Diplomarbeit

Beziehungen zwischen Branchingprogrammgröße und Formelgröße boolescher Funktionen

Oliver Giel

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

20. März 2000

Betreuer:

Prof. Dr. Ingo Wegener

Oliver Giel, Händelstr. 41, D-45657 Recklinghausen
Matrikelnummer 0046638, Fachbereich Informatik, Universität Dortmund

Inhaltsverzeichnis

1 Grundlegende Definitionen

In diesem Kapitel werden die zentralen Begriffe und Modelle dieser Arbeit eingeführt. Die meisten der aufgeführten Definitionen sind angelehnt an Wegener (1987, 1989, 1993, 2000).

Boolesche Funktionen

Eine *boolesche Funktion* $f \in B_{n,m}$ mit n booleschen Variablen als Eingabe und m booleschen Ausgaben ist eine Funktion $f\colon \{0,1\}^n \rightarrow \{0,1\}^m$. Wir betrachten hier boolesche Funktionen mit nur einer Ausgabe, d. h. Funktionen aus $B_{n,1}$, oder kurz B_n. In B_2 gibt es genau 16 verschiedene Funktionen, die formal von beiden Eingabevariablen abhängen. Mit $B_2^* \subset B_2$ bezeichnen wir diejenigen Funktionen innerhalb von B_2, die *essentiell* (und nicht nur formal) von beiden Variablen abhängen. Die Menge $U_2 \subset B_2$ enthält alle Funktionen aus B_2 mit Ausnahme von EXOR und NEXOR.

B_2		
	B_2^*	
	$\mathrm{AND}(a,b) = a \wedge b$	
$\mathrm{ID}_1(a,b) = a$	$\mathrm{NAND}(a,b) = \overline{a \wedge b}$	
$\mathrm{ID}_2(a,b) = b$	$<(a,b) = \overline{a} \wedge b$	
$\mathrm{NOT}_1(a,b) = \overline{a}$	$>(a,b) = a \wedge \overline{b}$	$\mathrm{EXOR}(a,b) = a \oplus b$
$\mathrm{NOT}_2(a,b) = \overline{b}$	$\mathrm{OR}(a,b) = a \vee b$	$\mathrm{NEXOR}(a,b) = \overline{a \oplus b}$
$0(a,b) = 0$	$\mathrm{NOR}(a,b) = \overline{a \vee b}$	
$1(a,b) = 1$	$\leq(a,b) = \overline{a} \vee b$	
	$\geq(a,b) = a \vee \overline{b}$	
U_2		

Tabelle 1.1: Funktionen in B_2.

Schaltkreise und Formeln

Die Menge der booleschen Funktionen, die von Schaltkreisen realisiert werden können, und die erforderliche Schaltkreisgröße hängt maßgeblich von der Wahl der Basis Ω ab, aus der die einzelnen Bausteine (Gatter) eines Schaltkreises entnommen sind. Eine Basis heißt *vollständig*, wenn sich alle booleschen Funktionen in dieser Basis darstellen lassen. Häufig wird als Basis $\{\wedge, \vee, \neg\}$ gewählt. (Dabei bezeichnet $\neg \notin B_2$ die Negationsfunktion für eine boolesche Variable.) Deren Vollständigkeit kann man leicht einsehen, da die Disjunktive Normalform (DNF) jeder booleschen Funktion in dieser Basis darstellbar ist. Wir gehen hier i. Allg. von der vollständigen Basis B_2 aus und nennen entsprechende Schaltkreise B_2-Schaltkreise oder einfach nur Schaltkreise.

Definition 1.1. Ein *B_2-Schaltkreis* S mit n booleschen Eingängen $x_1, \ldots, x_n$ besteht aus der Aufzählung seiner Bausteine $G_1, \ldots, G_c$ und seiner Ausgänge $A_1, \ldots, A_m$. Ein Baustein G_k wird durch Angabe seines Typs $\omega \in B_2$ und der Aufzählung seiner beiden Eingänge beschrieben. Eingänge von G_k dürfen die Konstanten 0 und 1, die Eingänge des Schaltkreises $x_1, \ldots, x_n$ und alle Bausteine $G_1, \ldots, G_{k-1}$ sein. Ausgang des Schaltkreises kann jeder Baustein, jeder Input x_i und jede der beiden Konstanten sein.

Gewöhnlich werden Schaltkreise durch gerichtete, azyklische Graphen veranschaulicht. Der Graph $G = (V, E)$ für einen Schaltkreis S, hat die Knotenmenge $V = \{x_1, \ldots, x_n, G_1 \ldots, G_c\}$ (ggf. um 0 und/oder 1 erweitert, falls diese Konstanten in S vorkommen). In der Kantenmenge E ist die Kante (v_i, v_j) genau dann enthalten, wenn der Baustein, der Eingang des Schaltkreises oder die Konstante an v_i ein Eingang für den Baustein an v_j ist. Für kommutative Bausteine aus B_2 ist diese Darstellung völlig ausreichend. Für nicht kommutative Bausteine, wie z. B. $<(a, b) = \overline{a} \wedge b$, ist eine zusätzliche Unterscheidung eingehender Kanten erforderlich.

Beispiel 1.2. Abbildung 1.1(a) zeigt einen Schaltkreis mit Eingängen x und y.

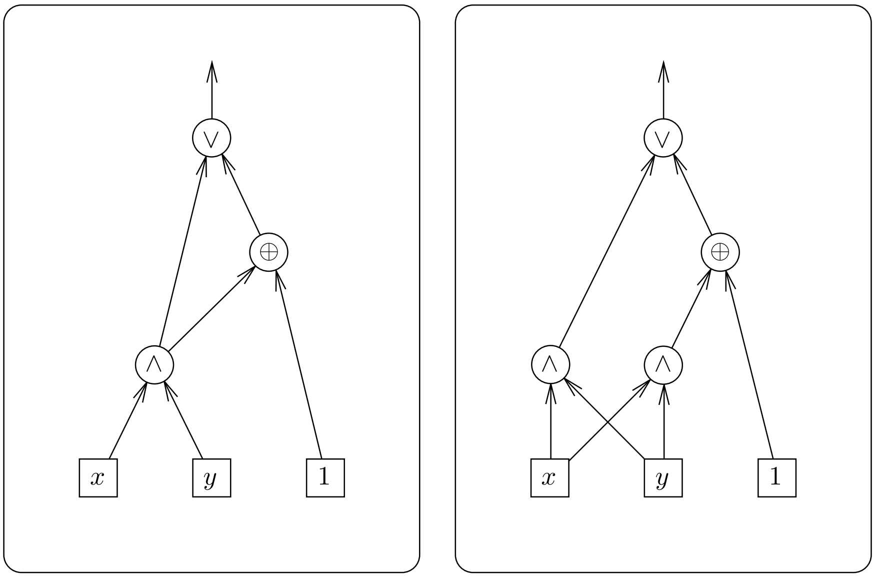
Definition 1.3.

i) Die *Größe* oder *Komplexität* eines B_2-Schaltkreises S heißt $\mathrm{C}(S)$ und ist die Zahl c seiner Bausteine.

ii) Die *Tiefe* von S, $\mathrm{D}(S)$, ist die größte Anzahl Bausteine auf einem gerichteten Pfad in S.

iii) Für eine boolesche Funktion f bezeichnet $\mathrm{C}(f)$ die Größe eines kleinsten B_2-Schaltkreises für f und $\mathrm{D}(f)$ die Tiefe eines B_2-Schaltkreises für f mit minimaler Tiefe.

Formeln sind eine eingeschränkte Klasse von Schaltkreisen. Sie sollen boolesche Ausdrücke mit binären Operatoren, wie z. B. $(x \wedge y) \vee ((x \wedge y) \oplus 1)$, widerspiegeln.



(a) Schaltkreis (b) Formel

Abbildung 1.1: Schaltkreis und Formel.

Definition 1.4. Eine B_2-Formel ist ein B_2-Schaltkreis mit nur einem Ausgang, bei dem der Fan-out der Bausteine durch 1 beschränkt ist.

Der Schaltkreis in Abbildung 1.1(b) ist eine Formel, die aus dem Schaltkreis in Abbildung 1.1(a) durch Duplizieren des $\wedge$-Bausteins mit zu großem Fan-out entstanden ist. Im Allgemeinen kann jeder Schaltkreis durch wiederholtes Kopieren der Bausteine von Subschaltkreisen mit zu großem Fan-out in eine Formel verwandelt werden. Durch den beschränkten Fan-out der Bausteine müssen Formeln mehrfach verwendete Zwischenergebnisse mehrfach in getrennten Subschaltkreisen berechnen.

Wir nehmen immer an, dass in einer Formel keine „überflüssigen“ Bausteine mit Fan-out 0 vorkommen. Dann bilden die inneren Knoten des zugrundeliegenden Graphen einen wurzelgerichteten Baum. Die Baumeigenschaft wird nur an den „Blättern“, den Variablen und Konstanten, gestört, weil deren Fan-out nicht beschränkt ist.

Definition 1.5. Eine Formel heißt *read-once*, wenn jeder ihrer Eingänge (Variablen) Fan-out 1 hat.

Das natürliche Maß für die Größe einer Formel ist ihre Schaltkreisgröße, also die Zahl ihrer Bausteine. Diese entspricht genau der Zahl innerer Knoten des zugehörigen Graphen, welcher bei B_2-Formeln ein Binärbaum ist. Wegen der Baumeigenschaft ist auch die folgende Definition sinnvoll. Sie zielt darauf ab, die Vorkommen von Variablen und Konstanten in dem booleschen Ausdruck, den die Formel darstellt, zu zählen. So soll z. B. der Formel $(x \wedge y) \vee ((x \wedge y) \oplus 1)$ in Abbildung 1.1(b) die Größe 5 zugeschrieben werden.

Definition 1.6. Die *Größe* einer B_2-Formel F, $\mathrm{L}(F)$, ist gleich der Summe der Fan-outs aller ihrer Eingänge (Variablen) und der Konstanten. Für eine boolesche Funktion f bezeichnet $\mathrm{L}(f)$ die Größe einer kleinsten B_2-Formel für f. Mit $\mathrm{L}^*(F)$ und $\mathrm{L}^*(f)$ wird entsprechend die Formelgröße über der Basis $\{\wedge, \vee, \neg\}$ bezeichnet.

$\mathrm{L}(f)$ und die Zahl der Bausteine in einer kleinsten B_2-Formel für f unterscheiden sich nur um die additive Konstante 1: Der Schaltkreisgraph einer kleinsten B_2-Formel für f mit n Bausteinen ist ein Binärbaum und ein Binärbaum mit n inneren Knoten besitzt genau $n+1$ Blätter.

Negationen ($\neg$) sind im L^*-Maß offensichtlich „umsonst“, sie erhöhen nicht die Zahl der Blätter, denn der zugrunde liegende Graph ist kein Binärbaum. Dies ist nicht etwa eine schwerwiegende Schwäche des Modells. O.B.d.A. kann man für Formeln über $\{\wedge, \vee, \neg\}$ immer annehmen, dass Negationen nur in Form von negierten Variablen vorkommen. Dies kann man erreichen, indem man, an der Wurzel beginnend, rekursiv die deMorgan-Regeln $\overline{(a \wedge b)} = \overline{a} \vee \overline{b}$ und $\overline{(a \vee b)} = \overline{a} \wedge \overline{b}$ anwendet. Dabei bleibt die Gesamtzahl der AND- und OR-Bausteine unverändert und Negationen werden zu den Eingängen propagiert. Das Negieren der Eingänge wird dann schließlich als kostenlos angesehen. Man kann dies damit rechtfertigen, dass dort, wo Schaltkreise eingesetzt werden, die Eingänge meistens auch in negierter Form zur Verfügung stehen. Dies ist z. B. in sequentiellen Schaltwerken der Fall, wo die Eingänge von Schalkreisen an Ausgänge von Flip-Flops angeschlossen werden, die immer zwei zueinander inverse Ausgänge besitzen.

Manchmal wird L^* auch als Formelgröße über der Basis U_2 (vgl. Tabelle 1.1) definiert. Man erhält für boolesche Funktionen f im Wesentlichen dieselbe Formelgröße wie über $\{\wedge, \vee, \neg\}$, denn minimale Formeln lassen sich (bis auf eine unbedeutende Ausnahme) ohne Größenzuwachs in der jeweils anderen Basis simulieren. Nach obiger Betrachtung dürfen wir annehmen, dass in einer minimalen $\{\wedge, \vee, \neg\}$-Formel Negationen nur in Form von negierten Variablen vorkommen. Dann gibt es oberhalb der Literale nur $\wedge$- und $\vee$-Bausteine, die wir in U_2-Formeln auch zur Verfügung haben. Die Negationsbausteine an den Variablen integrieren wir in die unterste Ebene der AND- und OR-Bausteine, indem wir diese entsprechend bei einer negierten Variablen als Eingang durch $<$-, $>$-, $\leq$- oder $\geq$-Bausteine ersetzt (vgl. Tabelle 1.1), bei zwei negierten Variablen als Eingang durch NAND- oder NOR-Bausteine. Bei dieser Simulation entstehen keine zusätzlichen Kosten. Lediglich $\{\wedge, \vee, \neg\}$-Formeln aus nur einer negierten Variablen ($\overline{x}$) mit Kosten 1 können wir als U_2-Formel nur mit Kosten 2 realisieren, z. B. als $\mathrm{NOT}_1(x, x)$. Bei der umgekehrten Simulation beobachtet man, dass 0- und 1-Bausteine einer U_2-Formel durch Konstanten 0 und 1 ersetzt werden können, NOT_1- und NOT_2-Bausteine durch $\neg$-Bausteine und Identerbausteine ID_1 und ID_2 sogar ganz eingespart werden können. Durch diese Ersetzungen können die Kosten nur sinken. Die übrigen Bausteine können durch jeweils einen $\wedge$- oder $\vee$-Baustein und einen (kostenlosen) $\neg$-Baustein ersetzt werden.

