n-2} & p_{1,n-2} & p_{0,n-2} & & & \\
 & p_{n-1,n-1} & p_{n-2,n-1} & \cdots & p_{1,n-1} & p_{0,n-1} & & & & \\
\hline
r_{2n-1} & r_{2n-2} & r_{2n-3} & \cdots & r_n & r_{n-1} & r_{n-2} & \cdots & r_1 & r_0
\end{array}$$

pm_n ist symmetrisch in den Variablen, die in gleichen Spalten der obigen Multiplikationsmatrix stehen. (Denn es ist klar, daß man die entsprechenden Bits vertauschen kann, ohne das Ergebnis der Addition zu ändern.)
Geht man vor wie oben beschrieben, so werden somit in einer 1. Stufe die Bits in den einzelnen Spalten der Multiplikationsmatrix getrennt aufaddiert. Da in einer Spalte höchstens n Einträge stehen, erhält man also $2n$ Binärzahlen mit Maximallänge $\lceil \log(n+1) \rceil$.
Da diese Binärzahlen mit ihrem niederwertigsten Bit alle in verschiedenen Spalten der Matrix beginnen, kann man (wie oben anhand von pm_4 dargestellt) durch „Verschieben" von Bits daraus $\lceil \log(n+1) \rceil$ Binärzahlen mit Maximallänge $2n$ bilden, die noch aufaddiert werden müssen, um das Endresultat zu erhalten. Man erhält also eine Matrix mit $\lceil \log(n+1) \rceil$ Zeilen der Länge $2n$.
Danach fährt man in einer 2. Stufe mit dem spaltenweisen Addieren von Bits fort und

Bitbreite	Stufen
3	1
4 – 7	2
8 – 127	3
128 – $(2^{127} - 1)$	4

Tabelle 4.2: Anzahl der benötigten Additionsstufen bei pm_n für verschiedene Bitbreiten n.

erhält eine Matrix mit $\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil$ Zeilen und $2n$ Spalten, bei der wiederum die Summe der Binärzahlen in den einzelnen Zeilen das Endergebnis liefert.
Dies wird so lange fortgeführt, bis man nach $O(\log^\star n)$ Stufen nur noch 2 Binärzahlen der Länge $2n$ zu addieren hat.[1]
Die Addition dieser beiden Binärzahlen erfolgt dann durch einen Conditional–Sum–Addierer.

Im folgenden sollen Kosten und Tiefe einer solchen Realisierung für pm_n abgeschätzt werden:

Zunächst wird die Anzahl der Additionsstufen abgeschätzt, die benötigt werden, bis man schließlich bei 2 Binärzahlen angelangt ist:
Allgemein gilt: Sind in Stufe i noch n_i Binärzahlen zu addieren, so sind in Stufe $i+1$ dann $\lceil \log(n_i + 1) \rceil$ Binärzahlen zu addieren.
Aus den Beziehungen

$$\begin{aligned} \lceil \log(\lceil \log(n+1) \rceil + 1) \rceil &\leq \log(\lceil \log(n+1) \rceil) + 1 \\ &\leq \log((\log n) + 1) + 1 \end{aligned}$$

und

$$\log((\log n) + 1) + 1 \quad \leq \log n \text{ für } n \geq 8$$

kann man folgern, daß nach spätestens $2\log^\star(n)$ Stufen n Binärzahlen zu 2 Binärzahlen reduziert sind.
In Tabelle 4.2 ist für verschiedene Bitbreiten die Anzahl der benötigten Additionsstufen angegeben. Man erkennt, daß man (auch in Zukunft) für praktisch realisierbare Bitbreiten nicht mehr als 4 Additionsstufen benötigen wird.

Die Addition von n 1–Bit–Zahlen kann bei Verwendung eines Divide–and–conquer–Verfahrens mit zweiseitigen, gleichmächtigen Zerlegungen mit einer linearen Anzahl von 2–Input–Gattern und Tiefe $O(\log n \cdot \log\log n)$ erfolgen. Verwendet man einen Schaltkreis BS_n aus

[1] $\log^\star$ ist hierbei folgendermaßen definiert: Mit $\log^{(0)}(n) := n$ und $\log^{(i)}(n) := \log(\log^{(i-1)}(n))$ für $i \in \mathbf{N}$ gilt $\log^\star n := \min\{m \mid \log^{(m)}(n) \leq 1\}$.

[Weg89] zur Addition der n 1–Bit–Zahlen, so kann man mit $C_{B_2}(BS_n) = 8\frac{1}{3}n + O(\log^2 n)$ 2–Input–Gattern und Tiefe $D_{B_2}(BS_n) = 3\log_{\frac{3}{2}} n + O(\log\log n) \leq 5.13\log n + O(\log\log n)$ auskommen.

Die Kosten für die Realisierung von pm_n lassen sich dann wie folgt abschätzen:
Die Kosten der ersten Stufe werden abgeschätzt durch

$$\begin{aligned}
C_{B_2}(1.\ \text{Stufe}) &= C_{B_2}(BS_n) + 2 \cdot \sum_{i=2}^{n-1} C_{B_2}(BS_i) \\
&\leq 8\frac{1}{3} \cdot [n + 2 \cdot \sum_{i=2}^{n-1} i] + O(n\log^2 n) \\
&= 8\frac{1}{3} \cdot [n + 2 \cdot \frac{(n-2)(n+1)}{2}] + O(n\log^2 n) \\
&= 8\frac{1}{3} \cdot [n^2 - 2] + O(n\log^2 n) \\
&= 8\frac{1}{3} \cdot n^2 + O(n\log^2 n).
\end{aligned}$$

Die Kosten für die restlichen Stufen lassen sich durch $O(n\log^2 n)$ abschätzen:
Ist n_i die Anzahl der zu summierenden Binärzahlen auf Stufe i, so lassen sich die Kosten auf Stufe i abschätzen durch

$$2n \cdot C_{B_2}(BS_{n_i}) = \underbrace{16\frac{2}{3}n \cdot n_i}_{K_1(n_i)} + \underbrace{O(n\log^2 n_i)}_{K_2(n_i)}.$$

Ist $s \leq 2\log^\star(n)$ die Anzahl der benötigten Stufen, so gilt

$$\begin{aligned}
\sum_{i=2}^{s} K_2(n_i) &= O(s \cdot n \cdot (\log\log n)^2) \\
&= O(n \cdot \log^\star n \cdot (\log\log n)^2) \\
&= O(n \cdot \log^2 n).
\end{aligned}$$

Wegen $\lceil\log(m+1)\rceil \leq \frac{m}{2}$ für $m \geq 6$ gilt

$$\begin{aligned}
\sum_{i=2}^{s} K_1(n_i) &= 16\frac{2}{3}n \cdot (\lceil\log(n+1)\rceil + \lceil\log(\lceil\log(n+1)\rceil + 1)\rceil + \ldots) \\
&\leq 16\frac{2}{3}n \cdot (\lceil\log(n+1)\rceil + \frac{1}{2}\lceil\log(n+1)\rceil + \frac{1}{4}\lceil\log(n+1)\rceil + \ldots) + O(n) \\
&\leq 33\frac{1}{3}n \cdot \lceil\log(n+1)\rceil + O(n) \\
&= 33\frac{1}{3}n \cdot \log n + O(n)
\end{aligned}$$

$$= O(n \cdot \log^2 n).$$

Die Gesamtkosten für die Stufen 2 bis s betragen also tatsächlich $O(n \log^2 n)$. Folglich betragen die Kosten zur Reduktion der n Binärzahlen zu 2 Binärzahlen

$$8\frac{1}{3} \cdot n^2 + O(n \log^2 n).$$

Die Tiefe dieses Schaltkreises ergibt sich als

$$\begin{aligned}
\sum_{i=1}^{s} D_{B_2}(BS_{n_i}) &= (\sum_{i=1}^{s} 3 \log_{\frac{3}{2}} n_i) + O(\log^\star n \log\log n) \\
&\le 5.13 \cdot (\log n + \log(\lceil \log(n+1) \rceil) + \ldots) + O(\log^\star n \log\log n) \\
&\le 5.13 \cdot (\log n + \log((\log n) + 1) + \ldots) + O(\log^\star n \log\log n) \\
&= 5.13 \cdot \log n + O(\log^\star n \log\log n)
\end{aligned}$$

Im Vergleich dazu betragen die Kosten für einen Wallace–Tree–Multiplizierer [Wal64] (siehe auch [KMO89]) mit Bitbreite n für $n = 2^k$ (ebenfalls nur für den Reduktionsteil ohne den abschließenden Addierer)

$$(\frac{n}{2} - 1) \cdot 2n \cdot C_{B_2}(\mathit{CSA4to2})^\dagger = (\frac{n}{2} - 1) \cdot 2n \cdot 10 = 10n^2 - 20n.$$

Die Tiefe beträgt hierbei $5 \log n - 5$.
Kosten und Tiefe der obigen Realisierung sind also in derselben Größenordnung wie beim Wallace–Tree–Multiplizierer, wobei die Konstante bei der Gatteranzahl der obigen Realisierung etwas niedriger ist und die Konstante bei die Tiefe etwas höher ist.

Entwurf eines fehlererkennenden Dividierers

Das implementierte Logiksynthesewerkzeug ist in das Entwurfssystem CADIC [BHK+87] integriert, das auf einem in [Hot65] entwickelten Kalkül basiert und eine Beschreibung von Schaltungen mit rekursiven, parametrisierten Gleichungen zuläßt. Belegt man die Parameter der Gleichungen mit festen Werten, so lassen sich die Gleichungen zu einem Schaltkreis fester Größe „expandieren“. Ein System von Gleichungen definiert auf diese Weise nicht nur einen einzelnen Schaltkreis, sondern eine ganze Schaltkreisfamilie. Damit eignet sich CADIC in besonderer Weise zur kompakten, hierarchischen Beschreibung regulärer Schaltkreise.

† *CSA4to2* ist ein Reduktionsbaustein, der aus 2 Volladdierern besteht.

Anhand des Entwurfs eines fehlererkennenden Dividierers [TY88] wird demonstriert, daß die automatische Logiksynthese nicht unbedingt in Konkurrenz zu Handentwürfen regelmäßiger Schaltungen zu sehen ist, sondern auch in Verbindung mit solchen Entwürfen nutzbringend eingesetzt werden kann.

Es erfolgte ein parametrisierter Entwurf für einen n–Bit–Dividierer (in einer redundanten Zahlendarstellung) mit dem Entwurfssystem CADIC. Der Dividierer ist regelmäßig unter Verwendung bestimmter Grundmodule aufgebaut. In Abbildung 4.6 ist für die Bitbreite $n = 16$ die Expansion des entworfenen n–Bit–Dividierers bis zur Ebene der Grundmodule DU, M und L angegeben. Die Grundmodule sind so spezifiziert, daß sowohl zur Bestimmung der Quotientenbits als auch zu der nötigen Addition bzw. Subtraktion in der redundanten Zahlendarstellung pro Zeile des Dividiererfeldes nur konstante Laufzeit benötigt wird. Die Spezifikation erfolgte durch Tabellen, die allerdings nicht dazu geeignet waren, eine Idee für einen Handentwurf der Grundmodule zu liefern. Aus diesem Grund wurde eine Realisierung für diese Grundmodule mit Hilfe des Logiksynthesewerkzeuges automatisch generiert. Abbildung 4.7 zeigt den Dividierer (hier wiederum für Bitbreite 16), bei dem die Grundmodule DU, M und L jeweils durch ihre automatisch generierte Realisierung ersetzt sind.

Das vorliegende Beispiel zeigt exemplarisch, wie die hier entwickelten Logiksyntheseverfahren als Teilkomponente im Rahmen eines Entwurfsablaufs genutzt werden können.

Benchmarkresultate

Abschließend werden noch Resultate der Logiksynthese für Benchmarkschaltungen angegeben.

Gatterrealisierungen

Zum Vergleich mit *sis* [BRS+87, S+92] wurden für Benchmarkschaltungen mit beiden Werkzeugen Realisierungen mit 2–Input–Gattern generiert. Für die resultierenden Schaltungen wurden dann mit dem Tool *TimberWolf* (integriert in das System *octtools* der Universität Berkeley) Layouts generiert.

Tabelle 4.3 zeigt die Ergebnisse für Layoutgrößen und Signalverzögerungen der resultierenden Schaltungen sowohl für unser Werkzeug, das im folgenden *mulopII* genannt wird, als auch für *sis*. Obwohl es auch Beispiele wie *cm151a* und *cm162a* gibt, die nicht unbedingt für eine disjunkte Zerlegung geeignet erscheinen, übertrifft das hier vorgestellte Werkzeug *mulopII* *sis* in fast drei Viertel der Fälle. Insgesamt sind die Layoutgrößen von *sis* um rund 76 % schlechter. Trotzdem sind bei zwei Drittel der Benchmarks die Signalverzögerungen der Schaltungen, die durch unser Werkzeug *mulopII* generiert wurden, nicht größer oder geringer als die Signalverzögerungen der entsprechenden Schaltkreise von *sis*.