Im Allgemeinen hängt für Formeln – abgesehen von Konstanten Faktoren – die Darstellungskraft einer vollständigen binären Basis allein vom Vorhandensein von EXOR- oder NEXOR-Bausteinen ab. Für die Parity-Funktion $x_1 \oplus \cdots \oplus x_n$ ist bekannt, dass $\mathrm{L}^*(x_1 \oplus \cdots \oplus x_n) \geq n^2$, während offensichtlich $\mathrm{L}(x_1 \oplus \cdots \oplus x_n) = n$.

Definition 1.7. Eine B_2-Formel mit Größe ℓ heißt *balanciert*, wenn ihre Tiefe $d = \lceil \log \ell \rceil$ ist.

Branchingprogramme

Branchingprogramme berechnen boolesche Funktionen. Sie können auf verschiedene Weisen definiert werden, je nachdem, welchen Blickwinkel man einnehmen möchte. Hier sollen zwei verschiedene, gleichwertige Definitionen genügen. Die erste Definition greift auf straight-line Programme zurück, die uns später noch in anderer Ausprägung begegnen werden, während sich die zweite Definition auf Graphen stützt.

Definition 1.8. Ein *ite straight-line Programm* ist eine Folge von Anweisungen $I_1, \ldots, I_s$. Eine einzelne Anweisung ist von der Form $I_j = \mathit{ite}(C_j, T_j, E_j)$. Die Bedingung (condition) C_j, der „Then-Zweig“ T_j und der „Else-Zweig“ E_j sind aus $\{0, 1, x_1, \ldots, x_n, I_1, \ldots, I_{j-1}\}$, wobei $x_1, \ldots, x_n$ Variablen sind. Jede Anweisung I_j repräsentiert eine boolesche Funktion $f_j \in B_n$, die rekursiv durch $\mathit{ite}(x, y, z) := xy \vee \overline{x}z$ (“if x then y else z”) definiert ist. Das Programm berechnet f_s.

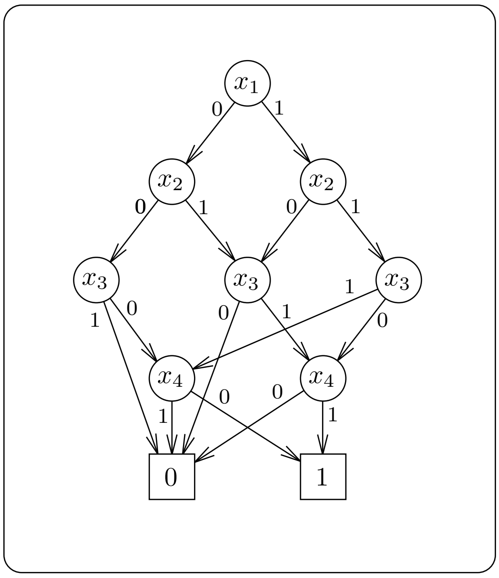
Definition 1.9. Ein *Branchingprogramm* (BP) ist ein *ite* straight-line Programm, bei dem als Bedingungen C_j nur Variablen $\{x_1, \ldots, x_n\}$ erlaubt sind und T_j und E_j aus $\{0, 1, I_1, \ldots, I_{j-1}\}$ sind.

Definition 1.10. Ein *Branchingprogramm* (BP) besteht aus einem gerichteten, azyklischen Graphen und einer Markierung der Knoten und Kanten. Jeder Knoten hat entweder Fan-out 0 oder Fan-out 2. Jeder innere Knoten (Fan-out 2) ist mit einer Variablen $x_i \in \{x_1, \ldots, x_n\}$ markiert und eine seiner ausgehenden Kanten mit 0, die andere Kante mit 1. Senkenknoten (Fan-out 0) sind mit booleschen Konstanten, 0 oder 1, markiert. Die Berechnung für $f(a)$, $a \in \{0,1\}^n$, startet an einem ausgezeichneten inneren Knoten, der Quelle. An einem x_i-Knoten wird die Kante mit Markierung a_i gewählt. Ein Branchingprogramm stellt $f \in B_n$ dar, falls der Berechnungspfad für alle $a \in \{0,1\}^n$ an einer Senke mit Markierung $f(a)$ endet.

Beispiel 1.11. Abbildung 1.2 stellt dasselbe Branchingprogramm für $f(x_1, x_2, x_3, x_4) = [x_1 + x_2 + x_3 + x_4 \equiv 0 \bmod 3]$ in beiden Modellen gegenüber. Die Schreibweise „$[\mathit{Ausdruck}]$“ steht für den booleschen Wert 1, falls *Ausdruck* wahr ist, und sonst für 0.

$$\begin{aligned} I_1 &= ite(x_4, 0, 1), \\ I_2 &= ite(x_4, 1, 0), \\ I_3 &= ite(x_3, 0, I_1), \\ I_4 &= ite(x_3, I_2, 0), \\ I_5 &= ite(x_3, I_1, I_2), \\ I_6 &= ite(x_2, I_4, I_3), \\ I_7 &= ite(x_2, I_5, I_4), \\ I_8 &= ite(x_1, I_7, I_6) \end{aligned}$$

(a) als *ite* straight-line Programm



(b) als Graph

Abbildung 1.2: BP für $f(x_1, x_2, x_3, x_4) = [x_1 + x_2 + x_3 + x_4 \equiv 0 \bmod 3]$.

Branchingprogramme der beiden Modelle können ineinander überführt werden: Jeder innere Knoten mit Markierung x_i, 1-Nachfolger v_k und 0-Nachfolger v_ℓ eines Branchingprogramms nach Definition 1.10 wird durch eine Anweisung $ite(x_i, I_k, I_\ell)$ ersetzt, wobei für Referenzen auf Senken direkt die entsprechende Konstante für I_k oder I_ℓ eingesetzt wird. Wenn wir die Knoten in einer umgekehrten topologischen Reihenfolge abarbeiten, erhalten wir ein *ite* straight-line Programm, das Definition 1.9 genügt. Die Zahl der Anweisungen entspricht der Zahl der inneren Knoten. Wenn andererseits ein Branchingprogramm nach Definition 1.9 mit Anweisungen $I_1, \ldots, I_s$ gegeben ist, konstruieren wir einen Branchingprogramm-Graphen mit s inneren Knoten $v_1, \ldots, v_s$ und zwei zusätzlichen Senkenknoten für 0 und 1. Die Markierung von v_i wird die Bedingung C_i, die 1-Kante von v_i führt zum Knoten v_k, falls $T_i = I_k$, oder zur entsprechenden Senke, falls $T_i = 0$ oder $T_i = 1$. Für die 0-Kante verfahren wir analog.

Es ist klar, dass in Branchingprogrammen nach Definition 1.10 eine oder zwei Senken für 0 und 1 in jedem Fall genügen. Die Zahl der Knoten insgesamt in einem solchen Branchingprogramm mit s inneren Knoten ist dann $s+1$ oder $s+2$, die Zahl der Kanten genau $2s$. Daher ist s ein gutes Maß für die Branchingprogrammgröße.

Definition 1.12. i) Die *Größe* eines Branchingprogramms ist gleich der Anzahl seiner inneren Knoten oder Anweisungen. Für eine boolesche Funktion f bezeichnet BP(f) die Branchingprogrammgröße von f, d. h. die Größe eines kleinsten Branchingprogramms für f.

ii) Die *Tiefe* eines Branchingprogramms ist gleich der Länge (Anzahl der Kanten) seines längsten gerichteten Pfades.

Definition 1.13. Ein Branchingprogramm heißt *geschichtet*, wenn alle Knoten so auf Leveln angeordnet sind, dass alle Knoten eines Levels dieselbe Variable testen und Kanten nur von einem Level zum darauf folgenden Level verlaufen. Die *Breite* eines geschichteten Branchingprogramms ist die größte Anzahl Knoten auf einem seiner Level.

Definition 1.14. Ein *inkonsistenter Pfad* eines Branchingprogramms ist ein gerichteter Pfad, der für eine Variable x_i sowohl eine 0-Kante enthält, die einen x_i-Knoten verlässt, als auch eine 1-Kante enthält, die einen x_i-Knoten verlässt.

Ein Berechnungspfad für einen Input $a \in \{0, 1\}^n$ enthält keine inkonsistenten Pfadabschnitte, da 0- und 1-Kanten konsistent zu a gewählt sind. Gleichwohl erlauben wir die Existenz inkonsistenter Pfade in Branchingprogrammen.

2 Einleitung und Historie

Schaltkreise, Formeln und Branchingprogramme gelten als die wichtigsten Rechenmodelle für boolesche Funktionen. Ihre zugehörigen Größenmaße $\mathrm{C}(f)$, $\mathrm{L}(f)$ und $\mathrm{BP}(f)$ sind daher die wichtigsten Komplexitätsmaße für boolesche Funktionen. Um die „Darstellungskraft“ der Modelle zu vergleichen, werden ihre Komplexitätsmaße zueinander in Beziehung gesetzt und versucht, möglichst gute obere und untere Schranken nachzuweisen. Dieses Kapitel gibt einen kurzen Überblick über einige grundlegende Resultate, die Beziehungen zwischen Schaltkreisgröße, Formelgröße und Branchingprogrammgröße betreffen, und stellt zwei aktuellere Ergebnis vor.

Ältere Resultate

Die Aussagen im folgenden Satz gelten als wohl bekannt (siehe z. B. die Monographien von Wegener (1987, 2000)).

Satz 2.1. *Für alle $f \in B_n$ gilt:*

i) $C(f) \leq L(f) - 1$,

ii) $C(f) \leq 3 \cdot BP(f)$,

iii) $L(f) \leq 2^{D(f)}$,

iv) $BP(f) \leq L^*(f)$ *und*

v) $L^*(f) \leq O\big(L(f)^{2{,}096}\big)$.

Beweis/Beweisideen.

i) Jede B_2-Formel der Größe ℓ entspricht einem Graphen mit $\ell - 1$ inneren Knoten oder Bausteinen, die einen Binärbaum bilden. Eine kleinste B_2-Formel für f ist ein Schaltkreis mit $\mathrm{L}(f) - 1$ Bausteinen.

ii) Ein Branchingprogramm kann auch als Schaltkreis mit *ite*-Bausteinen aufgefasst werden. Ein x_i-Knoten entspricht einem *ite*-Baustein ($ite(x, y, z) := xy \vee \overline{x}z$), dessen x-Eingang mit dem x_i-Eingang des Schaltkreises verbunden ist und dessen y- und z-Eingänge mit dem 1- bzw. 0-Nachfolger verbunden sind. Die Senken entsprechen im Schaltkreis den Konstanten 0 und 1. Jeder *ite*-Baustein kann nun durch drei B_2-Bausteine ersetzt werden (vgl. Tabelle 1.1).

iii) In einem Schaltkreis für f mit minimaler Tiefe $d = \mathrm{D}(f)$ duplizieren wir solange die Bausteine der Subschaltkreise mit zu großem Fan-out, bis jeder Baustein nur noch Fan-out 1 hat. Die inneren Knoten des Schaltkreises (d. h. die Bausteine) entfalten wir so zu einem binären Baum mit Tiefe $d-1$, der zusammen mit den Variablen und Konstanten eine Formel für f mit Tiefe d bildet. Der Graph dieser Formel enthält höchstens $2 \cdot 2^{d-1} = 2^d$ Kanten, die von den Variablen und Konstanten zu Bausteinen verlaufen. Die Summe der Fan-outs über alle Variablen und Konstanten der Formel ist daher höchstens 2^d.