Eine der Motivationen für die Durchführung kommunikationsminimaler Zerlegungen war die Minimierung globaler Verbindungen in Hinblick auf die Layoutgenerierung. Es ist zu

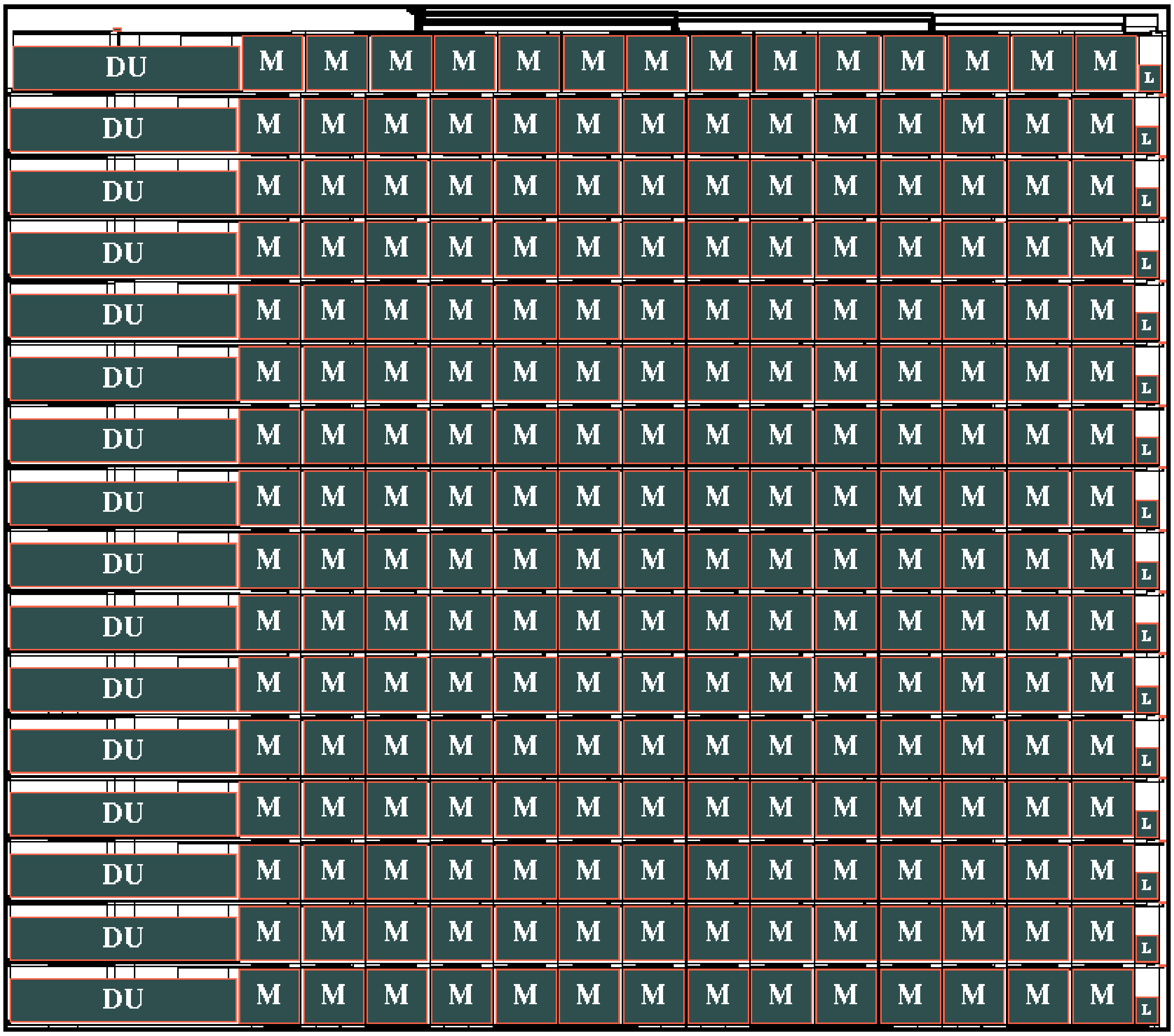


Abbildung 4.6: Mit dem Entwurfssystem CADIC entworfener 16–Bit–Dividierer.

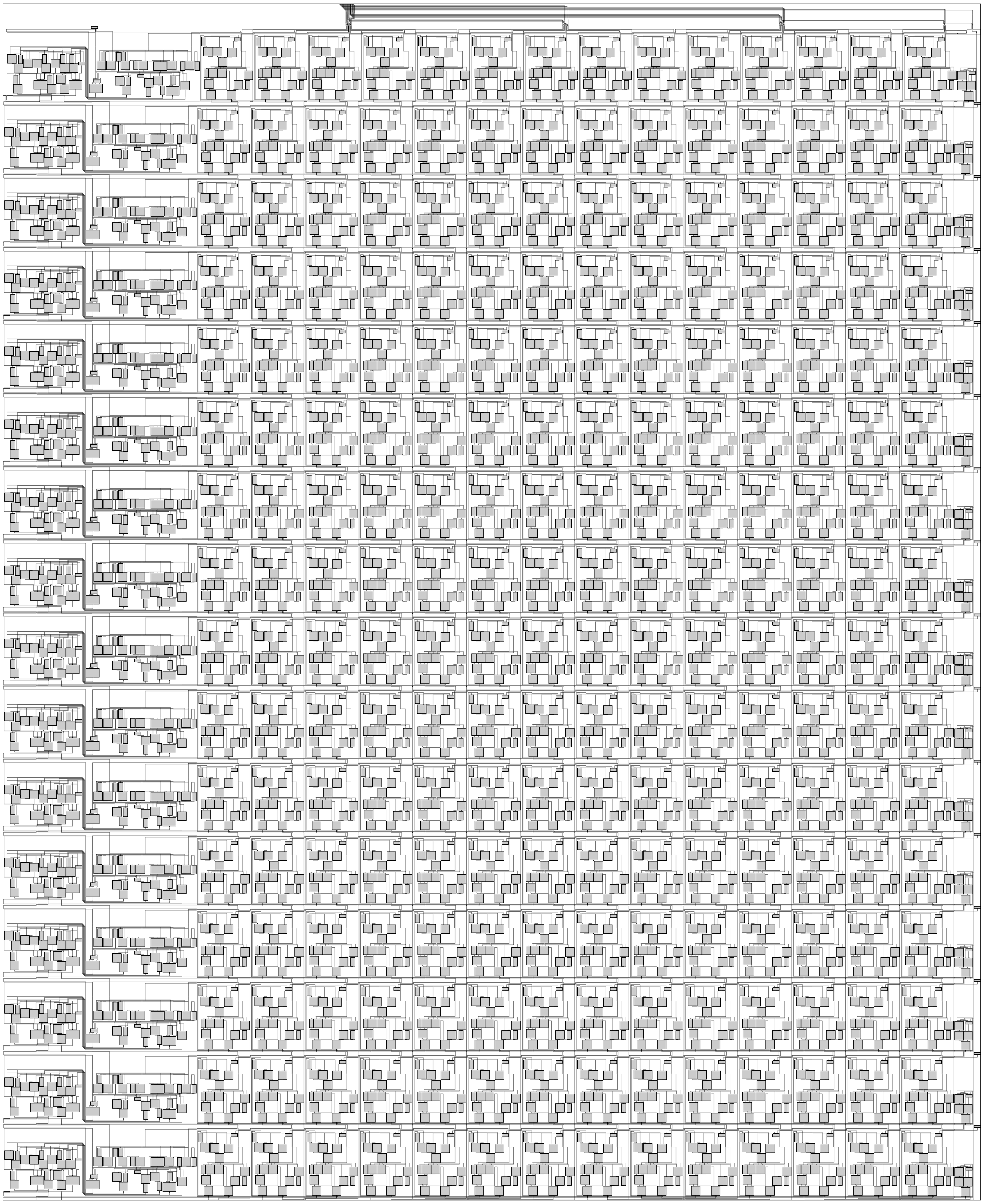


Abbildung 4.7: 16–Bit–Dividierer (Schaltungen DU, M und L jeweils ersetzt durch automatisch generierte Realisierung).

	Layoutgröße			Signalverzögerung		
Schaltkreis	*sis*	*mulopII*	Verhältnis	*sis*	*mulopII*	Verhältnis
9symml	1194336	201400	5.93	27.6	13.6	2.03
C17	28800	25704	1.12	4.2	4.2	1.00
cm138a	103896	87480	1.19	5.8	6.8	0.85
cm151a	95312	140728	0.68	12.6	19.2	0.66
cm152a	85536	106704	0.80	10.0	13.2	0.76
cm162a	131976	178296	0.74	12.0	12.4	0.97
cm163a	144008	140728	1.02	13.0	9.4	1.38
cm82a	74784	61320	1.22	7.2	7.0	1.03
cm85a	165456	170288	0.97	10.2	13.0	0.78
cmb	204792	120624	1.70	9.4	6.6	1.42
decod	140448	119496	1.18	6.2	5.0	1.24
f51m	561184	251392	2.23	51.0	18.4	2.77
majority	42200	39168	1.08	7.8	6.6	1.18
parity	99408	96976	1.03	5.0	5.0	1.00
z4ml	156288	93800	1.67	16.2	9.8	1.65
$\sum$	3228K	1834K	1.76	198.2	150.2	1.32

Tabelle 4.3: Vergleich zwischen unserem Werkzeug *mulopII* und *sis* hinsichtlich Layoutgröße und Signalverzögerung.

hoffen, daß sich durch die Verwendung eines Layouttools, das in der Lage ist, die hierarchische Struktur des zerlegten Schaltkreises zu nutzen, die Layoutgrößen noch weiter verbessern lassen.

Aber auch hier lassen sich schon Auswirkungen der Kommunikationsminimierung erkennen: In Abbildung 4.8 sind Layouts der Schaltkreise dargestellt, die von *sis* und unserem Werkzeug für die Benchmarkschaltung *9symml* generiert wurden. Man erkennt, daß bei unserer Realisierung nicht nur die Anzahl der Grundzellen wesentlich kleiner ist, sondern auch die Höhe der Verdrahtungskanäle wesentlich niedriger ist.

Abschließend werden zur Motivation der Durchführung der Zerlegungen auf der Basis von ROBDDs die Laufzeiten der ROBDD–basierten Version *mulopII* unseres Werkzeuges mit denen der früheren tabellenbasierten Version *mulop* verglichen. Anhand von Tabelle 4.4 erkennt man eine beträchtliche Verringerung der Laufzeiten.

In der letzten Spalte der Tabelle 4.4 ist noch der Anteil der Laufzeit angegeben, der bei der ROBDD–basierten Version für Lösungen des Problems CDF aus Kapitel 3.4, d.h. zur Berechnung gemeinsamer Zerlegungsfunktionen, investiert wird. Man sieht, daß eine sehr effiziente Möglichkeit zur Berechnung gemeinsamer Zerlegungsfunktionen mehrerer Ausgangsfunktionen gefunden wurde.

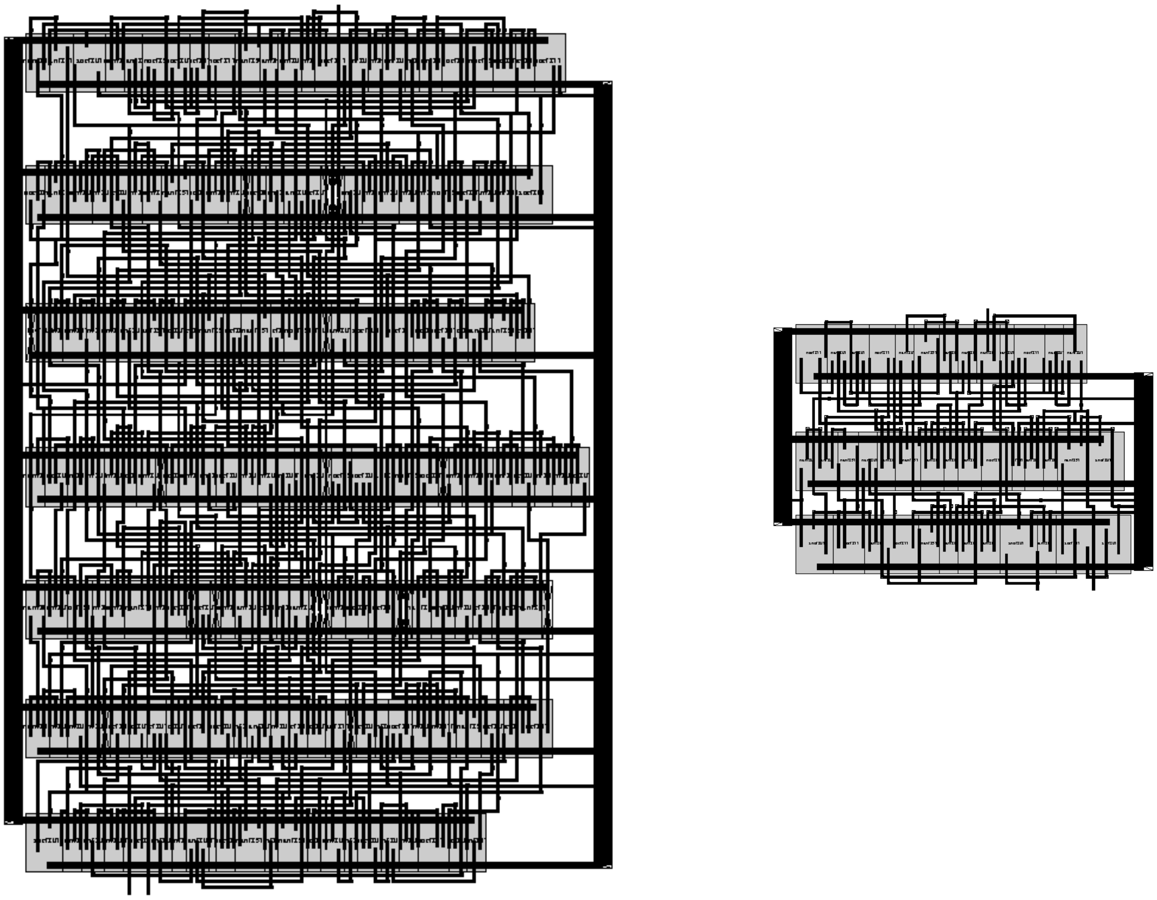


Abbildung 4.8: Layouts für Benchmarkschaltkreis *9symml*. Die linke Schaltung wurde durch *sis* erzeugt, die rechte durch unser Werkzeug *mulopII*.

FPGA–Realisierungen

Zerlegungsverfahren sind besonders gut geeignet für die Synthese von look-up table–basierten FPGAs (= **f**ield **p**rogrammable **g**ate **a**rrays). FPGAs sind vorgefertigte Schaltungen, die durch den Benutzer programmiert werden können. Sie haben wegen ihrer schnellen und flexiblen Einsetzbarkeit in den letzten Jahren (speziell beim Bau von Prototypen) immer größere Bedeutung erhalten. Solche look-up table–basierte FPGAs sind in der Lage, Funktionen mit einem Ausgang bis zu einer bestimmten Grenze b von Eingängen durch eine Funktionstabelle (**l**ook-**u**p **t**able = LUT) zu realisieren.

Insofern ist die Anpassung des rekursiven Zerlegungsverfahrens auf die FPGA–Synthese sehr leicht: Die Rekursion muß lediglich bei Funktionen mit höchstens b Eingängen abgebrochen werden. Zusätzlich wird die Funktion zur Kostenabschätzung einer Funktion mit

Schaltkreis	Laufzeit *mulop*	Laufzeit *mulopII*	Verhältnis	Anteil der Laufzeit für CDF (*mulopII*)
9symml	1.40 sec	1.23 sec	1.14	0.69%
C17	0.32 sec	0.15 sec	2.13	0.01%
cm138a	1.01 sec	0.18 sec	5.61	0.89%
cm151a	4.16 sec	1.09 sec	3.82	0.13%
cm152a	2.15 sec	0.50 sec	4.30	0.36%
cm162a	350.65 sec	3.32 sec	105.62	0.26%
cm163a	2923.31 sec	2.35 sec	1243.96	0.08%
cm82a	0.38 sec	0.21 sec	1.81	0.01%
cm85a	7.46 sec	3.73 sec	2.00	0.27%
cmb	1836.13 sec	2.52 sec	728.62	0.05%
decod	26.15 sec	2.56 sec	10.21	1.93%
f51m	3.14 sec	1.83 sec	1.71	0.28%
majority	0.44 sec	0.08 sec	5.50	0.01%
parity	111.06 sec	1.37 sec	81.07	0.00%
z4m1	0.66 sec	0.76 sec	0.87	0.16%

Tabelle 4.4: Vergleich der Laufzeiten der ROBDD–basierten Version *mulopII* unseres Werkzeugs und der früheren tabellenbasierten Version *mulop*.

n Eingängen aus den Abschnitten 3.1.4 bzw. 3.4.1 an die FPGA–Synthese angepaßt.

Die hier angegebenen Experimente sind durchgeführt für den FPGA–Typ *Xilinx XC3000*. Diese FPGAs können Funktionen mit bis zu $b = 5$ Eingängen durch eine LUT (look-up table) realisieren.[3] Der angegebene FPGA–Typ hat die zusätzliche Eigenschaft, daß statt einer Funktion mit 5 Eingängen auch 2 Funktionen mit 4 Eingängen durch ein CLB (= **c**onfigurable **l**ogic **b**lock) realisiert werden können, sofern die Gesamtzahl der *verschiedenen* Eingänge beider Funktionen 5 nicht übersteigt.

Tabelle 4.5 enthält einen Vergleich einer früheren Version des Synthesewerkzeuges aus [SM95b], in das die Ausnutzung von don't cares noch nicht integriert war, mit der jetzigen Version mit Ausnutzung von don't cares. Die in der Tabelle angegebenen Zahlenwerte entsprechen den Anzahlen der benötigten CLBs bei *XC3000*–FPGAs. Die Benchmarkfunktionen, auf die die Logiksynthese angewendet wurde, sind *totale* Funktionen. Partielle Funktionen entstehen nur im Verlauf des Verfahrens auf höheren Rekursionsstufen. Trotzdem erkennt man teilweise noch beträchtliche Verbesserungen bei der Version mit don't care–Ausnutzung (z.B. bei *alu2* eine Verbesserung um über 35 %). Speziell bei kleineren Beispielen ist aber teilweise auch keine Veränderung zu erkennen, da sich ausgehend von den totalen Funktionen (wegen geringer Rekursionstiefe) keine großen don't care–Mengen ansammeln.

[3] Die Funktion zur Kostenabschätzung einer Funktion mit n Eingängen wurde in diesen Experimenten als $cost(n) = 1$ für $n \leq 5$ und $cost(n) = 0.15n^2$ für $n > 5$ gewählt.

Schaltkreis	Anzahl Inputs	Anzahl Outputs	Anzahl von CLBs: keine dc–Ausnutzung	Anzahl von CLBs: dc–Ausnutzung
5xp1	7	10	9	9
9sym	9	1	7	7
alu2	10	6	51	33
apex7	49	37	45	41
b9	41	21	30	28
C499	41	32	65	50
C880	60	26	87	71
clip	9	5	14	13
count	35	16	26	26
duke2	22	29	114	108
e64	65	65	55	55
f51m	8	8	8	8
misex1	8	7	9	8
misex2	25	18	24	24
rd73	7	3	5	5
rd84	8	4	8	8
rot	135	107	146	135
sao2	10	4	20	18
vg2	25	8	18	18
z4ml	7	4	4	4
Summe			745	669

Tabelle 4.5: Vergleich der CLB–Anzahlen der Version des Algorithmus ohne don't care–Ausnutzung aus [SM95b] mit den CLB–Anzahlen der jetzigen Version mit don't care–Ausnutzung.

Bei den CLB–Anzahlen des vorliegenden Synthesewerkzeuges ist allerdings zu beachten, daß die angesprochene Zusammenfassung von LUTs zu CLBs bei unserem Werkzeug nur mit einer Heuristik erfolgt ist. Das Problem der Zusammenfassung von LUTs zu CLBs läßt sich aber auch exakt durch ganzzahlige lineare Programmierung [MNS+90] lösen.[4]

In Tabelle 4.6 werden schließlich die Resultate mit den Ergebnissen aktueller Arbeiten zur FPGA–Synthese verglichen. Hierbei sind *FGMap* [LPPS93] und *mis-pga (new)* [MSBS91] Werkzeuge, die bei der Zerlegung *keine* gemeinsamen Zerlegungsfunktionen berechnen. *IMODEC* [WEA95] ist ein an der TU München entwickeltes Verfahren, das Zerlegungen für Funktionen mit mehreren Ausgängen benutzt und in das wesentliche Ideen aus [SM93] und [MS94] eingegangen sind.

Es überrascht auf den ersten Blick etwas, daß die Ergebnisse der Verfahren *mis-pga (new)* und *FGMap* nicht noch schlechter sind, da sie keine gemeinsamen Zerlegungsfunktionen berechnen, um Logiksharing zu erreichen. Der Grund dafür liegt aber in der Tatsache, daß die beiden Tools auf schon vorstrukturierten Schaltkreisen als Eingabe aufsetzen und

[4] Eine Implementierung der Verfahren aus [MNS+90] steht uns bisher leider noch nicht zur Verfügung.

	Anzahl	Anzahl	Anzahl von CLBs			
Schaltkreis	Inputs	Outputs	*mulopII*	*FGMap*	*mis-pga(new)*	*IMODEC*
5xp1	7	10	9	15	13	9
9sym	9	1	7	7	7	7
alu2	10	6	33	53	96	46
apex7	49	37	41	47	43	41
b9	41	21	28	27	32	-
C499	41	32	50	49	66	50
C880	60	26	71	74	72	81
clip	9	5	13	20	23	12
count	35	16	26	24	30	26
duke2	22	29	108	178	94	122
e64	65	65	55	55	56	55
f51m	8	8	8	11	15	8
misex1	8	7	8	8	9	9
misex2	25	18	24	21	25	21
rd73	7	3	5	7	5	5
rd84	8	4	8	12	9	8
rot	135	107	135	194	143	127
sao2	10	4	18	27	28	17
vg2	25	8	18	23	18	19
z4ml	7	4	4	5	4	4
Teilsumme			641	830	756	667
Summe			669	857	788	-

Tabelle 4.6: Experimentelle Resultate für *XC3000*-Typ

auf dieser Beschreibung weiterrechnen. Insofern ist also Logiksharing meist schon in der Eingabe an diese Algorithmen realisiert, so daß sich der angesprochene Mangel nicht so gravierend auswirkt, wie man zunächst annehmen würde.

Zusammenfassung

In dieser Arbeit werden Logiksyntheseverfahren entwickelt, die auf der rekursiven Zerlegung Boolescher Funktionen beruhen.

Wesentliche Merkmale der Algorithmen sind die Identifizierung mehrfachverwendbarer Teillogik, die Ausnutzung von Struktureigenschaften wie Symmetrien und die Ausnutzung von don't cares partieller Funktionen.

Der Erfolg der gewählten Vorgehensweise wird anhand experimenteller Resultate demonstriert.

Auch bei Problemstellungen, die schon intensiv unter Einsatz *menschlicher* Intelligenz bearbeitet wurden, konnten mit Hilfe des hier vorgestellten *automatischen* Logiksyntheseverfahrens konkurrenzfähige Entwürfe erzielt werden.

Anhand von Beispielen erkennt man sogar, daß es denkbar ist, Realisierungen, die durch das automatische Logiksynthesewerkzeug erzeugt wurden, als Anregung für parametrisierte Entwürfe (d.h. Entwürfe mit variabler Bitbreite) zu nutzen.

Summary

In this thesis we develop methods for logic synthesis by recursive functional decomposition.

Essential features of the algorithms are the identification of reusable subcircuits to achieve as much logic sharing as possible, the exploitation of structural properties of boolean functions like symmetries, and the exploitation of don't care sets of incompletely specified boolean functions.

The success of the chosen approach is shown by experimental results.

Even for problems, which were already studied intensively using *human* intelligence, we could achieve competitive designs by means of our *automatic* logic synthesis procedure.

By an analysis of the realizations produced by our synthesis tool it is even possible to get an idea how to derive regular parametric designs (i.e. designs with variable numbers of inputs).

Anhang A

Komplexität von CDF

Hier wird Satz 3.10 bewiesen: Das Problem **CDF** ist NP–hart.

Der Satz wird bewiesen durch Angabe einer Polynomzeittransformation vom NP–vollständigen Problem 3–PARTITION nach CDF.

Das Problem 3–PARTITION lautet (siehe [GJ79] oder [Meh84b]):

Problem 3P (3–PARTITION)

Gegeben: Gewichte $a_1, \ldots, a_{3n} \in \mathbf{Z}^+$, eine Grenze $B \in \mathbf{Z}^+$, so daß $B/4 < a_i < B/2\ \forall i$ und $\sum_{i=1}^{3n} a_i = n \cdot B$.

Gesucht: Gibt es eine Partition $S_1, \ldots, S_n$ von $\{1, \ldots, 3n\}$, so daß $\sum_{j \in S_i} a_j = B\ \forall i$?

Satz A.1 *Das Problem* **3–PARTITION** *ist NP–vollständig im strengen Sinne, d.h es ist auch NP–vollständig, wenn die Zahlen der Eingabe unär kodiert sind.*

Beweis: Beweis durch Transformation von 3DM, siehe [GJ79], S. 96ff. □

Die Polynomzeittransformation zum Beweis, daß CDF NP–hart ist, erfolgt in 2 Stufen. Zunächst wird gezeigt, daß ein Problem 2^l–PARTITION NP–vollständig ist, das folgendermaßen definiert ist:

Problem 2^l–PARTITION

Gegeben: Ganze Zahlen $c_1, \ldots, c_k \in \mathbf{Z}^+$, eine Grenze $B' = 2^b$ mit $b \in \mathbf{N}$ und eine Zahl $M = 2^l$ mit $l \in \mathbf{N}$, so daß $\sum_{i=1}^{k} c_i = M \cdot B' = 2^{b+l}$ $(k > M)$.

Gesucht: Gibt es eine Partition $T_1, \ldots, T_M$ von $\{1, \ldots, k\}$, so daß $\sum_{j \in T_i} c_j = B'$ für $1 \leq i \leq M$?

Satz A.2 *Das Problem* 2^l–**PARTITION** *ist* NP*–vollständig im strengen Sinne, d.h. es ist auch* NP*–vollständig, wenn die Zahlen der Eingabe unär kodiert sind.*

Beweis: Hier wird nur bewiesen, daß 2^l–PARTITION NP–hart ist. Der Beweis erfolgt durch Polynomzeittransformation von 3–PARTITION nach 2^l–PARTITION.

Sei eine Instanz von 3–PARTITION gegeben durch $a_1, \ldots, a_{3n}, B$.
Dazu wird (in Polynomzeit) eine Instanz von 2^l–PARTITION berechnet mit

$$\begin{aligned}
B' &= 2^{\lceil \log B \rceil + 2} \quad (\text{d.h. } b = \lceil \log B \rceil + 2) \\
k &= 3n + 2^{\lceil \log n \rceil} \\
c_i &= a_i \quad \forall\, 1 \le i \le 3n \\
c_i &= B' - B = 2^{\lceil \log B \rceil + 2} - B \quad \forall\, 3n < i \le 4n \\
c_i &= B' \quad \forall\, 4n < i \le 3n + 2^{\lceil \log n \rceil} \\
M &= 2^{\lceil \log n \rceil} \quad (\text{d.h. } l = \lceil \log n \rceil)
\end{aligned}$$

Es gilt dann:

$$\begin{aligned}
\sum_{i=1}^{k} c_i &= \sum_{i=1}^{3n} a_i + \sum_{i=3n+1}^{4n} (B' - B) + \sum_{i=4n+1}^{3n+2^{\lceil \log n \rceil}} B' \\
&= n \cdot B + n \cdot (B' - B) + (2^{\lceil \log n \rceil} - n) \cdot B' \\
&= 2^{\lceil \log n \rceil} \cdot B' \\
&= M \cdot B'
\end{aligned}$$

Außerdem gilt $k > M$.

Zu zeigen ist:
$\exists$ Partition $S_1, \ldots, S_n$ von $\{1, \ldots, 3n\}$, so daß $\sum_{j \in S_i} a_j = B$ $\forall 1 \le i \le n$

$$\Longleftrightarrow$$

$\exists$ Partition $T_1, \ldots, T_M$ von $\{1, \ldots, k\}$, so daß $\sum_{j \in T_i} c_j = B'$ $\forall 1 \le i \le M$

„$\Longrightarrow$“:

Vorauss.: $\exists$ Partition $S_1, \ldots, S_n$ von $\{1, \ldots, 3n\}$, so daß $\sum_{j \in S_i} a_j = B$ $\forall 1 \le i \le n$.
Konstruiere $T_1, \ldots, T_M$ wie folgt:

$$\text{Für } 1 \le i \le n : T_i = S_i \cup \{3n + i\}$$

$$\text{Für } n < i \leq 2^{\lceil \log n \rceil} : T_i = \{3n + i\}$$

Da $S_1, \ldots, S_n$ eine Partition von $\{1, \ldots, 3n\}$, folgt, daß $T_1, \ldots, T_M$ eine Partition von $\{1, \ldots, k\}$ ist.
Noch z. z.: $\sum_{j \in T_i} c_j = B'$ für $1 \leq i \leq M$.