iv) Wir konstruieren ein Branchingprogramm mit $\mathrm{L}^*(f)$ inneren Knoten. Unsere Konstruktion erfolgt induktiv über $\ell = \mathrm{L}^*(f)$. Für $\ell = 1$ besteht eine minimale $\{\wedge, \vee, \neg\}$-Formel für f nur aus einer Variablen x_i, einer negierten Variablen $\overline{x}_i$ oder einer Konstanten. Entsprechende Branchingprogramme haben Größe 0 bzw. 1. Im Induktionschritt betrachten wir eine minimale Formel mit Größe $\ell > 1$ und wir können annehmen, dass die Aussage für Formeln der Größen $1, \ldots, \ell-1$ richtig ist. Falls der Baustein an der Wurzel ein $\neg$-Baustein ist, steht unterhalb davon eine Formel für $\overline{f}$. Es ist $\mathrm{BP}(f) = \mathrm{BP}(\overline{f})$, denn es genügt, die Markierungen der Senken auszutauschen. Wir können o.B.d.A. annehmen, dass an der Wurzel ein $\wedge$-Baustein steht, denn jeder $\vee$-Baustein kann mit den deMorgan-Regeln durch einen $\wedge$-Baustein und einige $\neg$-Bausteine ersetzt werden. Dadurch wächst die Formelgröße in unserem Kostenmaß nicht an und mit $\neg$-Bausteinen können wir auf die obige Weise ohne Zusatzkosten im Branchingprogramm umgehen. Dann ist $f = g \wedge h$, und g und h sind durch (Sub-)Formeln mit Größe ℓ_g bzw. ℓ_h dargestellt. Es gilt $\ell_g + \ell_h = \ell$. Da $\ell_g, \ell_h < \ell$, können g und h gemäß Induktionsannahme durch Branchingprogramme G und H mit ℓ_g bzw. ℓ_h inneren Knoten dargestellt werden. Ein Branchingprogramm F für f erhalten wir durch AND-Synthese von G und H, bei der die 1-Senke von G durch die Quelle von H ersetzt wird. F hat dann $\ell_g + \ell_h = \ell$ innere Knoten.

v) In jeder B_2-Formel F mit $\ell \geq 2$ Blättern gibt es eine Subformel G, die zwischen 1/3 und 2/3 der Blätter besitzt: Da F mehr als $(2/3)\ell$ Blätter hat, gibt es in F einen oder mehrere Bausteine, die Wurzeln von ineinander geschachtelter (Sub-)Formeln mit mindestens $(2/3)\ell$ Blättern sind. (Der Wurzelbaustein von F gehört in jedem Fall dazu.) Wir betrachten den untersten dieser Bausteine. Dessen Söhne sind Wurzeln von Subformeln mit weniger als $(2/3)\ell$ Blättern. Wenigstens einer der Söhne ist Wurzel einer Subformel mit mindestens $(1/3)\ell$ Blättern, weil der Vaterknoten Wurzel einer Subformel mit mindestens $(2/3)\ell$ Blättern ist.

Die Beweisidee beruht darauf, die gegebene B_2-Formel F rekursiv aufzuteilen. Dazu wird in F eine Subformel G gewählt, sodass $\mathrm{L}(G)$ zwischen $(1/3)\ell$ und $(2/3)\ell$ und möglichst nahe bei $(1/2)\ell$ liegt. In F ersetzen wir die Subformel G

durch 0 und 1 und nennen diese Formeln H_0 bzw. H_1. Dann ist

$$F = ite(G, H_1, H_0) = (G \wedge H_1) \vee (\neg G \wedge H_0).$$

Das Verfahren wird jetzt rekursiv für G, H_0 und H_1 angewendet.

Pratt (1975) analysiert diese Konstruktion und berechnet für eine Konstante c $c \cdot \ell^{\log_3 10} = O\big(\ell^{2{,}096}\big)$ als obere Schranke für die Größe der konstruierten $\{\wedge, \vee, \neg\}$-Formel.

□

Die Abschätzungen in Punkt (i), (ii), (iii) und (iv) sind für sich genommen sogar optimal. Punkt (iv) zusammen mit Punkt (v) liefert uns eine erste Aussage über die Branchingprogrammgröße boolescher Funktionen bzgl. ihrer Formelgröße:

$$\mathrm{BP}(f) \leq O\big(\mathrm{L}(f)^{2{,}096}\big).$$

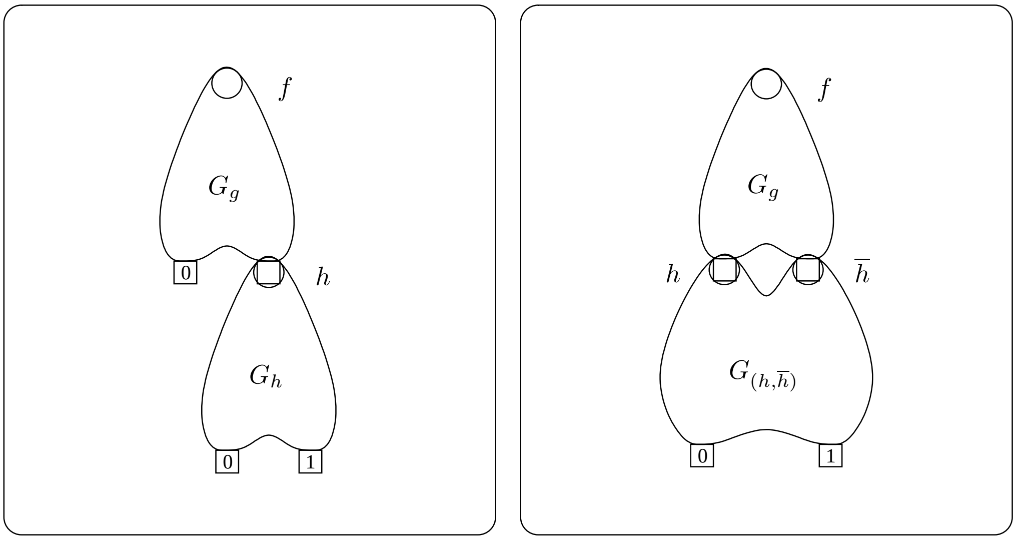
Aktuelle Resultate

Verfahren von Sauerhoff, Wegener und Werchner

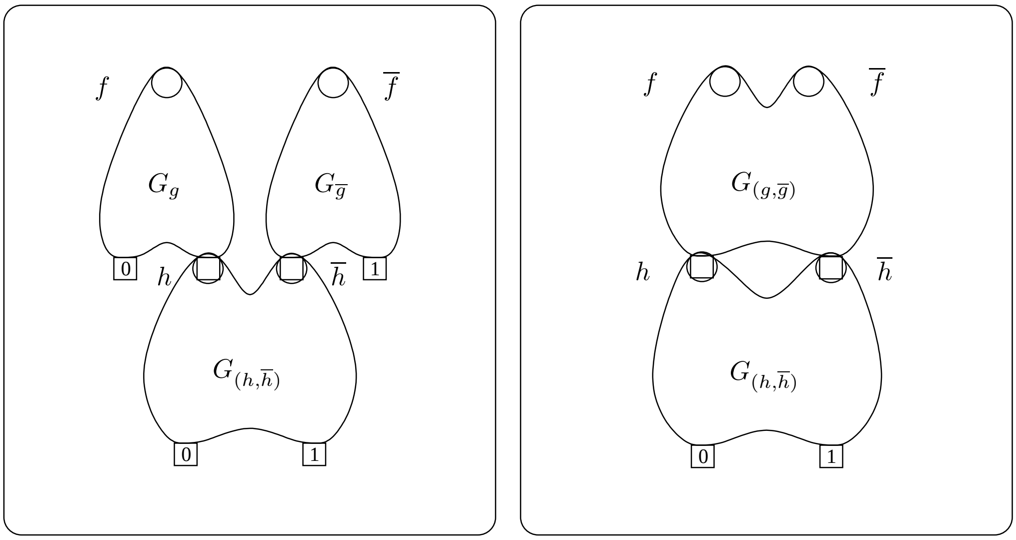
Sauerhoff, Wegener und Werchner (1999) geben ein Verfahren an, das unmittelbar aus einer gegebenen B_2-Formel ein Branchingprogramm konstruiert.

Sei F eine B_2-Formel für $f(x_1, \dots, x_n)$ und $\otimes \in B_2$ der Baustein an der Wurzel von F. Dann hat F die Gestalt $F = G \otimes H$, wo G und H Subformeln von F sind, die Funktionen g bzw. h berechnen. Wir brauchen hier nur die Fälle „$\otimes = \wedge$“ und „$\otimes = \oplus$“ zu betrachten. Alle übrigen Bausteine können mit Hilfe der deMorgan-Regeln durch einen dieser Bausteine und zusätzliche Negationsbausteine mit einem Eingang ($\neg$) ersetzt werden (vgl. Tabelle 1.1). Negationen können wir mit Branchingprogrammen auf einfache Art handhaben. Es genügt, die Markierungen der 0- und 1-Senken gegeneinander auszutauschen. Offensichtlich gilt $\mathrm{BP}(f)$=BP $(\overline{f})$.

Das Konstruktionsverfahren arbeitet rekursiv und basiert auf der folgenden Fallunterscheidung. Dabei werden, je nach Fall, Branchingprogramme für g und h oder auch für $(g, \overline{g})$ und $(h, \overline{h})$ benötigt, die rekursiv berechnet werden. Ein Branchingprogramm für $(f, \overline{f})$ ist ein Branchingprogramm mit zwei Quellen, eine für f, die andere für $\overline{f}$. Es vereint sozusagen zwei Branchingprogramme in einem. Dabei nutzen gewöhnlich beide Funktionen einen Teil der Knoten gemeinsam. Im Folgenden bezeichnen $s(f)$ und $s(f, \overline{f})$ die Größe der rekursiv konstruierten Branchingprogramme für f bzw. $(f, \overline{f})$. Die Rekursion stoppt, sobald eine Subformel F mit nur einem Blatt (Variable oder Konstante) erreicht ist. Ein minimales Branchingprogramm für das Testen einer Variablen braucht nur einen inneren Knoten, der mit der Variablen markiert ist. Branchingprogramme für Konstanten bestehen nur aus einer entsprechenden Senke und haben Größe 0. Damit gilt für Blätter der Formel $s(f) = \mathrm{BP}(f) \leq 1$ und $s(f, \overline{f}) = \mathrm{BP}(f, \overline{f}) \leq 2$.



(a) Fall 1: $f = g \wedge h$ und BP für f. (b) Fall 2: $f = g \oplus h$ und BP für f.



(c) Fall 3: $f = g \wedge h$ und BP für $(f, \overline{f})$. (d) Fall 4: $f = g \oplus h$ und BP für $(f, \overline{f})$.

Abbildung 2.1: Konstruktion der Branchingprogramme für f und $(f, \overline{f})$.

Fall 1: $\boldsymbol{f = g \wedge h}$ und BP für $\boldsymbol{f}$. Wir kombinieren Branchingprogramme G_g und G_h für g bzw. h per AND-Synthese zu einem Branchingprogramm G_f für f. Dazu ersetzen wir die 1-Senke von G_g durch die Quelle von G_h. (Siehe Abbildung 2.1(a)). Es gilt

$$s(f) = s(g) + s(h). \tag{2.1}$$

Fall 2: $\boldsymbol{f = g \oplus h}$ und BP für $\boldsymbol{f}$. Hier benötigen wir Branchingprogramme G_g für g und $G_{(h,\overline{h})}$ für $(h, \overline{h})$. Die 0-Senke von G_g wird durch die Quelle für h und die 1-Senke von G_g durch die Quelle für $\overline{h}$ in $G_{(h,\overline{h})}$ ersetzt (Abbildung 2.1(b)). Ein Berechnungspfad für eine Variablenbelegung $a = (a_1, \ldots, a_n)$, der an der Quelle von G_g startet, erreicht die 1-Senke von $G_{(h,\overline{h})}$ für $g(a) = 0$ und $h(a) = 1$ oder für $g(a) = 1$ und $\overline{h}(a) = 1 \Leftrightarrow h(a) = 0$. Das zusammengesetzte Branchingprogramm realisiert also $g \oplus h$. Da $\oplus$ kommutativ ist, vertauschen wir die Rollen von g und h, falls dies zu einem kleineren Branchingprogramm für f führt. Daher ist

$$s(f) = \min\bigl\{s(g) + s(h, \overline{h}), s(h) + s(g, \overline{g})\bigr\}. \tag{2.2}$$

Fall 3: $\boldsymbol{f = g \wedge h}$ und BP für $\boldsymbol{(f, \overline{f})}$. Wir brauchen Branchingprogramme G_g, $G_{\overline{g}}$ und $G_{(h,\overline{h})}$. Das Branchingprogramm $G_{\overline{g}}$ gewinnen wir, indem wir G_g kopieren und dann die Markierungen der Senken gegeneinander austauschen. Quelle für f wird die Quelle von G_g, dessen 1-Senke durch die Quelle für h in $G_{(h,\overline{h})}$ ersetzt wird. Für $\overline{f}$ benutzen wir die Quelle von $G_{\overline{g}}$ und ersetzen dessen 0-Senke durch die $\overline{h}$-Quelle von $G_{(h,\overline{h})}$ (Abbildung 2.1(c)). Ein Berechnungspfad für eine Variablenbelegung a, der an der Quelle für f startet, erreicht die 1-Senke von $G_{(h,\overline{h})}$ genau dann, wenn $g(a) = h(a) = 1$. Ein Berechnungspfad, der an der Quelle für $\overline{f}$ startet, erreicht eine 1-Senke, wenn $\overline{g}(a) = 1$ oder $g(a) \wedge \overline{h}(a) = 1$. An dieser Quelle wird also $\overline{g} \vee \bigl(g \wedge \overline{h}\bigr) = \overline{g} \vee \overline{h} = \overline{g \wedge h}$ berechnet. Auch hier vertauschen wir die Rollen von g und h, falls dies zu einem kleineren Ergebnis führt. Wir erhalten

$$s(f, \overline{f}) = \min\bigl\{2s(g) + s(h, \overline{h}), 2s(h) + s(g, \overline{g})\bigr\}. \tag{2.3}$$

Fall 4: $\boldsymbol{f = g \oplus h}$ und BP für $\boldsymbol{(f, \overline{f})}$. Wir benutzen Branchingprogramme $G_{(g,\overline{g})}$ und $G_{(h,\overline{h})}$. Die 0-Senke von $G_{(g,\overline{g})}$ wird durch die h-Quelle, die 1-Senke durch die $\overline{h}$-Quelle von $G_{(h,\overline{h})}$ ersetzt (Abbildung 2.1(d)). Von der Quelle für f erreicht man die 1-Senke für $g(a) = 0$ und $h(a) = 1$ oder für $g(a) = 1$ und $\overline{h}(a) = 1$. Das entspricht $\overline{g}h \vee g\overline{h} = g \oplus h$. Ein Berechnungspfad, der bei der Quelle für $\overline{f}$ startet, erreicht die 1-Senke für $\overline{g}(a) = 0$ und $h(a) = 1$ oder für $\overline{g}(a) = 1$ und $\overline{h}(a) = 1$. Die $\overline{f}$-Quelle berechnet also $gh \vee \overline{g}\overline{h} = \overline{g \oplus h}$. Als Größe erhalten wir

$$s(f, \overline{f}) = s(g, \overline{g}) + s(h, \overline{h}). \tag{2.4}$$

Am Ende eines jeden Rekursionsschritts werden noch alle Senken mit gleicher Markierung verschmolzen.