<u>1. Fall</u>: $1 \leq i \leq n$:

$$\sum_{j \in T_i} c_j = \sum_{j \in S_i} c_j + c_{3n+i} = \sum_{j \in S_i} a_j + (B' - B) = B + B' - B = B'$$

<u>2. Fall</u>: $n < i \leq 2^{\lceil \log n \rceil}$:

$$\sum_{j \in T_i} c_j = c_{3n+i} = B'$$

„$\Longleftarrow$“:

Vorauss.: $\exists$ Partition $T_1, \ldots, T_M$ von $\{1, \ldots, k\}$, so daß $\sum_{j \in T_i} c_j = B' \; \forall 1 \leq i \leq M$
Dann gilt:

- Es gibt $2^{\lceil \log n \rceil} - n$ Mengen T_i mit $T_i = \{3n + j\}$, wobei $j \in \{n+1, \ldots, 2^{\lceil \log n \rceil}\}$, da $c_{3n+j} = B'$ und $c_i > 0 \; \forall 1 \leq i \leq k$.
  Seien o.B.d.A. $T_i = \{3n + i\}$ für $n + 1 \leq i \leq 2^{\lceil \log n \rceil}$.
- Für die restlichen T_i mit $1 \leq i \leq n$ gilt:
  Es gibt kein T_i mit $\{i_1, i_2\} \subseteq T_i$ und $3n < i_1, i_2 \leq 4n$, denn sonst gilt:

$$\begin{aligned}
c_{i_1} + c_{i_2} &= (B' - B) + (B' - B) \\
&= B' + (B' - 2B) \\
&= B' + (2^{\lceil \log B \rceil + 2} - 2B) \\
&= B' + (4 \cdot 2^{\lceil \log B \rceil} - 2B) \\
&\geq B' + (4B - 2B) \\
&= B' + 2B \\
&> B', \quad \text{da } B > 0.
\end{aligned}$$

  $\Longrightarrow \; \forall 1 \leq i \leq n$ gilt: T_i enthält höchstens ein i_1 mit $3n < i_1 \leq 4n$.
  Da aber alle i_1 mit $3n < i_1 \leq 4n$ in einer der Mengen T_i $(1 \leq i \leq n)$ enthalten sein müssen, gilt $\forall 1 \leq i \leq n$:
  T_i enthält *genau* ein i_1 mit $3n < i_1 \leq 4n$.
  Gelte o.B.d.A.: $3n + i \in T_i \quad \forall 1 \leq i \leq n$

Wähle nun:

$$S_i = T_i \setminus \{3n + i\} \quad \forall 1 \leq i \leq n$$

$$\Longrightarrow \qquad S_i \subseteq \{1, \ldots, 3n\}$$

$S_1, \ldots S_n$ bilden eine Partition von $\{1, \ldots, 3n\}$.
Für $1 \leq i \leq n$ gilt:

$$\sum_{j \in S_i} a_j = \sum_{j \in S_i} c_j = \sum_{j \in T_i} c_j - c_{3n+i} = B' - (B' - B) = B.$$

□

Nachdem nun gezeigt ist, daß 2^l–PARTITION NP–hart ist (auch bei unärer Kodierung der vorkommenden Zahlen), kann Satz 3.10 bewiesen werden:

Beweis: Beweis durch Polynomtransformation von 2^l–Partition nach CDF:
Sei eine Instanz von 2^l–Partition gegeben durch $c_1, \ldots, c_k \in \mathbf{Z}^+$, $B' = 2^b$, $M = 2^l$ mit $\sum_{i=1}^k c_i = M \cdot B' = 2^{b+l}$, $k > M$.
Alle Zahlen sind unär kodiert!
Daraus wird nun in Polynomzeit eine Instanz des Problems CDF berechnet:
Es werden Funktionen f_1 und $f_2 \in B_n$ berechnet mit den Eingangsvariablen $x_1, \ldots, x_p$, $y_1, \ldots, y_p$, $p = l + b + 1$. Die Funktionen werden durch ihre Zerlegungsmatrizen Z_1 und Z_2 hinsichtlich der Variablenteilmenge $X^{(1)} = \{x_1, \ldots, x_p\}$ definiert. Die Zerlegungsmatrizen sind quadratisch mit 2^{l+b+1} Zeilen und 2^{l+b+1} Spalten.

Zur Konstruktion der Zerlegungsmatrizen:
Die Matrizen werden aus k Blöcken konstruiert, entsprechend den k Gewichten $c_1, \ldots, c_k$. (Die Blöcke werden gerade so konstruiert, daß gemeinsame Zerlegungsfunktionen von f_1 und f_2 den Zeilen der einzelnen Blöcke den gleichen Funktionswert zuordnen müssen.)
Bei der Definition von Z_1 und Z_2 wird folgende Kurzschreibweise benutzt: $un(i)$ ist definiert als

$$\underbrace{1 \ldots 1}_{i \text{ Mal}} \quad \underbrace{0 \ldots 0}_{2^{l+b+1}-i \text{ Mal}} \quad .$$

Also steht $un(i)$ für eine Zerlegungsmatrixzeile, die mit i Einsen beginnt, und dann nur noch Nullen enthält.
Aufbau von Block Nr. i:

<u>1. Fall</u>: $c_i = 1$
Block i besteht sowohl für Z_1 als auch für Z_2 aus zwei gleichen Zeilen:

Für Z_1: $\boxed{\begin{array}{c} un(\sum_{j=1}^{i-1} c_j) \\ un(\sum_{j=1}^{i-1} c_j) \end{array}}$ Für Z_2: $\boxed{\begin{array}{c} un(\sum_{j=1}^{i-1} c_j) \\ un(\sum_{j=1}^{i-1} c_j) \end{array}}$

<u>2. Fall</u>: $c_i > 1$
Block i besteht für Z_1 und Z_2 aus $2c_i$ Zeilen:

Für Z_1: $\boxed{\begin{array}{c} un(\sum_{j=1}^{i-1} c_j) \\ un(\sum_{j=1}^{i-1} c_j) \\ un(\sum_{j=1}^{i-1} c_j+1) \\ un(\sum_{j=1}^{i-1} c_j+1) \\ \vdots \\ un(\sum_{j=1}^{i-1} c_j+(c_i-2)) \\ un(\sum_{j=1}^{i-1} c_j+(c_i-1)) \\ un(\sum_{j=1}^{i-1} c_j+(c_i-1)) \end{array}}$ Für Z_2: $\boxed{\begin{array}{c} un(\sum_{j=1}^{i-1} c_j) \\ un(\sum_{j=1}^{i-1} c_j+1) \\ un(\sum_{j=1}^{i-1} c_j+1) \\ un(\sum_{j=1}^{i-1} c_j+2) \\ \vdots \\ un(\sum_{j=1}^{i-1} c_j+(c_i-1)) \\ un(\sum_{j=1}^{i-1} c_j+(c_i-1)) \\ un(\sum_{j=1}^{i-1} c_j) \end{array}}$

Die beiden Zerlegungsmatrizen haben $2\sum_{i=1}^{k} c_i = 2MB' = 2^{l+b+1}$ Zeilen und $2MB'$ Spalten und sind in polynomieller Zeit berechenbar.
Da Zeilen aus verschiedenen Blöcken verschieden sind und bei den Zeilen *eines* Blockes immer je 2 Zeilen paarweise gleich sind, beträgt die Anzahl *verschiedener* Zeilen $\frac{1}{2}(2^{l+b+1}) = 2^{l+b}$.
$\Longrightarrow$ Die minimale Anzahl der Zerlegungsfunktionen bei Zerlegung hinsichtlich $X^{(1)}$ beträgt sowohl bei f_1 als auch bei f_2 $\lceil\log(vz(X^{(1)}, f_1))\rceil = \lceil\log(vz(X^{(1)}, f_2))\rceil = l+b$.
Schließlich wird die natürliche Zahl h der Instanz von CDF gewählt als $h = l$, d.h. das Problem besteht darin, ob es Funktionen $\alpha_1, \ldots, \alpha_l \in B_p$ gibt, die alle als gemeinsame Zerlegungsfunktionen in einseitigen kommunikationsminimalen Zerlegungen von f_1 und f_2 hinsichtlich $X^{(1)}$ verwendet werden können.
Zu zeigen ist nun: $\exists$ Partition $T_1, \ldots, T_M$ von $\{1, \ldots, k\}$, so daß $\sum_{j\in T_i} c_j = B'$ $\forall 1 \le i \le M$

$$\Longleftrightarrow$$

$\exists h = l$ gemeinsame Zerlegungsfunktionen von f_1, f_2.

Bevor diese Äquivalenz bewiesen wird, wird noch eine Schreibweise definiert:
Die Zerlegungsmatrizen wurden durch k Blöcke definiert. $\{0,1\}^p$ zerfällt in k disjunkte Mengen $block_1, \ldots, block_k$, wenn man definiert:

$$(x_1, \ldots, x_p) \in block_i$$

$$\Longleftrightarrow$$

Die Zeile mit Nummer $int(x_1, \ldots, x_p)$ fällt in Block i.

„$\Longrightarrow$“:

Vorauss.: $\exists$ Partition $T_1, \ldots, T_M$ von $\{1, \ldots, k\}$, so daß $\sum_{j\in T_i} c_j = B'$ $\forall 1 \le i \le M$.

Es werden nun l gemeinsame Zerlegungsfunktionen $\alpha_1, \ldots, \alpha_l$ von f_1 und f_2 definiert:

Für alle $1 \leq i \leq 2^l$ und alle $j \in T_i$ setze für alle $\mathbf{x} = (x_1, \ldots, x_p) \in block_j$

$$\alpha(\mathbf{x}) = (\alpha_1(\mathbf{x}), \ldots, \alpha_l(\mathbf{x})) = bin_l(i-1)$$

(D.h. falls $T_i = \{i_1, \ldots, i_q\}$, dann erhalten alle Blöcke $block_{i_1}, \ldots block_{i_q}$ den Funktionswert $bin_l(i-1)$.)

Die Anzahl verschiedener Zeilen $anz_{bin_l(i-1)}$ der Zerlegungsmatrizen, denen durch α der Funktionswert $bin_l(i-1)$ zugeordnet wird, beträgt (sowohl für Z_1 als auch für Z_2):

$$\begin{aligned} anz_{bin_l(i-1)} &= \sum_{j \in T_i} (\text{Anzahl verschiedener Zeilen in Block } j) \\ &= \sum_{j \in T_i} c_j \\ &= B' = 2^b \end{aligned}$$

$$\Longrightarrow vz(X^{(1)}, f_1, \alpha) = vz(X^{(1)}, f_2, \alpha) = 2^b.$$

Nach Lemma 3.10 existieren daher einseitige Zerlegungen von f_1 und f_2 hinsichtlich $X^{(1)}$ der Form

$$\begin{aligned} &f_1(x_1, \ldots, x_p, y_1, \ldots, y_p) = \\ &\quad g^{(1)}(\alpha_1(\mathbf{x}), \ldots, \alpha_l(\mathbf{x}), \alpha^{(1)}_{l+1}(\mathbf{x}), \ldots, \alpha^{(1)}_{l+b}(\mathbf{x}), y_1, \ldots, y_p) \\ &f_2(x_1, \ldots, x_p, y_1, \ldots, y_p) = \\ &\quad g^{(2)}(\alpha_1(\mathbf{x}), \ldots, \alpha_l(\mathbf{x}), \alpha^{(2)}_{l+1}(\mathbf{x}), \ldots, \alpha^{(2)}_{l+b}(\mathbf{x}), y_1, \ldots, y_p) \end{aligned}$$

„$\Longleftarrow$“:

Vorauss.: $\exists\ l$ gemeinsame Zerlegungsfunktionen $\alpha_1, \ldots, \alpha_l$ von f_1 und f_2.

Da Z_1 und Z_2 jeweils 2^{l+b} verschiedene Zeilen haben und die Gesamtzahl der Zerlegungsfunktionen bei der Zerlegung von f_1 bzw. f_2 jeweils $l + b$ beträgt, ist es nicht möglich, daß Zerlegungsfunktionen den Indizes *gleicher* Zeilen von Z_1 bzw. von Z_2 *verschiedene* Zerlegungsfunktionswerte zuordnen.
$\Longrightarrow$ Auch $\alpha_1, \ldots, \alpha_l$ ordnen den Indizes gleicher Zeilen gleiche Werte zu.

<u>Beh.1</u>: $\alpha(x_1, \ldots, x_p) = \alpha(x'_1, \ldots, x'_p)$ für alle $(x_1, \ldots, x_p)$, $(x'_1, \ldots, x'_p) \in block_i$ $(1 \leq i \leq k)$.
Beweis:
$\alpha_1, \ldots, \alpha_l$ sind gemeinsame Zerlegungsfunktionen von f_1 und f_2.
Zeilen 1 und 2 von Block i in Z_1 sind gleich $\Rightarrow$ α liefert auf den Indizes von

Zeilen 1 und 2 den gleichen Funktionswert.
Zeilen 2 und 3 von Block i in Z_2 sind gleich $\Rightarrow$ α liefert auf den Indizes von Zeilen 2 und 3 den gleichen Funktionswert.
Betrachtet man weiter Zeile 3 und 4 von Block i in Z_1 usw., so erhält man die Behauptung.

Beh.2: Sowohl bei f_1 als auch bei f_2 ordnet α den Indizes von *genau* $2^b = B'$ *verschiedenen* Zeilen gleiche Funktionswerte zu.
Beweis:

- Nach Lemma 3.10 ist $vz(X^{(1)}, f_1, \alpha) \leq 2^b$ und $vz(X^{(1)}, f_2, \alpha) \leq 2^b$. Für jeden Funktionswert von α ist also die Zahl der verschiedenen Zeilen, deren Indizes dieser Funktionswert zugeordnet wird, kleiner oder gleich 2^b.
- Es gibt $2^l \cdot 2^b$ verschiedene Zeilen der Zerlegungsmatrizen Z_1 bzw. Z_2 und genau 2^l verschiedene Funktionswerte von α sind möglich. Die Anzahl der verschiedenen Zeilen, deren Indizes der gleiche Funktionswert zugeordnet wird, beträgt also *genau* 2^b.

Aus den Behauptungen 1 und 2 ergibt sich, daß sich die Blöcke der Zerlegungsmatrizen in Gruppen einteilen lassen, wobei die Anzahl der verschiedenen Zeilen in einer solchen Gruppe gerade 2^b beträgt.

Definiere nun für $i = 1, \ldots, 2^l$:

$$T_i = \{j \mid \alpha(\mathbf{x}) = bin_l(i-1) \, \forall \, \mathbf{x} \in block_j\}.$$

$T_1, \ldots, T_M$ bilden eine Partition von $\{1, \ldots, k\}$ wegen Behauptung 1.
Die Anzahl verschiedener Zeilen, deren Indizes durch α der Funktionswert $bin_l(i-1)$ zugeordnet wird, ergibt sich als

$$2^b = B' = \sum_{j \in T_i} (\text{Anzahl verschiedener Zeilen in Block } j) = \sum_{j \in T_i} c_j.$$

Also gilt für die angegebene Partition $T_1, \ldots, T_M$:

$$\sum_{j \in T_i} c_j = B'$$

und $T_1, \ldots, T_M$ stellt eine Lösung der gegebenen Instanz des 2^l–PARTITION–Problems dar.

$\square$

Anhang B

Aufzählung von Cliquen

Bei der Bestimmung von gemeinsamen Zerlegungsfunktionen für Teilmengen der Funktionen $f_1, \ldots, f_m$ in Abschnitt 3.4.2.2 wird ein Graph G_{CDF} betrachtet, dessen Knoten die Funktionen $f_1, \ldots, f_m$ sind, wobei genau dann eine Kante zwischen zwei Funktionen f_{i_1} und f_{i_2} existiert, wenn es eine strikte Zerlegung von f_{i_1} und f_{i_2} gibt, in der eine Zerlegungsfunktion gemeinsam verwendet werden kann. Bei der Suche nach gemeinsamen Zerlegungsfunktionen kann man sich folglich auf solche Teilmengen von $f_1, \ldots, f_m$ beschränken, die in G_{CDF} eine Clique bilden. Zur Aufzählung der Cliquen von G_{CDF} mit abnehmender Größe wird ein Algorithmus verwendet, der aus einem Algorithmus von Carraghan/Pardalos [CP90] zur exakten Bestimmung der größten Clique in einem Graphen hergeleitet ist.
Es ist zu beachten, daß man im Fall, daß für eine Ausgangsfunktion f_j schon alle Zerlegungsfunktionen bestimmt worden sind, keine Teilmengen von $\{f_1, \ldots, f_m\}$ mehr zu betrachten braucht, die f_j enthalten, so daß für solche f_j während des Aufzählverfahrens Knoten aus G_{CDF} gelöscht werden müssen.

Es folgt der Algorithmus zur Aufzählung der Cliquen mit einer kurzen Erläuterung zur Funktionsweise des Algorithmus:

```
procedure init_clique
begin
  ∀ (1 ≤ j ≤ m) : enum[1][j] = j
  knotenanz[1] = m
  start[1] = 0
  aktuelle_zeile = 1
end

function nächste_clique(int k)
begin
  while (aktuelle_zeile ≥ 1) do
        start[aktuelle_zeile] = start[aktuelle_zeile] + 1
        if ((aktuelle_zeile − 1) + (knotenanz[aktuelle_zeile] − start[aktuelle_zeile] + 1) < k)
          /* Dann gibt es keine Fortsetzung der begonnenen Clique mit Größe k */
          then
                /* Ändere in vorangegangener Zeile */
                aktuelle_zeile = aktuelle_zeile − 1
```

```
            else
                if (aktuelle_zeile < k)
                  /* Aktuelle Clique noch kleiner als k */
                  then
                       /* Fortsetzung der aktuellen Clique: */
                       aktuelle_zeile = aktuelle_zeile + 1
                       knotenanz[aktuelle_zeile] = 0
                       start[aktuelle_zeile] = 0
                       for j = start[aktuelle_zeile − 1] + 1 to knotenanz[aktuelle_zeile − 1] do
                           if ({enum[aktuelle_zeile − 1][start[aktuelle_zeile − 1]], enum[aktuelle_zeile − 1][j]} ∈ E)
                             then
                                  knotenanz[aktuelle_zeile] = knotenanz[aktuelle_zeile] + 1
                                  enum[aktuelle_zeile][knotenanz[aktuelle_zeile]] = enum[aktuelle_zeile − 1][j]
                           fi
                       od
                   else
                       /* Neue Clique der Größe k gefunden */
                       return 1
                fi
          fi
  od
  /* Keine neue Clique der Größe k mehr vorhanden */
  return 0
end

procedure lösche_knoten(int lösch_knoten)
begin
  for (i = 1 to aktuelle_zeile) do
      if (enum[i][start[i]] = lösch_knoten)
        then
             /* Keine neue Clique mehr aufzählen, die Knoten lösch_knoten enthält */
             aktuelle_zeile = i
             return
      fi
      /* Knoten aus Zeile i löschen: */
      j = start[i] + 1
      while (enum[i][j] ≠ lösch_knoten) do
            j++
      od
      while (j < knotenanz[i]) do
            enum[i][j] = enum[i][j + 1]
            j++
      od
      knotenanz[i] = knotenanz[i] − 1
  od
end

begin
  for k = m downto 1 do
      init_clique()
      while (nächste_clique(k) = 1) do
            tm = {f_enum[i][start[i]] | 1 ≤ i ≤ k}
            /* Berechne (unter Voraussetzung der bisher schon berechneten Zerlegungsfunktionen) mit Hilfe des
               branch–and–bound–Verfahrens eine maximale Anzahl gemeinsamer Zerlegungsfunktionen der
               Funktionen in tm. */
            foreach f_j ∈ tm do
              if (Alle Zerlegungsfunktionen von f_j bestimmt)
                then
                     lösche_knoten(j)
              fi
            od
      od
  od
end
```

Die Cliquen in G_{CDF} werden mit abnehmender Größe aufgezählt. G_{CDF} habe m Knoten. Der Einfachheit halber seien die Knoten aus $\{1, \dots m\}$, die Kantenmenge von G_{CDF} sei mit E bezeichnet. Die Cliquen der Größe k werden dann in lexikographischer Reihenfolge aufgezählt.
Für eine bestimmte Cliquengröße k wird der aktuelle Zustand bei der Aufzählung gespeichert in der $m \times m$–Matrix *enum*, in dem Vektor *start* der Länge m, dem Vektor *knotenanz* der Länge m und der Variablen *aktuelle_zeile* (mit Wert aus $\{1, \dots, m\}$). In der Matrix *enum* werden Knotennummern eingetragen. Die Variable *aktuelle_zeile* gibt an, wieviele Zeilen in *enum* schon belegt sind. In der Prozedur *init_clique* wird die erste Zeile von *enum* mit den Knotennummern 1 bis m belegt. In $knotenanz[i]$ wird gespeichert, wieviele Knoten in Zeile i eingetragen worden sind.
Die Knoten $enum[1][start[1]]$, $\dots$, $enum[aktuelle\_zeile][start[aktuelle\_zeile]]$ stellen den Anfang der aktuell aufgezählten Clique dar. Ist *aktuelle_zeile* noch kleiner als k, so wird versucht, die Clique fortzusetzen. Es wird eine neue Zeile der Matrix *enum* belegt (Zeilen 24–34 des Algorithmus). Diese neue Zeile enthält alle Knoten aus $enum[aktuelle\_zeile - 1][start[aktuelle\_zeile - 1] + 1], \dots, enum[aktuelle\_zeile - 1][knotenanz[aktuelle\_zeile - 1]]$, für die es eine Kante zu $enum[aktuelle\_zeile - 1][start[aktuelle\_zeile - 1]]$ gibt. Aufgrund dieses Konstruktionsprinzips der Matrix *enum* ergibt sich sofort, daß die Knoten $enum[1][start[1]], \dots, enum[aktuelle\_zeile - 1][start[aktuelle\_zeile - 1]]$,
$enum[aktuelle\_zeile][j]$ $(1 \leq j \leq knotenanz[aktuelle\_zeile])$ eine Clique in G_{CDF} bilden.
Die aktuelle Clique wird solange fortgesetzt, bis sie die Größe k erreicht hat (Zeile 37) oder bis klar ist, daß sie die Größe k nicht mehr erreichen kann, da die Anzahl der in ihr schon vorhandenen Knoten[1] plus der Anzahl der Knoten, um die die Clique maximal noch erweitert werden kann[2] kleiner als k ist (Zeile 15). In diesem Fall wird wieder eine Zeile der Matrix *enum* gelöscht (Zeile 19) und in einem weiteren Durchlauf der **while**–Schleife aus Zeilen 13–40 $start[aktuelle\_zeile]$ um 1 erhöht und evtl. die nächste Zeile von *enum* aufgrund der Änderung von $start[aktuelle\_zeile]$ konstruiert.

Die Prozeduraufrufe $nächste\_clique(k)$ (Zeile 74) liefern also, solange die Rückgabewerte gleich 1 sind, in lexikographischer Reihenfolge sämtliche Cliquen der Größe k in G_{CDF}. Für eine solche Clique wird dann mit Hilfe des branch–and–bound–Verfahrens aus Kapitel 3.4.2.1 eine maximale Anzahl an gemeinsamen Zerlegungsfunktionen bestimmt. Sind im Verlauf des Verfahrens für eine Funktion f_j schon sämtliche Zerlegungsfunktionen bestimmt, so wird der zugehörige Knoten j aus G_{CDF} entfernt und muß daher durch die Prozedur *lösche_knoten* (Zeilen 47–67) aus der Datenstruktur zur Aufzählung der Cliquen entfernt werden.

[1]nämlich $enum[1][start[1]], \dots, enum[aktuelle\_zeile][start[aktuelle\_zeile]]$
[2]nämlich $enum[aktuelle\_zeile][start[aktuelle\_zeile] + 1], \dots,$
$\dots, enum[aktuelle\_zeile][knotenanz[aktuelle\_zeile]]$

Anhang C

Testen rekursiv zerlegter Schaltungen

C.1 Berechnung von Freiheitsrelationen

C.1.1 Sonderfälle bei der Berechnung von FR_α

Die Berechnung der Freiheitsrelation FR_α aus Abschnitt 3.7.1.2 erfolgt für die Sonderfälle $m = 1$ (nur eine Ausgangsfunktion) und $FR_f = R(f)$ (FR_f ist eine totale Funktion) durch auf die Sonderfälle zugeschnittene Spezialverfahren:

1. $m = 1$:
   Dann ist FR_f darstellbar als partielle Funktion fp. Aus den ROBDDs zu fp und der totalen Zusammensetzungsfunktion g kann man dann FR_α bestimmen[1]:
   Der ROBDD zu fp wird nach den Variablen aus der Menge $X^{(1)} = \{x_1, \ldots, x_p\}$ geschnitten. Die Verbindungsknoten unterhalb der Schnittlinie repräsentieren (partielle) Kofaktoren $fpcof_1, \ldots, fpcof_{vz(X^{(1)},fp)}$. Zu diesen Kofaktoren gehören Äquivalenzklassen $FPK_1, \ldots, FPK_{vz(X^{(1)},fp)}$, die eine Partition $\{0,1\}^p/_\equiv$ bilden, wobei für zwei Elemente $\epsilon, \delta \in \{0,1\}^p$ $\epsilon \equiv \delta$ genau dann, wenn die (partiellen) Kofaktoren $fp_{x_1^{\epsilon_1},\ldots,x_p^{\epsilon_p}}$ und $fp_{x_1^{\delta_1},\ldots,x_p^{\delta_p}}$ gleich sind (vergleiche Kapitel 3.2). ROBDDs zu den charakteristischen Funktionen $\chi_{FPK_1}, \ldots, \chi_{FPK_{vz(X^{(1)},fp)}}$ lassen sich leicht aus dem ROBDD zu fp bestimmen.
   Analog wird der ROBDD zu g nach den Variablen aus der Menge $A = \{a_1, \ldots, a_r\}$ geschnitten. Man erhält Verbindungsknoten, die die (totalen) Kofaktoren $gcof_1, \ldots, gcof_{vz(A,g)}$ repräsentieren, und die zugehörigen Äquivalenzklassen $code_1, \ldots, code_{vz(A,g)}$ aus $\{0,1\}^r/_\equiv$.
   Ist nun $gcof_i$ eine totale Erweiterung eines (partiellen) Kofaktors $fpcof_j$, so ist für alle $(\tilde{x_1}, \ldots, \tilde{x_p}) \in FPK_j$ und alle Codes $(\tilde{a_1}, \ldots, \tilde{a_r}) \in code_i$:

$$(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{a_1}, \ldots, \tilde{a_r}) \in FR_\alpha,$$

[1] Nehme an, daß in der Variablenordnung des ROBDD zu fp $x_1, \ldots, x_p$ vor $y_1, \ldots, y_q$ stehen und beim ROBDD zu g $a_1, \ldots, a_r$ vor $y_1, \ldots, y_q$ stehen.

da dann für alle $(\tilde{y_1}, \ldots, \tilde{y_q}) \in D(fpcof_j)$ gilt:

$$\begin{array}{rcl} f(\tilde{x_1}, \ldots, \tilde{x_p}, \tilde{y_1}, \ldots, \tilde{y_q}) & = & fpcof_j(\tilde{y_1}, \ldots, \tilde{y_q}) \\ & = & gcof_i(\tilde{y_1}, \ldots, \tilde{y_q}) \quad = \quad g(\tilde{a_1}, \ldots, \tilde{a_r}, \tilde{y_1}, \ldots, \tilde{y_q}). \end{array}$$