Das obige Verfahren kann so umgesetzt werden, dass nur lineare Zeit bzgl. der Größe des konstruierten Branchingprogramms benötigt wird. Das einzige Hindernis ist, dass in Fall 2 und Fall 3 die Minimumsbildung erst möglich wird, wenn die Größen $s(g)$, $s(g,\overline{g})$, $s(h)$ und $s(h,\overline{h})$ schon bekannt sind. Dazu ist es aber nicht notwendig, die zugehörigen Branchingprogramme für g, $(g,\overline{g})$, h und $(h,\overline{h})$ zuerst rekursiv zu konstruieren, um unmittelbar danach zwei von diesen vier Branchingprogrammen wieder zu verwerfen. Es genügt, die Knoten der gesamten Formel zunächst einmal in post-order Reihenfolge zu durchlaufen und für jeden Knoten $s(f)$ und $s(f,\overline{f})$ mit Hilfe der Gleichungen (2.1) bis (2.4) zu berechnen, wo f die von dem jeweiligen Knoten dargestellte Funktion ist. Dieser Durchlauf benötigt nur lineare Zeit bzgl. der Größe von F. Falls wir nur an der Größe des resultierenden Branchingprogramms für F interessiert sind, und nicht an dem Branchingprogramm selbst, sind wir an diese Stelle fertig. Ansonsten wird danach in einem zweiten rekursiven Durchlauf in linearer Zeit bzgl. der Größe des resultierenden Branchingprogramms die oben beschriebene Konstruktion durchgeführt. Offensichtlich dominiert die Laufzeit für den zweiten Durchlauf.

Mit ihrer Analyse des Verfahrens können Sauerhoff, Wegener und Werchner $\mathrm{BP}(f) \leq 1{,}360\mathrm{L}(f)^{\beta}$ mit $\beta = \log_4(3+\sqrt{5}) < 1{,}195$ als obere Schranke für die Branchingprogrammgröße boolescher Funktionen f beweisen. Anhand eines Beispiels zeigen sie, dass es eine Folge boolesche Funktionen (f_n), $f_n \in B_n$, gibt, für die das vorgestellte Konstruktionsverfahren, angewendet auf Formeln für (f_n) mit optimaler Größe $\mathrm{L}(f_n)$, Branchingprogramme mit Größe mindestens $1{,}340\mathrm{L}(f_n)^{\beta} - o(1)$ erzeugt. Obere und untere Schranke liegen also dicht beieinander, sodass die Analyse als sehr exakt anzusehen ist. Diese Schranke von Sauerhoff, Wegener und Werchner ist zur Zeit die beste, veröffentlichte obere Schranke für dieses Problem.

Ansatz von Cleve

Algebraische Formeln sind Formeln über Ringen $(\mathcal{R}, +, \cdot, 0, 1)$ und können analog zu B_2-Formeln aufgefasst werden: Die Variablen und Konstanten algebraischer Formeln können Werte aus $\mathcal{R}$ annehmen und die Menge aus beiden Ringoperationen $\{+, \cdot\}$ bildet die Basis. Cleve (1991) hat gezeigt, dass für jedes $\varepsilon > 0$ balancierte, algebraische Formeln der Größe ℓ durch sogenannte algebraische straight-line Programme konstanter Breite und Länge $O(\ell^{1+\varepsilon})$ simuliert werden können.

Wir greifen dieses Ergebnis auf und wählen als Ring $\mathbb{Z}_2 = (\{0,1\}, \oplus, \wedge, 0, 1)$. Damit lässt sich dieses Ergebnis unmittelbar auf balancierte, boolesche $\{\oplus, \wedge\}$-Formeln übertragen. Wir werden später in Kapitel 4 sehen, dass die resultierenden straight-line Programme in geschichtete Branchingprogramme derselben Länge mit ebenfalls konstanter Breite umgewandelt werden können. Für jedes $\varepsilon > 0$ kann also jede balancierte, boolesche $\{\oplus, \wedge\}$-Formel der Größe ℓ durch ein Branchingprogramm der

Größe $O(\ell^{1+\varepsilon})$ simuliert werden. Allerdings verbirgt sich bei diesem Verfahren in der O-Notation eine doppelt exponentiell in $1/\varepsilon$ wachsende multiplikative Konstante, die größer als $2^{2^{1/\varepsilon}}$ ist.

Um dieses Ergebnis auf balancierte B_2-Formeln auszuweiten, kann man versuchen, zuerst B_2-Formeln in $\{\oplus, \wedge\}$-Formeln umzuwandeln. Dabei kann sich aber leicht die Formeltiefe verdoppeln, sodass die entstehenden B_2-Formeln bzgl. der ursprünglichen Formelgröße nicht mehr balanciert sind. Somit muss dieser unmittelbare Ansatz scheitern. Glücklicherweise kann man den Umwandlungschritt so in die Konstruktion von Cleve integrieren, dass man auch für balancierte B_2-Formeln Branchingprogrammgröße $O(\ell^{1+\varepsilon})$ gewährleisten kann. Wir wollen diesen Gedanken hier nicht weiter verfolgen, sondern stattdessen in Kapitel 4 $O(\ell^{1+\varepsilon})$ als neue obere Schranke für die Branchingprogrammgröße für beliebige, d. h. insbesondere auch unbalancierte, B_2-Formeln beweisen. Dabei werden wir die zentralen Ideen der Arbeit von Cleve kennenlernen.

3 Komplexitätstheoretische Ergebnisse

Bisher haben wir gesehen, dass wir aus B_2-Formeln mit Größe ℓ Branchingprogramme konstruieren können, deren Größe durch ein kleines Polynom in ℓ beschränkt bleibt. Für eine Folge boolescher Funktion $f = (f_n)$, $f_n \in B_n$, impliziert polynomielle B_2-Formelgröße also auch polynomielle Branchingprogrammgröße. Wie sieht es aber mit der umgekehrten Richtung aus? Können wir aus Branchingprogrammen mit polynomieller Größe auch polynomiell große B_2-Formeln gewinnen, oder gibt es vielleicht Funktionenfolgen mit polynomieller Branchingprogrammgröße und exponentieller B_2-Formelgröße?

Im zweiten Teil dieses Kapitels werden wir sehen, dass wir uns bei der Suche nach optimalen Branchingprogrammen für Formeln auf read-once Formeln beschränken können.

Formelgröße für Branchingprogramme

Definition 3.1. Sei $g\colon \mathbb{N} \to \mathbb{R}$.

i) $g(n)$ wächst *polynomiell*, wenn $g(n) = O(p(n))$ für ein Polynom $p(n)$.

ii) $g(n)$ wächst *exponentiell*, wenn $g(n) = \Omega(2^{n^\varepsilon})$ für ein $\varepsilon > 0$.

Satz 2.1(ii) belegt, dass es Schaltkreise für f mit Größe $O(\mathrm{BP}(f))$ gibt. Mit Blick auf den Beweis können wir für die Schaltkreistiefe nur die triviale Schranke $O(\mathrm{BP}(f))$ angeben. Wenn wir nun die Entfaltungstechnik aus dem Beweis zu Satz 2.1(iii) auf den Schaltkreis anwenden, erhalten wir $\mathrm{L}(f) \leq 2^{O(\mathrm{BP}(f))}$. Für Funktionen mit polynomieller Branchingprogrammgröße $p(n)$ kann uns diese Abschätzung nur exponentielle Formelgröße $O(2^{p(n)})$ zusichern. Dies liegt daran, dass wir keine gute Schranke für die Schaltkreistiefe haben. Im Folgenden werden wir leider auch keine polynomielle Schranke bzgl. der Branchingprogrammgröße zeigen können. Indem wir aber die Schaltkreistiefe verbessern, können wird eine subexponentielle Schranke angeben.

Definition 3.2. Eine *nichtuniforme Turingmaschine* ist eine Turingmaschine, die neben dem Arbeitsband und einem read-only Eingabeband mit einem zusätzlichen read-only Band ausgestattet ist. Für Eingaben x mit $|x| = n$ enthält dieses Extraband ein Orakel a_n, das zwar von n, aber nicht von x selbst abhängen darf. Die Rechenzeit wird wie bei normalen Turingmaschinen gemessen. Der Speicherplatzbedarf ist die Summe aus dem Speicherplatzbedarf auf dem Arbeitsband und $\lceil \log |a_n| \rceil$.

Der folgende Satz mit Beweis ist aus der Monographie von Wegener (1987).

Satz 3.3. *Sei* $f = (f_n)$, $f_n \in B_n$, *eine Folge boolescher Funktionen und sei* $S(f_n)$ *die Speicherplatzkomplexität einer nichtuniformen Turingmaschine für* (f_n). *Dann gilt* $S(f_n) = O(\log(\max\{BP(f_n), n\}))$.

Beweis. Sei G_n ein Branchingprogramm für f_n mit Größe $\mathrm{BP}(f_n)$. Wir simulieren G_n durch eine nichtuniforme Turingmaschine, deren Orakel a_n eine Kodierung des Branchingprogramms G_n ist. Auf dem Orakelband sind die Knoten von G_n in kodierter Form gespeichert. Jeder Knoten ist durch seine Knotennummer, den Index der Variablen, mit der er markiert ist, und die Knotennummern seiner 0- und 1-Nachfolger beschrieben. Jeder Knoten braucht $O(\log \mathrm{BP}(f_n) + \log n)$ Bits auf dem Orakelband.

$$\cdots \;\Big\|\; \underbrace{\text{Knotennummer}}_{O(\log \mathrm{BP}(f_n))} \;\Big|\; \underbrace{\text{Variablenindex}}_{O(\log n)} \;\Big|\; \underbrace{\text{0-Nachfolger}}_{O(\log \mathrm{BP}(f_n))} \;\Big|\; \underbrace{\text{1-Nachfolger}}_{O(\log \mathrm{BP}(f_n))} \;\Big\|\; \cdots$$

Insgesamt wird auf dem Orakelband für a_n Platz $O(\mathrm{BP}(f_n)(\log \mathrm{BP}(f_n) + \log n))$ benötigt.

Die Turingmaschine simuliert die Branchingprogrammrechnung Schritt für Schritt. Auf dem Arbeitsband merkt sich die Turingmaschine die Kodierung des gerade erreichten Knotens des Branchingprogramms. Zu Beginn eines jeden Simulationsschritts liest die Maschine das dem Variablenindex entsprechende Bit vom Eingabeband. Nun durchsucht die Turingmaschine ihr Orakelband nach dem Knoten mit der Nummer, die gemäß dem gelesenen Eingabebit gleich dem 0- oder 1-Nachfolger auf dem Arbeitsband ist. Sobald dieser gefunden ist, kopiert sie die Kodierung des neuen Knotens auf das Arbeitsband. Falls der erreichte Knoten eine Senke ist, löscht die Maschine ihr Arbeitsband, schreibt auf Position 1 des Bandes die Markierung der Senke, 0 oder 1, und hält an.

Diese Simulation kann mit Platz $s(f_n) = O(\lceil \log \mathrm{BP}(f_n) \rceil + \lceil \log n \rceil)$ auf dem Arbeitsband auskommen. Den nichtuniformen Platzbedarf $S(f_n)$ der Turingmaschine schätzen wir folgendermaßen ab.

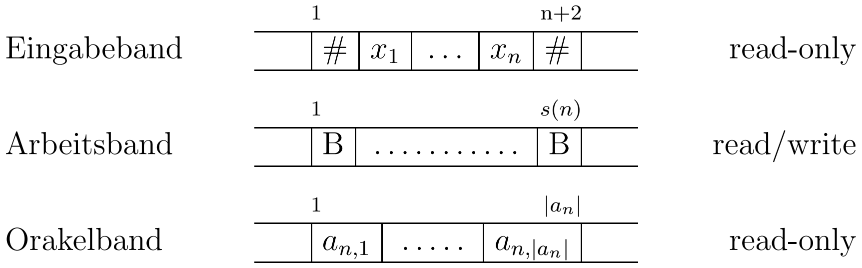
$$\begin{aligned}
S(f_n) &= s(f_n) + \lceil \log |a_n| \rceil \\
&= O\big(\log \mathrm{BP}(f_n) + \log n\big) + O\big(\log\big(\mathrm{BP}(f_n)(\log \mathrm{BP}(f_n) + \log n)\big)\big) \\
&= O\big(\log n + \log\big(\mathrm{BP}(f_n)(\log \mathrm{BP}(f_n) + \log n)\big)\big) \\
&= O\big(\log \mathrm{BP}(f_n) + \log n + \log(\log \mathrm{BP}(f_n) + \log n)\big) \\
&= O\big(\log \mathrm{BP}(f_n) + \log n\big) \\
&= O\big(\log(\max\{\mathrm{BP}(f_n), n\})\big)
\end{aligned}$$

□

Satz 3.4. *Die Folge boolescher Funktionen* $f = (f_n)$, $f_n \in B_n$, *werde von einer* $t(n)$*-zeitbeschränkten und* $S(n)$*-platzbeschränkten nichtuniformen Turingmaschine berechnet. Es sei* $\ell(n) := \max\{S(n), \lceil \log n \rceil\}$. *Dann gibt es für* $f = (f_n)$ *Schaltkreise der Tiefe* $O(\ell(n) \log t(n)) = O(\ell(n)^2)$.

In der Monographie von Wegener (1987) wird diegleiche Aussage für uniforme Turingmaschinen bewiesen. Der folgenden Beweis ist eine modifizierte Version davon.

Beweis. Die Menge der Zustände der Turingmaschine sei Q, $q_0 \in Q$ der Startzustand, das Bandalphabet sei Γ und $\mathrm{B} \in \Gamma$ sei das Leerzeichen (Blank). Die Maschine bekommt ihre Eingabe $x_1, \ldots, x_n$ eingerahmt von zwei Begrenzungszeichen $\# \in \Gamma$ auf dem read-only Eingabeband präsentiert. So kann sie durch Abzählen n ermitteln und braucht auf dem Eingabeband zu keiner Zeit Positionen außerhalb dieser $n+2$ Positionen zu besuchen.



Der Platzbedarf auf dem Arbeitsband sei für Eingaben der Länge n durch $s(n)$ beschränkt und die Länge des Orakels sei $|a_n|$, sodass $S(n) = s(n) + \lceil \log |a_n| \rceil$. Die Zahl der möglichen Konfigurationen der Turingmaschine für eine Eingabe der Länge n ist beschränkt durch

$$k(n) := \underbrace{|Q|}_{\#\text{Zustände}} \cdot \underbrace{(n+2)}_{\substack{\#\text{Kopfpos.}\\ \text{Eingabebd.}}} \cdot \underbrace{|\Gamma|^{s(n)}}_{\substack{\#\text{Inschriften}\\ \text{Arbeitsbd.}}} \cdot \underbrace{s(n)}_{\substack{\#\text{Kopfpos.}\\ \text{Arbeitsbd.}}} \cdot \underbrace{|a_n|}_{\substack{\#\text{Kopfpos.}\\ \text{Orakelbd.}}} .$$

Die Inschriften auf dem Eingabeband und dem Orakelband zählen wir nicht zur Konfiguration, da diese während der gesamten Rechnung unverändert bleiben. Aus der Definition von $k(n)$ folgt

$$\log k(n) = \log |Q| + \log(n+2) + \log |\Gamma| \cdot s(n) + \log s(n) + \log |a_n|.$$

Die Maschine muss für alle Eingaben stoppen und kann deswegen keine Konfiguration zweimal erreichen, denn sonst würde sie in einen endlosen Zyklus geraten. Daher ist $t(n) \le k(n)$ und somit $\log t(n) \le \log k(n) = O(\ell(n))$, da $|Q|$ und $|\Gamma|$ konstant sind. Also gilt auch $O(\ell(n) \log t(n)) = O(\ell(n)^2)$.

Wir modifizieren die Turingmaschine jetzt so, dass sie niemals stoppt, sondern stattdessen weiterarbeitet, dabei aber die Bandinschrift nicht mehr verändert. Wir wissen, dass die Rechnung spätestens nach $t(n)$ Schritten beendet ist. Die Rechnung für eine Eingabe $x = x_1, \ldots, x_n$ der Länge n kann also durch eine Konfigurationenfolge $k_0(x), \ldots, k_{t(n)}(x)$ beschrieben werden. Es ist $f_n(x) = 1$ genau dann, wenn die erste Zelle des Arbeitsbandes nach $t(n)$ Rechenschritten die 1 enthält.

Für jede Konfiguration $k(x)$ ist die Nachfolgekonfiguration $k'(x)$ eindeutig bestimmt. Sie hängt nicht von der gesamten Eingabe x ab, sondern nur von dem x_i, auf dem der Kopf auf dem Eingabeband steht. Wir definieren nun für jedes Konfigurationenpaar (k, k') die Funktion $a_{k,k'}(x)$. Diese soll genau dann 1 sein, wenn $k'(x)$ die direkte Nachfolgekonfiguration von $k(x)$ ist. (Die Bezeichnung $a_{k,k'}(x)$ hat nichts mit dem Orakel a_n zu tun.) Wenn in Konfiguration k der Kopf auf dem Eingabeband auf Position i steht, ist $a_{k,k'}(x)$ eine Funktion 0, 1, x_i, oder $\overline{x}_i$. Die Matrix $A(x) = (a_{k,k'}(x))$ kann also mit Schaltkreisen in Tiefe 1 berechnet werden.

Das boolesche Produkt C zweier boolescher Matrizen A und B ist durch

$$c_{i,j} = \bigvee_k (a_{i,k} \wedge b_{k,j})$$

definiert. $A^t(x)$ bezeichnet das Produkt von t Kopien von $A(x)$. Da

$$a^t_{k,k'}(x) = \bigvee_{k''} \big(a^{t-1}_{k,k''}(x) \wedge a_{k'',k'}(x)\big),$$

folgt, dass $a^t_{k,k'}(x) = 1$ genau dann, wenn bei Eingabe x die Konfiguration k' die t-te Nachfolgekonfiguration von k ist. Sei $T = 2^{\lceil \log t(n) \rceil} \geq t(n)$. Die Matrix $A^T(x)$ lässt sich durch fortgesetztes Quadrieren mit $\lceil \log t(n) \rceil$ booleschen Matrizenmultiplikationen berechnen. Jede Matrizenmultiplikation können wir mit einem Schaltkreis mit Tiefe $\lceil \log k(n) \rceil + 1$ berechnen: Alle Konjunktionen benötigen Tiefe 1 und die Disjunktion realisieren wir mit einem balancierten Baum aus $\vee$-Bausteinen in Tiefe $\lceil \log k(n) \rceil$. Die Tiefe für alle Matrizenmultiplikationen beträgt also $\lceil \log t(n) \rceil (\lceil \log k(n) \rceil + 1)$.

Die Turingmaschine startet in einer eindeutig bestimmten Anfangskonfiguration: Zustand q_0, alle Köpfe der Maschine stehen auf Position 1 und die Inschrift auf dem Arbeitsband besteht nur aus Leerzeichen. Wenn k_0 die Anfangskonfiguration und K^* die Menge aller akzeptierenden Konfigurationen ist, gilt

$$f_n(x) = \bigvee_{k \in K^*} a^T_{k_0,k}(x). \tag{3.1}$$

Die Gesamttiefe des Schaltkreises setzt sich zusammen aus der Tiefe zur Berechnung der Matrix $A(x)$, der Matrizenmultiplikationen und der Disjunktion in Gleichung (3.1), die wir wieder mit Tiefe $\lceil \log k(n) \rceil$ realisieren können:

$$\begin{aligned} 1 + \lceil \log t(n) \rceil \big(\lceil \log k(n) \rceil + 1\big) + \lceil \log k(n) \rceil &= O\big(\log t(n) \log k(n)\big) \\ &= O\big(\ell(n) \log t(n)\big). \end{aligned}$$

□

Satz 3.5. *Jede Folge boolescher Funktionen* $f = (f_n)$*,* $f_n \in B_n$*, die essentiell von allen ihren Variablen abhängen, kann durch eine Folge* $F = (F_n)$ *von* B_2*-Formeln mit Größe* $L(F_n) = BP(f_n)^{O(\log BP(f_n))}$ *berechnet werden.*

Beweis. Boolesche Funktionen, die essentiell von allen ihren Variablen $x_1, \ldots, x_n$ abhängen, können nur von Branchingprogrammen berechnet werden, die für jede Variable mindestens einen Knoten besitzen, der diese Variable testet. Deswegen gilt $\mathrm{BP}(f_n) \geq n$ und wir erhalten mit Satz 3.3 eine nichtuniforme Turingmaschine für (f_n) mit $S(f_n) = O(\log \mathrm{BP}(f_n))$. Aus dem Beweis wissen wir, dass in diesem Fall $S(f_n) \geq \lceil \log \mathrm{BP}(f_n) \rceil$ gilt, und somit $\lceil \log \mathrm{BP}(f_n) \rceil \geq \lceil \log n \rceil$. Diese Maschine kann mit Satz 3.4 wegen $\ell(n) = \max\{S(f_n), \lceil \log n \rceil\} = O(\log \mathrm{BP}(f_n))$ durch Schaltkreise mit Tiefe $O(\ell(n)^2) = O(\log^2 \mathrm{BP}(f_n))$ simuliert werden. Mit der Entfaltungstechnik aus dem Beweis zu Satz 2.1(iii) wandeln wir diese Schaltkreise in B_2-Formeln um. Die Formelgrößen sind durch

$$\mathrm{L}(F_n) \leq 2^{O(\log^2 \mathrm{BP}(f_n))} \leq 2^{c \log^2 \mathrm{BP}(f_n)} = \mathrm{BP}(f_n)^{c \log \mathrm{BP}(f_n)} = \mathrm{BP}(f_n)^{O(\log \mathrm{BP}(f_n))}$$

für eine gewisse Konstante c beschränkt. □

Falls die Branchingprogrammgröße für (f_n) durch ein Polynom $p(n)$ beschränkt ist, erhalten wir B_2-Formeln mit Größe

$$\mathrm{L}(F_n) \leq p(n)^{c \log p(n)}.$$

Das Polynom $p(n)$ schätzen wir mit einem hinreichend großen, aber konstanten $k \in \mathbb{N}$ durch n^k ab:

$$\mathrm{L}(F_n) \leq \left(n^k\right)^{c \log n^k} = \left(n^k\right)^{ck \log n} = n^{ck^2 \log n} = n^{O(\log n)}.$$

Es ist bekannt, dass für alle $\varepsilon > 0$ $n^{O(\log n)} = o(2^{n^\varepsilon})$ gilt. Daraus folgt, für alle $\varepsilon > 0$ gilt

$$\mathrm{L}(F_n) = o\big(2^{n^\varepsilon}\big),$$

d. h. $\mathrm{L}(F_n)$ wächst langsamer als exponentiell.

Korollar 3.6. *Jede Folge boolescher Funktionen* $f = (f_n)$*,* $f_n \in B_n$*, mit polynomieller Branchingprogrammgröße* $BP(f_n) = O(p(n))$ *wird von einer Folge von* B_2*-Formeln* $F = (F_n)$ *mit subexponentieller Größe* $L(F_n) = BP(f)^{O(\log BP(f))}$ *berechnet.*

Read-once Formeln und Branchingprogrammgröße

Es ist ein offenes Problem, die maximal notwendige Branchingprogrammgröße boolescher Funktionen, die durch B_2-Formeln der Größe $\mathrm{L}(f)$ dargestellt werden können, zu bestimmen. Wir kennen aus Kapitel 2 die obere Schranke $\mathrm{BP}(f) = O(\mathrm{L}(f)^{1{,}195})$. Welche untere Schranke kann man angeben? Funktionen $f \in B_n$, die essentiell von allen ihren Variablen abhängen und für die es read-once Formeln mit Größe $\mathrm{L}(f) = n$ gibt, können nur Branchingprogramme mit Größe $\mathrm{BP}(f) \geq \mathrm{L}(f)$ haben, da es für jede Variabel mindestens einen Knoten geben muss. Die Parity-Funktion für n Variablen ist so eine Funktion, da sie offensichtlich als read-once B_2-Formel mit n Blättern dargestellt werden kann und essentiell von allen Variablen abhängt. Daraus folgt $\mathrm{BP}(f) = \Omega(\mathrm{L}(f))$. Der folgende Satz besagt, dass man sich auf read-once Formeln beschränken kann, um die maximal notwendige Branchingprogrammgröße weiter einzugrenzen.