χ_{FR_α} ergibt sich dann als

$$\chi_{FR_\alpha} = \bigvee_{i=1}^{vz(X^{(1)},fp)} (\chi_{FPK_i} \wedge \bigvee \{\chi_{code_j} \mid 1 \leq j \leq vz(A,g), gcof_j \text{ Erweiterung von } fpcof_i\}).$$

2. $FR_f = R(f)$:
   Wenn die Freiheitsrelation FR_f eine totale Boolesche Funktion ist (wie z.B. zumeist bei der ersten Rekursionsstufe des Logiksyntheseverfahrens), dann kann zu einem $\tilde{\mathbf{x}}$ nur dann für verschiedene $\tilde{\mathbf{a}}$ und $\tilde{\mathbf{b}}$ sowohl $(\tilde{\mathbf{x}}, \tilde{\mathbf{a}})$ als auch $(\tilde{\mathbf{x}}, \tilde{\mathbf{b}})$ in FR_α enthalten sein, wenn für alle $\tilde{\mathbf{y}} \in \{0,1\}^q$ $g(\tilde{\mathbf{a}}, \tilde{\mathbf{y}}) = g(\tilde{\mathbf{b}}, \tilde{\mathbf{y}})$. (Setzt man voraus, daß die Zerlegung aller f_i strikt ist, so kann dies nur der Fall sein, wenn $\tilde{\mathbf{a}}$ oder $\tilde{\mathbf{b}}$ nicht im Bildbereich von α liegt.)
   Um FR_α zu berechnen, muß man also die Mengen $GK_1, \ldots, GK_{vz(A,g)}$ kennen, so daß für $\tilde{\mathbf{a}}$ und $\tilde{\mathbf{b}} \in GK_i$ $(1 \leq i \leq vz(A,g))$ $g(\tilde{\mathbf{a}}, \tilde{\mathbf{y}}) = g(\tilde{\mathbf{b}}, \tilde{\mathbf{y}})$ für alle $\tilde{\mathbf{y}} \in \{0,1\}^q$. Charakteristische Funktionen zu diesen Mengen lassen sich wie in Abschnitt 3.4.3 bestimmen[2].
   Stelle nun den ROBDD zu $R(\alpha)$ auf (wobei $x_1, \ldots, x_p$ in der Variablenordnung vor $a_1, \ldots, a_r$ liegen). Schneide diesen ROBDD nun nach der Variablenmenge $\{x_1, \ldots, x_p\}$. Ein Verbindungsknoten v repräsentiert dann eine charakteristische Funktion zu einer Menge $\{\tilde{\mathbf{a}}\}$, wobei $\tilde{\mathbf{a}}$ im Bild von α vorkommt. Der Teilbdd, dessen Wurzel v ist, muß dann ersetzt werden durch den ROBDD zu χ_{GK_i}, wobei GK_i $\tilde{\mathbf{a}}$ enthält. Führt man dies für alle Verbindungsknoten durch, so erhält man einen OBDD zu χ_{FR_α}.

C.1.2 Sonderfälle bei der Berechnung von FR_g

Auch die Berechnung der Freiheitsrelation FR_g aus Abschnitt 3.7.1.2 erfolgt für die Sonderfälle $m = 1$ (nur eine Ausgangsfunktion) und $FR_f = R(f)$ (FR_f ist eine totale Funktion) durch auf die Sonderfälle zugeschnittene Spezialverfahren:

1. $m = 1$:
   FR_f und FR_g sind dann darstellbar als partielle Funktionen. Aus den ROBDDs zu fp (partielle Funktion zu FR_f) und der totalen Zerlegungsfunktion α kann man dann (analog zum Vorgehen in Abschnitt 3.2.5) FR_g bestimmen[3]:

[2]Analog zur Bestimmung von Äquivalenzklassen zu ROBDD *kl* auf Seite 180.
[3]Nehme an, daß in der Variablenordnung des ROBDD zu fp $x_1, \ldots, x_p$ vor $y_1, \ldots, y_q$ stehen.

Der ROBDD zu fp wird nach den Variablen aus der Menge $X^{(1)} = \{x_1, \ldots, x_p\}$ geschnitten. Die Verbindungsknoten unterhalb der Schnittlinie repräsentieren (partielle) Kofaktoren $fpcof_1, \ldots, fpcof_{vz(X^{(1)},fp)}$. Zu diesen Kofaktoren gehören die Äquivalenzklassen $FPK_1, \ldots, FPK_{vz(X^{(1)},fp)}$ aus $\{0,1\}^p/_\equiv$. Die totale Zerlegungsfunktion α ist hervorgegangen aus einer Kodierung von Klassen $K_1, \ldots, K_l$ ($K_i \subseteq \{0,1\}^p$) durch Codes $(a_1^{(1)}, \ldots, a_r^{(1)}), \ldots, (a_1^{(l)}, \ldots, a_r^{(l)})$. Der ROBDD für die partielle Funktion $FR(g)$ ergibt sich wie in Abschnitt 3.2.5, indem man einen ROBDD konstruiert, bei dem man durch $(a_1^{(i)}, \ldots, a_r^{(i)})$ den (partiellen) Kofaktor $gem\_erw(\{fpcof_j \mid K_i \cap FPK_j \neq \emptyset\})$ erreicht. Durch ein $(\epsilon_1, \ldots, \epsilon_r) \in \{0,1\}^r$, das nicht als Ausgabe von α auftritt, wird dagegen der Knoten erreicht, der die don't care–Stellen repräsentiert.

2. $FR_f = R(f)$:
   In diesem Fall ist $FR(g)$ eine partielle Funktion. Die don't care–Stellen von $FR(g)$ sind bestimmt durch die Elemente von $\{0,1\}^r$, die nicht als Ausgaben von α auftreten können. Sei $code(\alpha^{(i)}) \subseteq \{0,1\}^{r_i}$ die Menge aller Codevektoren, die bei der Zerlegung von f_i auftreten und $\chi_{code(\alpha^{(i)})}$ die zugehörige charakteristische Funktion. Dann ist die charakteristische Funktion für die Menge aller Ausgaben von α gegeben durch $\bigwedge_{i=1}^m \chi_{code(\alpha^{(i)})}$, so daß

$$\chi_{FR(g)} = \chi_{R(g)} \vee \overline{\bigwedge_{i=1}^{m} \chi_{code(\alpha^{(i)})}}.$$

C.2 Beispiel für Redundanzen

Es wird hier ein Beispiel angegeben, das zeigt, daß das Verfahren aus Abschnitt 3.7.1.2 zwar dazu benutzt werden kann, die Anzahl der redundanten Stuck–at–Fehler zu verringern, aber daß nicht garantiert werden kann, daß das Resultat völlig frei von redundanten Stuck–at–Fehlern ist.

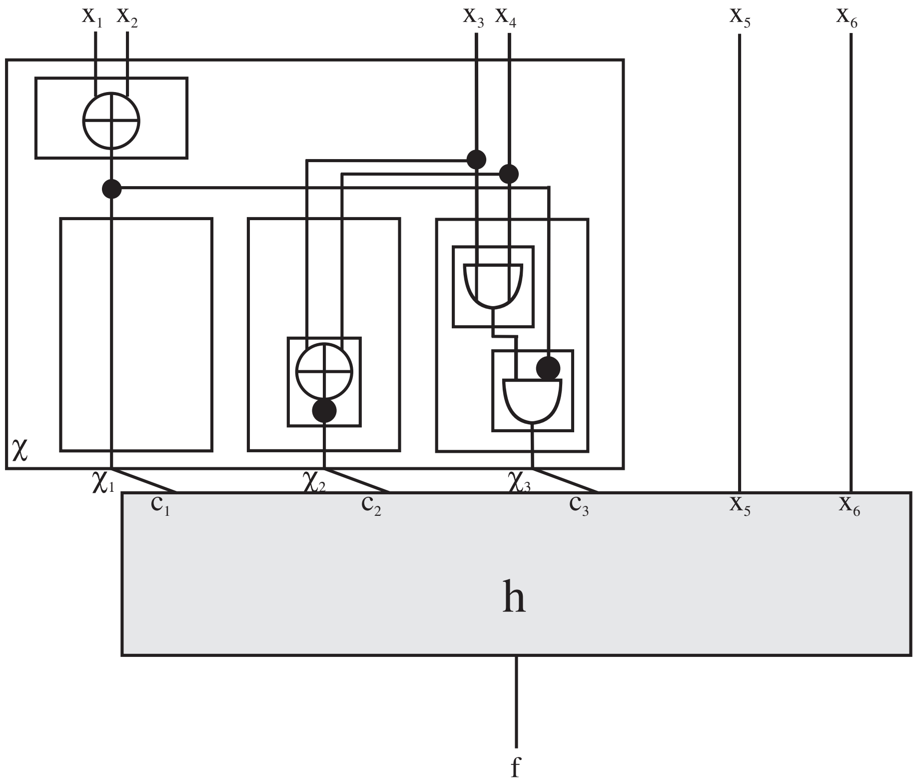
Der Grund dafür liegt darin, daß *gerade das Entfernen redundanter s-a–Fehler verhindert, daß man zum Zeitpunkt der Synthese eines Teilschaltkreises die Freiheitsrelation zu diesem Teilschaltkreis genau kennen kann.* Durch Entfernen redundanter s-a–Fehler wird die bei der Synthese eines Teilschaltkreises angenommene Umgebung des Teilschaltkreises im Gesamtschaltkreis evtl. verändert und damit wird evtl. auch seine Freiheitsrelation verändert.

Die zu realisierende Beispielfunktion ist durch die folgende Zerlegungsmatrix definiert:

f :

				x_5	0	0	1	1
				x_6	0	1	0	1
χ_1	χ_2	χ_3		$x_1x_2x_3x_4$				
0	1	0	$\Longleftarrow$	0 0 0 0	1	1	0	0
0	0	1	$\Longleftarrow$	0 0 0 1	0	1	1	1
0	0	1	$\Longleftarrow$	0 0 1 0	0	1	1	1
0	1	1	$\Longleftarrow$	0 0 1 1	1	0	0	0
1	1	0	$\Longleftarrow$	0 1 0 0	0	0	0	0
1	0	0	$\Longleftarrow$	0 1 0 1	0	0	1	1
1	0	0	$\Longleftarrow$	0 1 1 0	0	0	1	1
1	1	0	$\Longleftarrow$	0 1 1 1	0	0	0	0
1	1	0	$\Longleftarrow$	1 0 0 0	0	0	0	0
1	0	0	$\Longleftarrow$	1 0 0 1	0	0	1	1
1	0	0	$\Longleftarrow$	1 0 1 0	0	0	1	1
1	1	0	$\Longleftarrow$	1 0 1 1	0	0	0	0
0	1	0	$\Longleftarrow$	1 1 0 0	1	1	0	0
0	0	1	$\Longleftarrow$	1 1 0 1	0	1	1	1
0	0	1	$\Longleftarrow$	1 1 1 0	0	1	1	1
0	1	1	$\Longleftarrow$	1 1 1 1	1	0	0	0

Es erfolgt zunächst eine Zerlegung hinsichtlich $\{x_1, x_2, x_3, x_4\}$ wie durch die obige Zerlegungsmatrix angegeben. Es gibt 3 Zerlegungsfunktionen χ_1, χ_2, χ_3, die Zusammensetzungsfunktion sei h wie in der untenstehenden Abbildung angegeben.



Wesentlich ist lediglich, daß die Vektoren (000), (101) und (111) nicht im Bild von χ auftauchen, so daß h eine partielle Funktion ist.
Danach wird h hinsichtlich $\{c_1, x_5, x_6\}$ zerlegt:

h :

			c_2	0	0	1	1
			c_3	0	1	0	1
α_1	α_2		$c_1x_5x_6$				
0	1	$\Longleftarrow$	0 0 0	$\star$	0	1	1
1	1	$\Longleftarrow$	0 0 1	$\star$	1	1	0
0	0	$\Longleftarrow$	0 1 0	$\star$	1	0	0
0	0	$\Longleftarrow$	0 1 1	$\star$	1	0	0
0	0	$\Longleftarrow$	1 0 0	0	$\star$	0	$\star$
0	0	$\Longleftarrow$	1 0 1	0	$\star$	0	$\star$
1	0	$\Longleftarrow$	1 1 0	1	$\star$	0	$\star$
1	0	$\Longleftarrow$	1 1 1	1	$\star$	0	$\star$

Das obige Bild zeigt eine mögliche Kodierung der verschiedenen Zeilen der Zerlegungsmatrix. Die Zerlegungsfunktionen sind α_1 und α_2. Die Zusammensetzungsfunktion g kann man dann wählen wie durch die folgende Matrix angegeben (Eingangsvariablen a_1 und a_2

stehen für α_1 und α_2.)

$$
g: \quad
\begin{array}{c|cccc}
c_2 & 0 & 0 & 1 & 1 \\
c_3 & 0 & 1 & 0 & 1 \\
a_1 a_2 & & & & \\
\hline
0\ 0 & 0 & 1 & 0 & 0 \\
0\ 1 & 0 & 0 & 1 & 1 \\
1\ 0 & 1 & 1 & 0 & 0 \\
1\ 1 & 1 & 1 & 1 & 0
\end{array}
$$

Die Freiheitsrelation zu g ist durch die folgende Matrix einer partiellen Funktion gegeben:

$$
FR_g: \quad
\begin{array}{c|cccc}
c_2 & 0 & 0 & 1 & 1 \\
c_3 & 0 & 1 & 0 & 1 \\
a_1 a_2 & & & & \\
\hline
0\ 0 & 0 & 1 & 0 & 0 \\
0\ 1 & \star & 0 & 1 & 1 \\
1\ 0 & 1 & \star & 0 & \star \\
1\ 1 & \star & 1 & 1 & 0
\end{array}
$$