Satz 3.7 (Sauerhoff, Wegener und Werchner (1999)). *Sei $L(\ell)$ die Klasse der booleschen Funktionen, deren B_2-Formelgröße genau ℓ ist, und sei $BP(\ell) \subseteq L(\ell)$ die Klasse aller Funktionen mit maximaler Branchingprogrammgröße in $L(\ell)$. Dann gibt es in $BP(\ell)$ eine Funktion f, die durch eine read-once B_2-Formel der Größe ℓ dargestellt werden kann.*

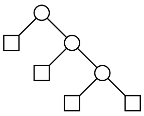
Beweis. Sei $f \in \mathrm{BP}(\ell)$. Wir betrachten den Graphen einer optimalen Formel für f. Wir wollen die Fan-out-1-Eigenschaft auf die Variablen und Konstanten ausweiten. Dazu duplizieren wie ggf. die Variablen und Konstanten, sodass der Graph zu einem Baum wird, dessen Blätter Variablen und Konstanten sind. Anschließend ersetzen wir die ℓ Blätter durch neue Variablen $x_1, \ldots, x_\ell$. Der Graph repräsentiert jetzt eine read-once Formel für eine Funktion $f^* \in \mathrm{L}(\ell)$. Es bleibt noch $f^* \in \mathrm{BP}(\ell)$ zu zeigen. Dazu genügt es $\mathrm{BP}(f^*) \geq \mathrm{BP}(f)$ zu beweisen. Sei G ein optimales Branchingprogramm für f^* mit Größe $\mathrm{BP}(f^*)$. Wir machen die Ersetzung der Variablen und Konstanten durch $x_1, \ldots, x_\ell$ rückgängig und erhalten ein Branchingprogramm für f mit Größe $\mathrm{BP}(f^*)$. Daraus folgt $\mathrm{BP}(f) \leq \mathrm{BP}(f^*)$. □

Häufig ist es bequemer, nur read-once Formeln zu betrachten, weil deren zugrundeliegender Graph ein Baum ist. In Kapitel 4 werden wir davon Gebrauch machen und einige Resultate nur für read-once Formeln beweisen. Mit der Methode im Beweis von Satz 3.7 sind wir aber meistens in der Lage solche Resultate auch auf uneingeschränkte Formeln zu übertragen.

4 Eine neue obere Schranke: $\mathbf{BP}(f) = O(\mathbf{L}(f)^{1+\varepsilon})$

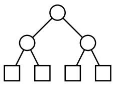
Wir werden in diesem Kapitel für alle $\varepsilon > 0$ $\mathrm{BP}(f) = O(\mathrm{L}(f)^{1+\varepsilon})$ als neue obere Schranke für die Branchingprogrammgröße boolescher Funktionen f beweisen. Vereinfacht gesagt setzt sich der Beweis aus einer Kette von Simulationen eines Berechnungsmodells durch das jeweils folgende zusammen, an deren Anfang beliebige B_2-Formeln und an deren Ende (geschichtete) Branchingprogramme stehen.

B_2-Formel



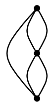
↓

balancierte $\{\oplus, \wedge\}$-Formel



↓

Parity-Branchingprogramm



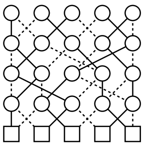
↓

linear bijection straight-line Program über $\mathbb{Z}_2$

$$R_j \leftarrow R_j \oplus (R_i \wedge 0)$$
$$R_j \leftarrow R_j \oplus (R_i \wedge 1)$$
$$R_j \leftarrow R_j \oplus (R_i \wedge x_u)$$

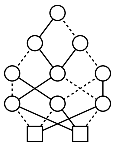
↓

Permutations-Branchingprogramm



↓

(geschichtetes) Branchingprogramm



Die Berechnungsmodelle Parity-Branchingprogramm, linear bijection straight-line Programm und Permutations-Branchingprogramm werden wir im Laufe dieses Kapitels kennenlernen. Der hier dargestellte Beweis benutzt die Arbeit von Bonet und Buss (1994) und basiert auf Ideen vor allem von Cleve (1991). Die entsprechenden Stellen sind mit den Autorennamen markiert.

Formeln balancieren

Das Ziel des ersten Schritts der Konstruktion ist es, beliebige B_2-Formeln der Größe ℓ so in äquivalente $\{\oplus, \wedge\}$-Formeln umzuwandeln, dass deren Tiefe durch $O(\log \ell)$ beschränkt bleibt. „Balancieren“ ist hier nicht im strengen Sinne von Definition 1.7 mit Tiefe $\lceil \log \ell \rceil$ zu verstehen, sondern es genügt uns Tiefe $O(\log \ell)$.

Lemma 4.1 (Brent (1974)). *Sei F eine Formel über einer Basis $\Omega \subseteq B_2$ und sei $2 \leq s \leq L(F)$. Dann gibt es in F eine Subformel G mit Größe mindestens s, sodass die unmittelbaren Subformeln von G, G_L und G_R, echt kleiner als s sind.*

Beweis. Jede minimale Subformel von F mit Größe mindestens s erfüllt die Bedingungen für G. Man findet eine solche Subformel G als Endpunkt eines Pfades, der an der Wurzel von F startet und sich solange in Richtung einer größten Subformeln wendet, wie diese mindestens Größe s hat. ☐

Der folgende Satz mit Beweis ist eine ausgearbeitete Darstellung wesentlicher Teile aus der Arbeit von Bonet und Buss (1994).

Satz 4.2 (Bonet und Buss (1994)). *Sei F eine $\{\oplus, \wedge\}$-Formel mit Größe ℓ. Dann gibt es für alle $c \geq 2$ eine äquivalente $\{\oplus, \wedge\}$-Formel F' mit*

$$\begin{aligned} &D(F') \leq 3c \cdot \ln 2 \cdot \log \ell \quad \textit{und} \\ &L(F') \leq \ell^{\alpha}, \quad \alpha = 1 + \frac{1}{1 + \log(c-1)}. \end{aligned}$$

Beweis. Wir führen einen Induktionsbeweis über die Formelgröße von F. Im Induktionsanfang sei zunächst $\ell = 1$. F ist eine Konstante, 0 oder 1, oder eine Variable x. F selbst hat die gewünschte Formeltiefe $d = 0 \leq 3c \cdot \ln 2 \cdot \log 1$. Eine Formel F mit Größe $l = 2$ hat genau einen Baustein mit zwei Eingängen, die mit Variablen oder Konstanten belegt sind. Auch hier hat F selbst die gewünschte Tiefe $d = 1 \leq 3c \cdot \ln 2 \cdot \log 2$. Wir wählen $F' = F$.

Für den Induktionsschritt sei $\ell \geq 3$ und wir dürfen annehmen, dass Formeln bis zur Größe $\ell - 1$ wie behauptet balanciert werden können. Mit Lemma 4.1 wählen wir eine Subformel G mit Größe mindestens $\frac{c-1}{c}\ell$, deren direkte Subformeln G_{L} und G_{R} echt kleiner als $\frac{c-1}{c}\ell$ sind. Da $\ell \geq 3$ und $c \geq 2$, ist die Größe von G mindestens 2 und G hat mindestens einen Baustein. Der Baustein an der Wurzel von G sei $\otimes \in \{\oplus, \wedge\}$, d. h. $G = G_{\mathrm{L}} \otimes G_{\mathrm{R}}$. Im Folgenden bezeichnen F_0, F_1 und F_x die Formel, die man erhält, wenn man in F die Subformel G durch 0, 1 bzw. eine neue Variable x ersetzt. Die von F dargestellte Funktion lässt sich nun auf die folgende Art als Formel schreiben.

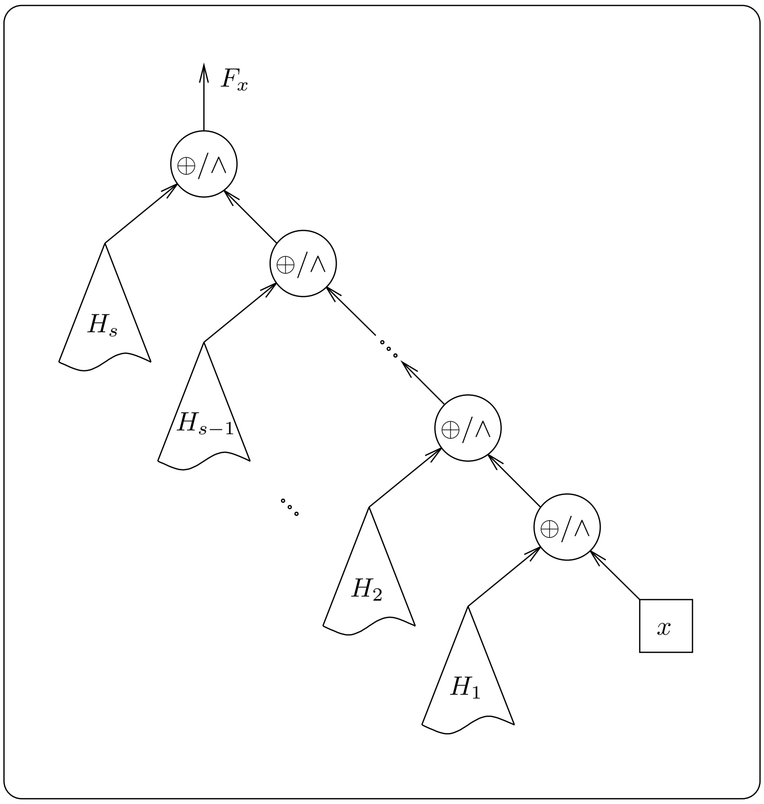


Abbildung 4.1: Pfad von x zur Wurzel der Formel F_x.

Behauptung 4.3. $F \equiv \big(G \wedge (F_0 \oplus F_1)\big) \oplus F_0$.

Beweis. Sei zunächst der Wert von $G = 0$. Dann haben F und F_0 denselben Wert. In obigem Ausdruck fällt der geklammerte Term weg. Wenn andererseits der Wert von $G = 1$ ist, haben F und F_1 denselben Wert. Obiger Ausdruck wird zu $F_0 \oplus F_1 \oplus F_0 \equiv F_1 \equiv F$. □

Wir betrachten nun in der Formel F_x den eindeutig bestimmten Pfad von x zur Wurzel. Es sei $\{H_1, \ldots, H_s\}$ die Menge der Subformeln, die Eingänge für Bausteine auf diesem Pfad sind und nicht x als Eingang haben. Siehe Abbildung 4.1. Die Teilmenge $\{H_{i_1}, \ldots, H_{i_r}\} \subseteq \{H_1, \ldots, H_s\}$ sei die Menge der Subformeln, die Eingänge für $\wedge$-Bausteine auf diesem Pfad sind.

Behauptung 4.4. $F_0 \oplus F_1 \equiv H_{i_1} \wedge \cdots \wedge H_{i_r}$.

Beweis. Falls $r = 0$, gibt es keine Konjunktion auf dem Pfad zu x, sondern nur $\oplus$-Bausteine. Der Wert von F_x hängt also von x ab. F_0 und F_1 berechnen für jede Belegung der Eingänge verschiedene Werte. Es folgt $F_0 \oplus F_1 \equiv 1$. Die „leere Konjunktion“ für $r = 0$ hat als Produkt ebenfalls den Wert 1.

Sei nun $r > 0$. Angenommen es gibt ein j, sodass $H_{i_j} = 0$. Dann ist der Wert der zugehörigen Konjunktionen in F_0 und F_1 identisch, nämlich 0. F_0 und F_1 müssen daher denselben Wert berechnen. Es folgt $F_0 \oplus F_1 \equiv 0$. Wenn hingegen alle H_{i_j} eine 1 berechnen, hängt der Wert jeder Konjunktion auf dem Pfad von derjenigen Subformel ab, die x beinhaltet. F_0 und F_1 berechnen verschiedene Werte. Somit folgt $F_0 \oplus F_1 \equiv 1$. □

Mit $H = H_{i_1} \wedge \cdots \wedge H_{i_r}$ können wir die von F dargestellte Funktion nun wie folgt schreiben:

$$F \equiv (G \wedge H) \oplus F_0.$$

Behauptung 4.5. $L(H) \leq \frac{\ell}{c}$.

Beweis. Es ist

$$\mathrm{L}(H) = \sum_{1 \leq j \leq r} \mathrm{L}(H_{i_j}) \leq \sum_{1 \leq i \leq s} \mathrm{L}(H_i) < \left(\sum_{1 \leq i \leq s} \mathrm{L}(H_i) \right) + 1 = \mathrm{L}(F_0),$$

also $\mathrm{L}(H) < \mathrm{L}(F_0)$. Dies ist wegen der Ganzzahligkeit der Formelgrößen äquivalent zu $\mathrm{L}(H) \leq \mathrm{L}(F_0) - 1$. Daraus folgt die Behauptung mit

$$\mathrm{L}(F_0) \leq \underbrace{\ell}_{\mathrm{L}(F)} - \underbrace{\frac{c-1}{c}\ell}_{\leq \mathrm{L}(G)} + \underbrace{1}_{\mathrm{L}(0)} = \frac{\ell}{c} + 1.$$

□

Mit der Größe von F_0 sind wir noch nicht zufrieden. Wir ersetzen F_0 durch eine äquivalente Formel F_0^*. Dazu eliminieren wir in F_0 den Baustein, an dem die 0 hängt, auf die folgende Art und Weise. Der oben beschriebene Pfad in F_x existiert auch in F_0, mit dem Unterschied, dass der Eingang x durch eine 0 ersetzt ist (Abbildung 4.1). Falls der unterste Baustein auf dem Pfad ein $\oplus$-Baustein ist, ersetzten wir $H_1 \oplus 0$ durch H_1, wenn es sich um einen $\wedge$-Baustein handelt, ersetzen wir $H_1 \wedge 0$ durch 0. Die modifizierte Formel F_0^* ist um mindestens 1 kleiner als F_0 und hat daher höchstens Größe $\frac{\ell}{c}$. Nun können wir die Formel noch einmal umschreiben:

$$F \equiv \Big(\big(\underbrace{G_{\mathrm{L}}}_{<\frac{c-1}{c}\ell} \otimes \underbrace{G_{\mathrm{R}}}_{<\frac{c-1}{c}\ell} \big) \wedge \underbrace{H}_{\leq \frac{\ell}{c}} \Big) \oplus \underbrace{F_0^*}_{\leq \frac{\ell}{c}}.$$

Wir benutzen jetzt die Induktionsvoraussetzung für G_{L}, G_{R}, H und F_0^* und erhalten äquivalente Formeln G'_{L}, G'_{R}, H' bzw. $F_0^{*\prime}$ mit folgenden Eigenschaften:

$$\mathrm{L}(G'_{\mathrm{L}}),\ \mathrm{L}(G'_{\mathrm{R}}) < \left(\frac{c-1}{c}\ell \right)^{\alpha}, \qquad \mathrm{L}(H'),\ \mathrm{L}(F'_0) \leq \left(\frac{\ell}{c} \right)^{\alpha},$$

$$\mathrm{D}(G'_{\mathrm{L}}),\ \mathrm{D}(G'_{\mathrm{R}}) < 3c \cdot \ln 2 \cdot \log\left(\frac{c-1}{c}\ell \right), \qquad \mathrm{D}(H'),\ \mathrm{D}(F'_0) \leq 3c \cdot \ln 2 \cdot \log \frac{\ell}{c}.$$

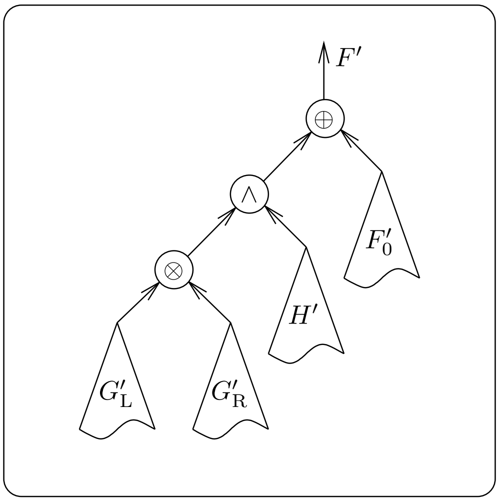


Abbildung 4.2: Formel F'.

Ersetzen liefert

$$F \equiv \big((G'_{\mathrm{L}} \otimes G'_{\mathrm{R}}) \wedge H'\big) \oplus F_0^{*\prime} = F'.$$

Es ist zu zeigen, dass F' (vgl. Abbildung 4.2) die behauptete Tiefe und Größe nicht überschreitet.

$$\begin{aligned}
\mathrm{D}(F') &= \max\{\mathrm{D}(G'_{\mathrm{L}})+3, \mathrm{D}(G'_{\mathrm{R}})+3, \mathrm{D}(H')+2, \mathrm{D}(F'_0)+1\} \\
&< 3c \cdot \ln 2 \cdot \log\left(\frac{c-1}{c}\ell\right) + 3 \\
&= 3c \cdot \ln 2 \cdot \log \ell + 3c \cdot \ln 2 \cdot \log\left(\frac{c-1}{c}\right) + 3 \\
&= 3c \cdot \ln 2 \cdot \log \ell + 3c \cdot \ln 2 \cdot \log\left(1 - \frac{1}{c}\right) + 3 \\
&= 3c \cdot \ln 2 \cdot \log \ell + 3c \cdot \ln\left(1 - \frac{1}{c}\right) + 3 \qquad \Big| \ln\left(1-\frac{1}{c}\right) < -\frac{1}{c} \\
&< 3c \cdot \ln 2 \cdot \log \ell - 3c \cdot \frac{1}{c} + 3 \\
&= 3c \cdot \ln 2 \cdot \log \ell
\end{aligned}$$

Damit ist die behauptete Tiefe gezeigt. Für die Größe gilt

$$\begin{aligned}
\mathrm{L}(F') &= \mathrm{L}(G'_\mathrm{L}) + \mathrm{L}(G'_\mathrm{R}) + \mathrm{L}(H') + \mathrm{L}(F_0^{*\prime}) \\
&\le \mathrm{L}(G_\mathrm{L})^\alpha + \mathrm{L}(G_\mathrm{R})^\alpha + \mathrm{L}(H)^\alpha + \mathrm{L}(F_0^*)^\alpha \qquad \Big| \; \mathrm{L}(H),\ \mathrm{L}(F_0^*) \le \ell - \mathrm{L}(G) \\
&\le \underbrace{\mathrm{L}(G_\mathrm{L})}_{=:g_\mathrm{L}}{}^\alpha + \underbrace{\mathrm{L}(G_\mathrm{R})}_{=:g_\mathrm{R}}{}^\alpha + 2\big(\ell - \underbrace{\mathrm{L}(G)}_{=:g}\big)^\alpha. \qquad (4.1)
\end{aligned}$$

Wenn wir – nur für einen Moment – g als Konstante betrachten, dann können wir mit den neuen Bezeichnern die rechte Seite von Ungleichung (4.1) als Funktion von g_L auffassen.

$$\mathrm{L}(F') \le \mathrm{L}(F')(g_\mathrm{L}) := g_\mathrm{L}^\alpha + (g - g_\mathrm{L})^\alpha + 2(\ell - g)^\alpha$$

Dabei haben wir $g_\mathrm{R} = g - g_\mathrm{L}$ benutzt. Es ist $1 \le g_\mathrm{L} < \frac{c-1}{c}\ell$. Die Funktion $\mathrm{L}(F')(g_\mathrm{L})$ sei für uns auf dem Intervall $\big[1, \lceil \frac{c-1}{c}\ell \rceil - 1\big]$ definiert, in dem wir eine obere Schranke für das Maximum suchen. Die erste Ableitung

$$\alpha g_\mathrm{L}^{\alpha-1} - \alpha(g - g_\mathrm{L})^{\alpha-1}$$

ist auf unserem Intervall monoton wachsend in g_L, da $\alpha > 1$ und $g > g_\mathrm{L}$. Daraus folgt, dass die Funktion $\mathrm{L}(F')(g_\mathrm{L})$ auf $\big[1, \lceil \frac{c-1}{c}\ell \rceil - 1\big]$ konvex von unten ist und ein Maximum auf dem Rand des Intervalls liegt. Im linken Randpunkt, $g_\mathrm{L} = 1$, gilt zusätzlich $g_\mathrm{R} = g - g_\mathrm{L} = \lceil \frac{c-1}{c}\ell \rceil - 1$, weil G mit Größe mindestens $\frac{c-1}{c}\ell$ gewählt wurde. Wir setzen die Randpunkte in Ungleichung (4.1) ein.

links: $g_\mathrm{L} = 1,\ g_\mathrm{R} = \lceil \frac{c-1}{c}\ell \rceil - 1$

$$\mathrm{L}(F') \le 1^\alpha + \left(\left\lceil \frac{c-1}{c}\ell \right\rceil - 1\right)^\alpha + 2(\ell - g)^\alpha$$

rechts: $g_\mathrm{L} = \lceil \frac{c-1}{c}\ell \rceil - 1,\ g_\mathrm{R} = g - \big(\lceil \frac{c-1}{c}\ell \rceil - 1\big)$

$$\mathrm{L}(F') \le \left(\left\lceil \frac{c-1}{c}\ell \right\rceil - 1\right)^\alpha + \left(g - \left(\left\lceil \frac{c-1}{c}\ell \right\rceil - 1\right)\right)^\alpha + 2(\ell - g)^\alpha$$

Da am rechten Rand wegen $g - g_\mathrm{L} \ge 1$

$$1^\alpha \le (g - g_\mathrm{L})^\alpha = \left(g - \left(\left\lceil \frac{c-1}{c}\ell \right\rceil - 1\right)\right)^\alpha$$

gilt, liefert die rechte Seite des Intervalls eine obere Schranke für das Maximum. Diese kann durch

$$\mathrm{L}(F') \le \mathrm{L}(F')(g) := \left(\frac{c-1}{c}\ell\right)^\alpha + \left(g - \frac{c-1}{c}\ell\right)^\alpha + 2(\ell - g)^\alpha \qquad (4.2)$$

nach oben abgeschätzt werden. Diese Funktion hängt von g ab – wir betrachten g ab jetzt wieder als Variable. Ihre erste Ableitung

$$\alpha\left(g-\frac{c-1}{c}\ell\right)^{\alpha-1}-2\alpha(\ell-g)^{\alpha-1}$$

wächst auf dem Intervall $\left[\frac{c-1}{c}\ell,\ell\right]$ monoton in g. Somit ist $\mathrm{L}(F')(g)$ über diesem Intervall konvex von unten und nimmt auch hier ein Maximum auf dem Rand des Intervalls ein. Wir rechnen nach, dass das Maximum am linken Rand, $g=\frac{c-1}{c}\ell$, liegt:

$$\begin{aligned}\mathrm{L}(F')\left(\frac{c-1}{c}\ell\right)&=\left(\frac{c-1}{c}\ell\right)^{\alpha}+\left(\frac{c-1}{c}\ell-\frac{c-1}{c}\ell\right)^{\alpha}+2\left(\ell-\frac{c-1}{c}\ell\right)^{\alpha}\\&=\left(\frac{c-1}{c}\ell\right)^{\alpha}+2\left(\ell-\frac{c-1}{c}\ell\right)^{\alpha}\\&>\left(\frac{c-1}{c}\ell\right)^{\alpha}+\left(\ell-\frac{c-1}{c}\ell\right)^{\alpha}\\&=\left(\frac{c-1}{c}\ell\right)^{\alpha}+\left(\ell-\frac{c-1}{c}\ell\right)^{\alpha}+2(\ell-\ell)^{\alpha}\\&=\mathrm{L}(F')(\ell).\end{aligned}$$

Mit Ungleichung (4.2) erhalten wir

$$\mathrm{L}(F')\leq\left(\frac{c-1}{c}\ell\right)^{\alpha}+2\left(\frac{\ell}{c}\right)^{\alpha}.$$

Es bleibt noch zu zeigen, dass für $\alpha=1+\frac{1}{1+\log(c-1)}$ gilt.

$$\left(\frac{c-1}{c}\ell\right)^{\alpha}+2\left(\frac{\ell}{c}\right)^{\alpha}\leq\ell^{\alpha}.$$

Dieses ist wegen $\ell^{\alpha}>0$ äquivalent zu

$$\left(\frac{c-1}{c}\right)^{\alpha}+2\left(\frac{1}{c}\right)^{\alpha}\leq 1.$$

Die linke Seite dieser Ungleichung fällt monoton für wachsendes α. Sei α_0 das eindeutig bestimmte $\alpha>1$, bei dem die linke Seite zu 1 wird.

$$\left(\frac{c-1}{c}\right)^{\alpha_0}+2\left(\frac{1}{c}\right)^{\alpha_0}=1 \tag{4.3}$$

Es genügt jetzt zu zeigen, dass $\alpha_0 \leq 1 + \frac{1}{1+\log(c-1)}$. Dazu multiplizieren wir die α_0 definierende Gleichung (4.3) mit c^{α_0} und erhalten

$$c^{\alpha_0} - (c-1)^{\alpha_0} = 2. \tag{4.4}$$

Die linke Seite wächst mit wachsendem α_0. Weil $c^2 - (c-1)^2 = 2c - 1$ wegen $c \geq 2$ größer als 2 ist, muss α_0 echt kleiner als 2 sein. Wir betrachten jetzt die Funktion $f(x) = x^{\alpha_0}$, die stetig in $[c-1, c]$ und differenzierbar in $]c-1, c[$ ist. Mit dem Mittelwertsatz der Differentialrechnung folgt:

$$\exists\, x_0 \in\,]c-1, c[\colon f'(x_0) = \frac{f(c) - f(c-1)}{c - (c-1)} = c^{\alpha_0} - (c-1)^{\alpha_0} \overset{(4.4)}{=} 2,$$

also

$$f'(x_0) = \alpha_0 x_0^{\alpha_0 - 1} = 2.$$

Da $f'(x)$ monoton wächst und $c - 1 < x_0$, ist $f'(c-1) = \alpha_0 (c-1)^{\alpha_0 - 1} < 2$. Logarithmieren ergibt

$$\begin{aligned} \log\big(\alpha_0 (c-1)^{\alpha_0 - 1}\big) &< \log 2 \\ \log \alpha_0 + \log\big((c-1)^{\alpha_0 - 1}\big) &< 1 \\ \log \alpha_0 + (\alpha_0 - 1)\log(c-1) &< 1. \end{aligned} \tag{4.5}$$