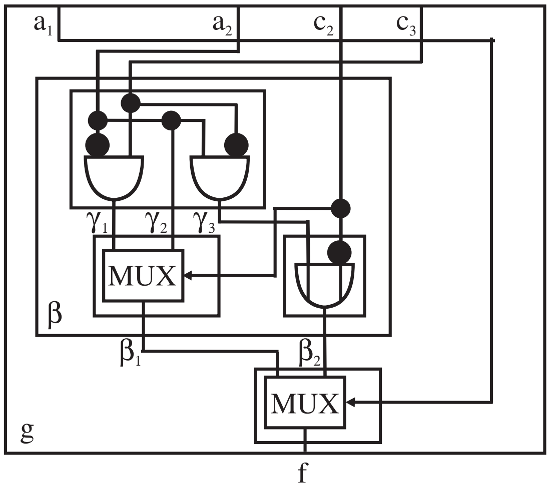
g wird hinsichtlich $\{c_2, c_3, a_2\}$ zerlegt. Die Zerlegungsfunktionen sind β_1 und β_2:

$$
\begin{array}{cc|cc|c}
(FR_\beta)_1 & (FR_\beta)_2 & \beta_1 & \beta_2 & \\
\hline
0 & 1 & 0 & 1 & \Longleftarrow \\
\star & \star & 0 & 1 & \Longleftarrow \\
1 & \star & 1 & 1 & \Longleftarrow \\
0 & 1 & 0 & 1 & \Longleftarrow \\
0 & 0 & 0 & 0 & \Longleftarrow \\
1 & 1 & 1 & 1 & \Longleftarrow \\
0 & \star & 0 & 0 & \Longleftarrow \\
1 & 0 & 1 & 0 & \Longleftarrow
\end{array}
\quad
g: \;
\begin{array}{c|cc}
a_1 & 0 & 1 \\
c_2 c_3 a_2 & & \\
\hline
0\ 0\ 0 & 0 & 1 \\
0\ 0\ 1 & 0 & 1 \\
0\ 1\ 0 & 1 & 1 \\
0\ 1\ 1 & 0 & 1 \\
1\ 0\ 0 & 0 & 0 \\
1\ 0\ 1 & 1 & 1 \\
1\ 1\ 0 & 0 & 0 \\
1\ 1\ 1 & 1 & 0
\end{array}
\quad
FR_g: \;
\begin{array}{c|cc}
a_1 & 0 & 1 \\
c_2 c_3 a_2 & & \\
\hline
0\ 0\ 0 & 0 & 1 \\
0\ 0\ 1 & \star & \star \\
0\ 1\ 0 & 1 & \star \\
0\ 1\ 1 & 0 & 1 \\
1\ 0\ 0 & 0 & 0 \\
1\ 0\ 1 & 1 & 1 \\
1\ 1\ 0 & 0 & \star \\
1\ 1\ 1 & 1 & 0
\end{array}
$$

Wichtig bei der nun folgenden Zerlegung von β_1 und β_2 ist, daß man bei der Zerlegung von β_1 2 Zerlegungsfunktionen benötigt, die von c_3 und a_2 abhängen, und nicht mit einer einzigen Zerlegungsfunktion auskommt (auch nicht durch Ausnutzung der Freiheitsrelation, die sich in diesem Fall wiederum durch partielle Funktionen für β_1 und β_2 darstellen läßt):

$$
\beta_1: \quad
\begin{array}{cc|c}
\gamma_1 & \gamma_2 & \\
\hline
0 & 0 & \Longleftarrow \\
0 & 1 & \Longleftarrow \\
1 & 0 & \Longleftarrow \\
0 & 1 & \Longleftarrow
\end{array}
\;
\begin{array}{c|cc}
c_2 & 0 & 1 \\
c_3 a_2 & & \\
\hline
0\ 0 & 0 & 0 \\
0\ 1 & 0(\star) & 1 \\
1\ 0 & 1 & 0 \\
1\ 1 & 0 & 1
\end{array}
\qquad
\beta_2: \quad
\begin{array}{c|c}
\gamma_3 & \\
\hline
0 & \Longleftarrow \\
1 & \Longleftarrow \\
0 & \Longleftarrow \\
0 & \Longleftarrow
\end{array}
\;
\begin{array}{c|cc}
c_2 & 0 & 1 \\
c_3 a_2 & & \\
\hline
0\ 0 & 1 & 0 \\
0\ 1 & 1(\star) & 1 \\
1\ 0 & 1(\star) & 1(\star) \\
1\ 1 & 1 & 0
\end{array}
$$

Insgesamt ergibt sich die folgende Realisierung von g:



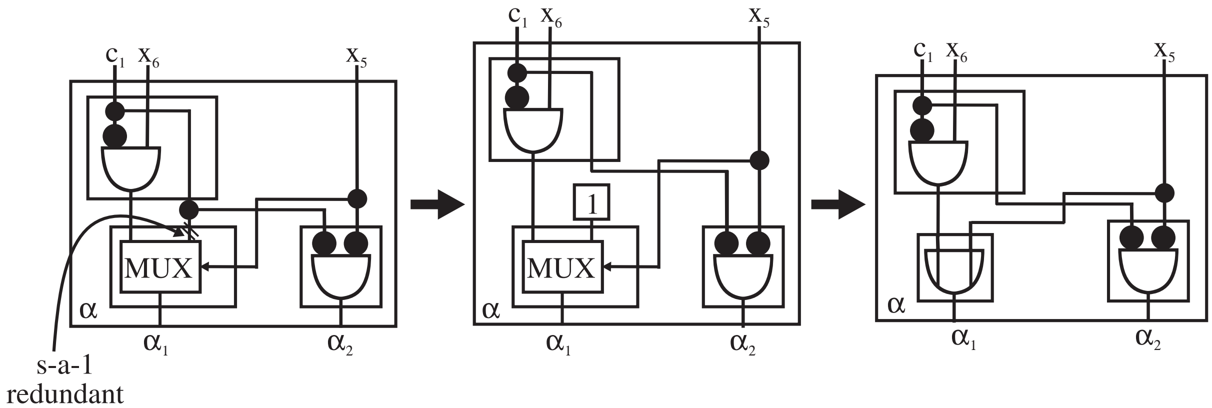
In dieser Realisierung tritt unter Voraussetzung der obigen Freiheitsrelation FR_g *kein redundanter* Fehler auf.

Allerdings wird nun bei der Realisierung der Zerlegungsfunktionen α_1 und α_2 zu h die Funktion α_1 zur Vermeidung eines redundanten Stuck–at–Fehlers zu einer Funktion α_1' verändert. Dadurch ändert sich FR_g (bzgl. der nun veränderten Umgebung) und in der obigen Realisierung von g wird ein Fehler redundant:

Die Freiheitsrelation von α ist durch die untenstehende Tabelle gegeben:

c_1	x_5	x_6	(α_1, α_2)
0	0	0	$(0,1)$
0	0	1	$(1,1)$
0	1	0	$(0,0),(1,0)$
0	1	1	$(0,0),(1,0)$
1	0	0	$(0,0)$
1	0	1	$(0,0)$
1	1	0	$(1,0)$
1	1	1	$(1,0)$

Bei der Zerlegung von α hinsichtlich $\{c_1, x_6\}$ würde sich ein redundanter s-a-1–Fehler ergeben:



Das Einpflanzen dieses s-a-1–Fehlers verändert die Ausgabe von α an den Stellen (010) und (011) von (00) zu (10). Betrachtet man die Freiheitsrelation von α, so erkennt man aber, daß es keine Belegung der primären Eingänge gibt, die diese Änderung an den primären Ausgängen sichtbar macht. Folglich kann man die Konstante 1 an dieser Stelle einsetzen und durch Konstantenpropagation eliminieren (siehe obige Abbildung).
Es ergibt sich insgesamt der Schaltkreis aus Abbildung C.1.

Durch diese Änderung von α zu α' wird aber die Freiheitsrelation von g geändert:

$$FR_g : \begin{array}{cc|cccc|} & c_2 & 0 & 0 & 1 & 1 \\ & c_3 & 0 & 1 & 0 & 1 \\ a_1 a_2 & & & & & \\ \hline 0\,0 & & 0 & 1 & 0 & 0 \\ 0\,1 & & \star & 0 & 1 & 1 \\ 1\,0 & & 1 & \star & 0 & \star \\ 1\,1 & & \star & 1 & 1 & 0 \\ \hline \end{array} \quad \Longrightarrow \quad FR_g : \begin{array}{cc|cccc|} & c_2 & 0 & 0 & 1 & 1 \\ & c_3 & 0 & 1 & 0 & 1 \\ a_1 a_2 & & & & & \\ \hline 0\,0 & & 0 & \star & 0 & \star \\ 0\,1 & & \star & 0 & 1 & 1 \\ 1\,0 & & 1 & 1 & 0 & 0 \\ 1\,1 & & \star & 1 & 1 & 0 \\ \hline \end{array}$$

Die Änderung von α_1 bewirkt, daß die Belegung $(0,0,0,1)$ von (a_1, a_2, c_2, c_3) nicht mehr als Eingabe von g vorkommen kann. Dies wurde aber bei der Synthese von g unter Vermeidung redundanter Stuck–at–Fehler vorausgesetzt. Der im Schaltkreis für g aus Abbildung C.1 markierte s-a-0–Fehler wird redundant, da man die Belegung $(0,0,0,1)$ von (a_1, a_2, c_2, c_3) an die Realisierung von g anlegen muß, um den Fehler zu testen.

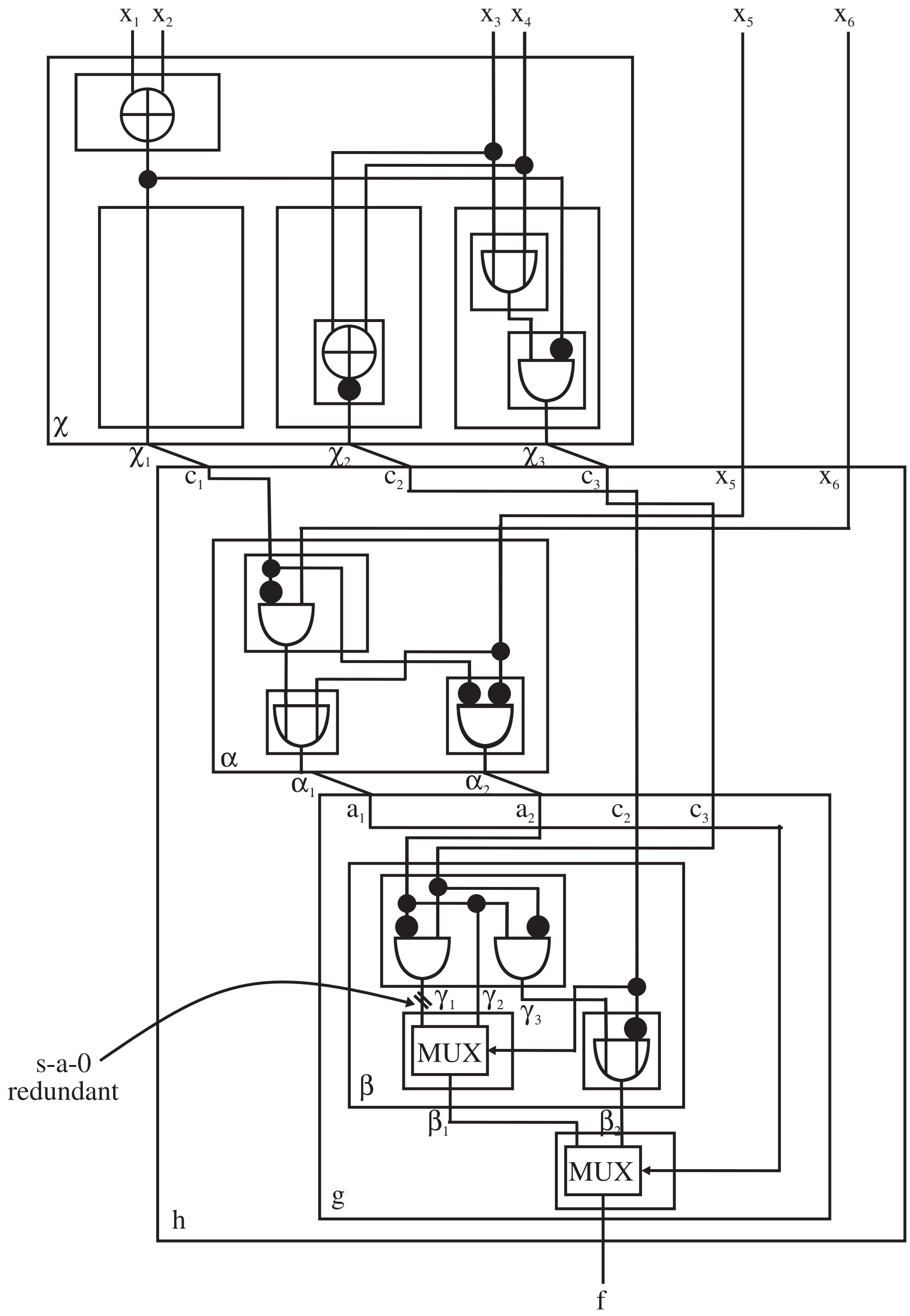


Abbildung C.1: Realisierung zu f mit redundantem s-a–Fehler.

Literaturverzeichnis

[AF81] V. K. Agarwal and A. S. F. Fung. Multiple Fault Testing of Large Circuits by Single Fault Test Sets. IEEE Transact. on Computers, C–30, pp. 855–865, November 1981.

[AH63] R. F. Arnold and M. A. Harrison. Algebraic Properties of Symmetric and Partially Symmetric Boolean Functions. IEEE Transact. on Electronic Computers, vol. EC–12, no. 3, pp. 244–251, June 1963.

[Ake78] S. Akers. Binary Decision Diagrams. IEEE Trans. on Computer–Aided Design, 27(6), pp. 509–516, 1978.

[Ash59] R. L. Ashenhurst. The Decomposition of Switching Functions. Proceedings of an International Symposium on the Theory of Switching, vol. 28 of Comp. Lab. of Harvard University, pp. 74–116, 1959.

[BHK+87] B. Becker, G. Hotz, R. Kolla, P. Molitor and H.G. Osthof. Hierarchical Design Based on a Calculus of Nets. In Proceedings of the 24th ACM/IEEE Design Automation Conference. pp. 649–653, 1987.

[BHM+84] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. Sangiovanni–Vincentelli. Logic Minimization Algorithms for VLSI Synthesis. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1984.

[BLM95] B. Bollig, M. Löbbing, I. Wegener. Simulated Annealing to Improve Variable Orderings for OBDDs. International Workshop on Logic Synthesis, Granlibakken (CA), 1995.

[BRB90] K. S. Brace, R. L. Rudell, R. E. Bryant. Efficient Implementation of a BDD Package. Proceedings of the 27th IEEE/ACM Design Automation Conference, pp. 40–45, 1990.

[Bre79] D. Brélaz. New Methods to Color Vertices of a Graph. CACM, 22, pp. 251–256, 1979.

[BRS+87] R. K. Brayton, R. Rudell, A. Sangiovanni–Vincentelli, A. R. Wang. MIS: A Multiple-Level Logic Optimization System. IEEE Trans. on CAD, CAD–6(11), Nov. 1987.

[Bry86] R. E. Bryant. Graph–Based Algorithms for Boolean Function Manipulation. IEEE Transact. on Computers, 35(8), pp. 677–691, 1986.

[BSM94] B. Bollig, P. Savicky, I. Wegener. On the Improvement of Variable Orderings for OBDDs. IFIP Workshop on Logic and Architecture Synthesis, Grenoble, pp. 71–80, Dec. 1994.

[CBM89] O. Coudert, C. Berthet, J. C. Madre. Verification of Synchronous Sequential Machines Based on Symbolic Execution. In J. Sifakis, editor, Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems, vol. 407 of Lecture Notes in Computer Science, Springer–Verlag, pp. 365–373, June 1989.

[CCM94] S. Chang, D. Cheng, M. Marek–Sadowska. Minimizing ROBDD Size of Incompletely Specified Multiple Output Functions. ED&TC, pp. 620–624, 1994.

[CM77] E. Cerny, M. A. Marin. An Approach to Unified Methodology of Combinational Switching Circuits. IEEE Transactions on Computers, vol. 26, pp. 745–756, 1977.

[CP90] R. Carraghan, P. M. Pardalos. An Exact Algorithm for the Maximum Clique Problem. Operations Research Letters, 9, pp. 375-382, 1990.

[Cur61] H. A. Curtis. A generalized tree circuit. J. Assoc. Comput. Mach., vol. 8, pp. 484–496, 1961.

[DBG95] R. Drechsler, B. Becker, N. Göckel. A Genetic Algorithm for Variable Ordering of OBDDs. International Workshop on Logic Synthesis, Granlibakken (CA), 1995.

[DBJT84] J. Darringer, D. Brand, W. Joyner, L. Trevillyan. LSS: A System for Production Logic Synthesis. IBM Journal of Research and Development, Vol. 28, No. 5, pp. 537–545, September 1984.

[DJBT81] J. Darringer, W. Joyner, L. Berman, L. Trevillyan. LSS: Logic Synthesis through Local Transformations. IBM Journal of Research and Development, Vol. 25, No. 4, pp. 365–388, Juli 1981.

[DS67] D. L. Dietmeyer, P. R. Schneider. Identification of Symmetry, Redundancy and Equivalence of Boolean Functions. IEEE Transact. Electron. Comput., vol. 16, pp. 804–817, December 1967.

[EH78] C. R. Edwards, S. L. Hurst. A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods. IEEE Trans. on Computers, vol. C-27, no. 11, pp. 985–997, November 1978.

[FFK88] M. Fujita, H. Fujisawa, N. Kawato. Evaluation and Improvements of Boolean Comparison Methods Based on Binary Decision Diagrams. IEEE Int'l Conf. on CAD, pp. 2–5, 1988.

[FMK91] M. Fujita, Y. Matsunaga, T. Kakuda. On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis. European Conf. on Design Automation, pp. 50–54, 1991.

[FOH93] H. Fuji, G. Oomoto, C. Hori. Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams. Int'l Conf. on CAD, pp. 38–41, 1993.

[FYBS93] E. Felt, G. York, R. Brayton, A. Sangiovanni-Vincentelli. Dynamic Variable Reordering for BDD Minimization. European Conf. on Design Automation, pp. 130–135, 1993.

[GJ79] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of **NP**–completeness. New York: Freeman, 1979.

[HDB96] A. Hett, R. Drechsler, B. Becker. Alternative Implementation of BDD–Packages by Multi–Operand Synthesis. European Conf. on Design Automation, 1996.

[HOI89] T. Hwang, R. M. Owens and M. J. Irwin. Exploiting Communication Complexity for Multilevel Logic Synthesis. IEEE Trans. Computer–Aided Design, vol. 9, pp. 1017–1027, Oct. 1990.

[HOI92] T. Hwang, R. M. Owens and M. J. Irwin. Efficiently Computing Communication Complexity for Multilevel Logic Synthesis. IEEE Trans. Computer–Aided Design, vol. 11, pp. 545–554, Mai 1992.

[Hot60] G. Hotz. Zur Reduktionstheorie der Booleschen Algebra. In Colloquium über Schaltkreis– und Schaltwerk–Theorie, Birkhäuser Verlag, 1960.

[Hot65] G. Hotz. Eine Algebraisierung des Syntheseproblems für Schaltkreise. EIK Journal of Information Processing and Cybernetics, 1:185–205, 209–231, 1965.

[Hot74] G. Hotz. Schaltkreistheorie. Walter de Gruyter, 1974.

[ICE91] ASIC Outlook 1991. Integrated Circuit Engineering Corporation, Scottsdale, Arizona, 1991.

[ISY91] N. Ishihura, H. Sawada, S. Yajima. Minimization of Binary Decision Diagrams Based on Exchanges of Variables. IEEE Int'l Conf. on CAD, pp. 472–475, 1991.

[Kar63] R. M. Karp. Functional Decomposition and Switching Circuit Design. J. Soc. Indust. Appl. Math. , vol. 11, no. 2, pp. 291–335, June 1963.

[KD91] B.–G. Kim and D. L. Dietmeyer. Multilevel Logic Synthesis of Symmetric Switching Functions. IEEE Transact. on CAD, vol. 10, no. 4, pp. 436–446, April 1991.

[KMO89] R. Kolla, P. Molitor, H.G. Osthof. Einführung in den VLSI–Entwurf. Leitfäden und Monographien der Informatik. B.G. Teubner Verlag, Stuttgart, 1989.

[Lee59] C. Lee. Representations of Switching Circuits by Binary Decision Diagrams. Bell System Technical Journal, 38, pp. 985–999, 1959.

[LP81] H. R. Lewis, C. H. Papademitriou. Elements of the Theory of Computation. Prentice Hall, 1981.

[LPPS93] Y. Lai, K. Pan, M. Pedram, S. Sastry. FGMAP: A Technology Mapping Algorithm for Look-Up Table Type FPGAs Based on Function Graphs. In *Workshop Notes IWLS*, pp. 9b1–9b4, Mai 1993.

[LPV94] Y.–T. Lai, M. Pedram, S. B. K. Vrudhula. EVBDD–Based Algorithms for Integer Linear Programming, Spectral Transformation, and Functional Decomposition. IEEE Transact. on CAD, vol. 13, no. 8, pp. 959–446, August 1994.

[Lup58] O. B. Lupanov. A Method of Circuit Synthesis. Izv. VUZ Radioviz 1, pp. 120–140, 1958.

[Meh84b] K. Mehlhorn. Data Structures and Algorithms 2: Graph Algorithms and NP–Completeness. EATCS Monographs on Theoretical Computer Science, Springer–Verlag, Berlin, Heidelberg, New York, Tokio, 1984.

[Mel96] S. Melchior. Diplomarbeit, Fachbereich 14 Informatik, Universität des Saarlandes, erscheint voraussichtlich 1996.

[MMD94] D. Möller, P. Molitor, R. Drechsler. Symmetry Based Variable Ordering for ROBDDs. IFIP Workshop on Logic and Architecture Synthesis, Grenoble, pp. 47–53 Dec. 1994.

[MMW93] D. Möller, J. Mohnke, M. Weber. Detection of Symmetry of Boolean Functions Represented by ROBDDs. IEEE Int'l Conf. on CAD, pp. 680–684, 1993.

[MNS+90] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, A. Sangiovanni–Vincentelli. Logic Synthesis for Programmable Gate Arrays. Proceedings 27th ACM/IEEE Design Automation Conference DAC, pp. 620–625, June 1990.

[Mol90] P. Molitor. Vorlesungsskript Logikminimierung 90/91.

[Mor92] C. Morgenstern. A New Backtracking Heuristic for Rapidly Four-Coloring Large Planar Graphs. Technical Report CoSc-1992-2, Texas Christian University, Fort Worth, Texas, 1992.

[MS94] P. Molitor, C. Scholl. Communication Based Multilevel Synthesis for Multi-Output Boolean Functions. In Proceedings of the 4th Great Lakes Symposium on VLSI, Notre Dame, Indiana, pp. 101–104, 1994.

[MSBS91] R. Murgai, N. Shenoy, R. K. Brayton, A. Sangiovanni-Vincentelli. Improved Logic Synthesis Algorithms for Table Look Up Architectures. In *Proc. Int'l Conf. on Computer Aided Design*, pp. 564-567, 1991.

[MWBS88] S. Malik, A. Wang, R. Brayton, A. Sangiovanni–Vincentelli. Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. IEEE Int'l Conf. on CAD, pp. 6–9, 1988.

[Pau76] W. J. Paul. Realizing Boolean Functions on Disjoint Sets of Variables. TCS 2, pp. 383–396, 1976.

[PS95] S. Panda, F. Somenzi. Who are the Variables in Your Neighborhood. IEEE Int'l Conf. on CAD, pp. 74–77, 1995.

[PSP94] S. Panda, F. Somenzi, B. F. Plessier. Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. IEEE Int'l Conf. on CAD, pp. 628–631, Nov. 1994.

[Red81] N. P. Redkin. Minimal realization of a binary adder. Probl. Kibern. 38, pp. 181–216, 1981.

[RK62] J. P. Roth and R. M. Karp. Minimization over Boolean Graphs. IBM Journal, vol. 6, no. 2, pp. 227–238, 1962.

[Rud93] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams, IEEE Int'l Conf. on CAD, pp. 42–47, 1993.

[S+92] E. Sentovich et al.. SIS: A System for Sequential Circuit Synthesis. Department of EE and CS, UC Berkeley, Mai 1992.

[Sch93a] C. Scholl. Mehrstufige Logiksynthese unter Ausnutzung von Symmetrien und nichttrivialen Zerlegungen. Diplomarbeit, Fachbereich 14 Informatik, Universität des Saarlandes, 1993.

[SM93] C. Scholl, P. Molitor. Mehrstufige Logiksynthese unter Ausnutzung von Symmetrien und nichttrivialen Zerlegungen. Technischer Bericht 02/1993, SFB 124, Teilprojekt B6, 1993.

[MS94] P. Molitor, C. Scholl. Communication Based Multilevel Synthesis for Multi-Output Boolean Functions. In Proceedings of the 4th Great Lakes Symposium on VLSI, Notre Dame, Indiana, pp. 101–104, 1994.

[SM94] C. Scholl, P. Molitor. Efficient ROBDD Based Computation of Common Decomposition Functions of Multi-Output Boolean Functions. In Proceedings of IFIP Workshop on Logic and Architecture Synthesis, Grenoble, France, pp. 61–70, December 1994.

[SM95a] C. Scholl, P. Molitor. Efficient ROBDD Based Computation of Common Decomposition Functions of Multi-Output Boolean Functions. In Logic and Architecture Synthesis, State-of-the-art and novel approaches, edited by Gabriele Saucier and Anne Mignotte, Chapman&Hall, pp. 57-63, 1995.

[SM95b] C. Scholl, P. Molitor. Communication Based FPGA Synthesis for Multi-Output Boolean Functions. In Proceedings of the Asia and South Pacific Design Automation Conference, Chiba, Japan, pp. 279–288, August 1995.

[SMHM96] C. Scholl, S. Melchior, G. Hotz, P. Molitor. Minimizing ROBDD Sizes of Incompletely Specified Boolean Functions by Exploiting Strong Symmetries. Submitted for publication, 1996.

[Sha38] C. E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. Trans. AIEE, vol. 57, pp. 713–723, 1938.

[Sha49] C. E. Shannon. The Synthesis of Two–Terminal Switching Circuits. Bell Systems Technical Journal 28, pp. 59–98, 1949.

[Sla60] J. Slansky. Conditional-Sum Addition Logic. IEEE Trans. Elect. Comp. 9, pp. 226–231, 1960.

[Sp91] U. Sparmann. Strukturbasierte Testmethoden für arithmetische Schaltkreise. Dissertation, Fachbereich 14 Informatik, Universität des Saarlandes, 1991.

[TY88] N. Takagi, S. Yajima. An On–Line Error–Detectable Divider with a Redundant Binary Representation and a Residue Code. In Proceedings of Int. Symp. on Fault–Tolerant Computing, pp. 174–179, 1988.

[W95] B. Wurth. Personal Communication, 1995.

[Wal64] C. S. Wallace. A Suggestion for a Fast Multiplier. IEEE Trans. on Computers 13, pp. 14–17, 1964.

[WEA95] B. Wurth, K. Eckl, K. Antreich. Functional Multi–Output Decomposition: Theory and an Implicit Algorithm. Proceedings 32nd IEEE/ACM Design Automation Conference DAC, June 1995.

[Weg87] I. Wegener. The Complexity of Boolean Functions. Wiley–Teubner, 1987.

[Weg89] I. Wegener. Effiziente Algorithmen für grundlegende Funktionen. Teubner Verlag, 1989.