Wegen $\alpha_0 \in\,]1, 2[$, ist $\log \alpha_0 > \alpha_0 - 1$. Dies in Ungleichung (4.5) eingesetzt liefert schlussendlich das gewünschte Resultat:

$$\begin{aligned} (\alpha_0 - 1) + (\alpha_0 - 1)\log(c-1) &< 1 \\ (\alpha_0 - 1)\big(1 + \log(c-1)\big) &< 1 \\ \alpha_0 - 1 &< \frac{1}{1 + \log(c-1)} \\ \alpha_0 &< 1 + \frac{1}{1 + \log(c-1)}. \end{aligned}$$

□

Lemma 4.6. *Jede B_2-Formel F mit Größe ℓ kann durch eine äquivalente $\{\oplus, \wedge\}$-Formel F' mit Größe $\ell' < 3\ell$ ersetzt werden.*

Beweis. Wir ersetzen jeden einzelnen der $\ell - 1$ B_2-Bausteine von F durch höchstens einen $\oplus$- oder $\wedge$-Baustein und höchstens einen $\neg$-Baustein (vgl. Tabelle 1.1). Benachbarte $\neg$-Bausteine heben sich in ihrer Wirkung gerade auf und werden eliminiert. In der Formel gibt es jetzt höchstens $\ell - 1$ $\oplus$- und $\wedge$-Bausteine und höchstens auf jeder der $2(\ell - 1)$ Kanten der ursprünglichen Formel einen $\neg$-Baustein. Möglicherweise gibt es noch einen zusätzlichen $\neg$-Baustein an der Wurzel der Formel. Insgesamt gibt es also höchstens $2(\ell - 1) + 1 = 2\ell - 1$ $\neg$-Bausteine. Jeden der $\neg$-Bausteine ersetzen wir nun durch einen $\oplus$-Baustein, dessen zweiter Eingang mit 1 belegt wird ($\overline{a} = a \oplus 1$). Bei der letzten Ersetzung wächst die Formelgröße um die Zahl der eingefügten Einsen, also höchstens um $2\ell - 1$. □

Das folgende Korollar folgt unmittelbar mit Lemma 4.6 aus Satz 4.2.

Korollar 4.7. *Sei F eine B_2-Formel mit Größe ℓ. Dann gibt es für alle $c \geq 2$ eine äquivalente $\{\oplus, \wedge\}$-Formel F' mit*

$$D(F') \leq 3c \cdot \ln 2 \cdot \log(3\ell) \quad \textit{und}$$
$$L(F') \leq (3\ell)^{\alpha}, \quad \alpha = 1 + \frac{1}{1 + \log(c-1)}.$$

Formeln in straight-line Programme transformieren

Um aus Formeln straight-line Programme zu konstruieren, transformieren wir Formeln zunächst in Parity-Branchingprogramme. Karchmer und Wigderson (1993) definieren ein etwas anderes Branchingprogramm-Modell mit verschiedenen Akzeptanzmodi. Parity-Branchingprogramme sind in der Sichtweise von Karchmer und Wigderson deterministische Branchingprogramme mit Akzeptanzmodus „1 (mod 2)“.

Definition 4.8. Ein *Parity-Branchingprogramm* ($\oplus$-BP) besteht aus einem gerichteten, azyklischen Graphen $G = (V, E)$ mit zwei ausgezeichneten Knoten $s, t \in V$ und einer Markierung $\mu\colon E \to \{x_1, \ldots, x_n\} \cup \{1\}$. Für eine Eingabe $a \in \{0,1\}^n$ sei $G_a(V, E_a)$ der unmarkierte Subgraph von G, sodass $e \in E_a$ genau dann, wenn entweder $\mu(e) = 1$, oder $\mu(e) = x_u$ und $a_u = 1$. Sei p_a die Anzahl der Pfade in G_a, die von der Quelle s zur Senke t führen. Das $\oplus$-BP akzeptiert eine Eingabe a, wenn $p_a \equiv 1 \bmod 2$. Ein $\oplus$-BP heißt *read-once*, wenn jede Variable x_u, $1 \leq u \leq n$, die Markierung höchstens einer Kante ist.

Lemma 4.9. *Jede read-once $\{\oplus, \wedge\}$-Formel der Größe ℓ kann durch ein read-once $\oplus$-BP mit höchstens $\ell + 1$ Knoten simuliert werden.*

Beweis. Wir beweisen das Lemma per Induktion über die Formeltiefe d. Formeln der Tiefe $d = 0$ sind Konstanten oder Variablen und haben Größe $\ell = 1$. Abbildung 4.3(a)–(c) zeigt die zugehörigen read-once $\oplus$-BPs mit 2 Knoten. Im Induktionsschritt dürfen wir annehmen, dass wir read-once $\oplus$-BPs für Subformeln mit geringerer Tiefe als d und Größe ℓ_g und ℓ_h haben, die Funktionen g bzw. h repräsentieren. Offensichtlich erhalten wir ein $\oplus$-BP für eine Formel, die $g \oplus h$ repräsentiert, indem wir die beiden Quellen und die beiden Senken der $\oplus$-BPs für g und h verschmelzen (Abbildung 4.3(d)). Ein $\oplus$-BP für eine Formel, die $g \wedge h$ repräsentiert,

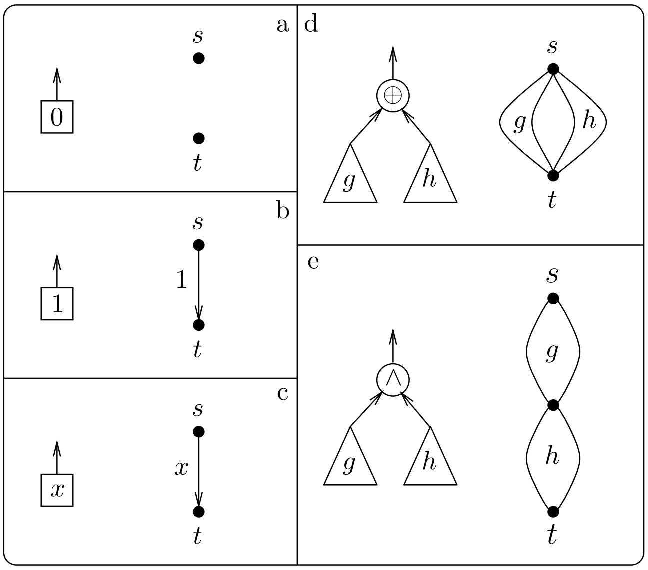


Abbildung 4.3: Formeln und zugehörige $\oplus$-BPs.

erhalten wir, indem wir die Senke des $\oplus$-BP für g mit der Quelle des $\oplus$-BPs für h verschmelzen (Abbildung 4.3(e)). Das zusammengesetzte $\oplus$-BP akzeptiert eine Eingabe genau dann, wenn jeder von einer ungeraden Anzahl von Pfaden im $\oplus$-BP für g durch eine ungerade Anzahl Pfade im $\oplus$-BP für h verlängert wird. In beiden Fällen hat das resultierende $\oplus$-BP höchstens $(\ell_g + 1) + (\ell_h + 1) - 1 = \ell_g + \ell_h + 1 = \ell + 1$ Knoten. Die read-once-Eigenschaft folgt unmittelbar aus der Konstruktion und der Tatsache, dass g und h disjunkte Variablenmengen haben. ☐

Definition 4.10. Ein *linear bijection straight-line Programm* (LBSP) über einem Ring $(\mathcal{R}, +, \cdot, 0, 1)$ ist eine Folge von Anweisungen der Form

$$\begin{aligned} &R_j \leftarrow R_j{+}(R_i \cdot c),\\ &R_j \leftarrow R_j{-}(R_i \cdot c),\\ &R_j \leftarrow R_j{+}(R_i \cdot x_u) \quad \text{oder}\\ &R_j \leftarrow R_j{-}(R_i \cdot x_u), \end{aligned}$$

wobei $R_1, \ldots, R_w$ Register sind, die Zahlen aus $\mathcal{R}$ aufnehmen können, $x_1, \ldots, x_n$ die Eingabevariablen bezeichnen, $c \in \mathcal{R}$ eine Konstante ist, $i, j \in \{1, \ldots, w\}$ und $i \neq j$. Die *Länge* eines LBSPs ist die Anzahl seiner Anweisungen, die *Breite* ist die Anzahl w seiner Register.

Für jede Eingabe $a = (a_1, \ldots, a_n) \in \mathcal{R}^n$ ist jede einzelne Anweisung eines LBSPs eine lineare Abbildung der w Register auf sich selbst, präziser gesagt, des Vektorraumes $\mathcal{R}^w$ in sich selbst. Wenn wir die Eingabe nicht fixieren, dann lässt sich jede Anweisung durch eine quadratische Matrix $A \in \{\mathcal{R} \cup X\}$ darstellen, wobei $X = \{x_1, \dots, x_n, -x_1, \dots, -x_n\}$. Die Hauptdiagonale von A ist mit Einsen gefüllt. Für eine Anweisung $R_j \leftarrow R_j \pm (R_i \cdot c)$ ist $A_{ij} = \pm c$ und für eine Anweisung $R_j \leftarrow R_j \pm (R_i \cdot x_u)$ ist $A_{ij} = \pm x_u$. Alle übrigen Einträge sind 0. Hier zwei Beispiele für $w = 3$.

$$R_2 \leftarrow R_2 + (R_3 \cdot c)\colon \qquad \big(R_1, R_2, R_3\big) \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & +c & 1 \end{pmatrix} = \big(R_1, R_2 + (R_3 \cdot c), R_3\big)$$

$$R_3 \leftarrow R_3 - (R_1 \cdot x_u)\colon \qquad \big(R_1, R_2, R_3\big) \cdot \begin{pmatrix} 1 & 0 & -x_u \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \big(R_1, R_2, R_3 - (R_1 \cdot x_u)\big)$$

Der Name *linear bijection* straight-line Programm verrät schon, dass diese linearen Abbildungen bijektiv sind. Genau genommen gilt dies erst, sobald wir eine Belegung der Variablen mit Werten aus $\mathcal{R}$ fixieren. Dann treten an die Stelle der Variablen x_u die Eingaben a_u und die gerade betrachteten Matrizen werden zu Matrizen über dem Ring $\mathcal{R}$. Aus der linearen Algebra ist bekannt, dass es für den Nachweis der Bijektivität genügt, zu zeigen, dass $\det A \neq 0$. Man kann leicht einsehen, dass für jede Anweisung eines LBSPs mit zugehöriger Matrix A und Variablenbelegung $a \in \mathcal{R}^n$ gilt, $\det A = 1$. Sei A_{ij} der Eintrag für Anweisungen wie oben beschrieben, also $i \neq j$. Wenn man z.B. von der i-ten Zeile das c-fache bzw. a_u-fache der j-ten Zeile subtrahiert, wird $A_{ij} = 0$. Bei diesen Zeilenumformungen bleibt die Determinante unverändert und wir erhalten die Einheitsmatrix mit Einsen auf der Hauptdiagonalen und allen übrigen Einträgen 0. Diese hat Determinante 1.

Ein LBSP mit Anweisungen $I_1, \dots, I_m$ ist also für jede Eingabe $a \in \mathcal{R}^n$ eine Hintereinanderschaltung von linearen Bijektionen, die man auch als Produkt der Matrizen der einzelnen Anweisungen $A_{I_1} \cdots A_{I_m}$ aufschreiben kann. Welche Abbildung insgesamt berechnet wird, steuert die Variablenbelegung a im Rahmen der Möglichkeiten, die ein konkretes Programm zulässt. In dieser Sichtweise bedeutet es ein LBSP auszuführen, einen Zeilenvektor $r = (r_1, \dots, r_w)$ mit Anfangswerten für die w Register von rechts mit den zugehörigen Matrizen $A_{I_1}, \dots, A_{I_m}$ zu multiplizieren:

$$(r_1, \dots, r_w) \cdot A_{I_1} \cdots A_{I_m}.$$

Definition 4.11. Wir sagen, ein LBSP *ändert* ein Register R_j um $R_i \cdot f(x_1, \dots, x_n)$, wenn es den Inhalt von R_j durch $R_j + R_i \cdot f(x_1, \dots, x_n)$ ersetzt und alle anderen Register im Endergebnis unverändert lässt. Ein LBSP *berechnet* eine Funktion $f(x_1, \dots, x_n)$, wenn es für zwei verschiedenen Register R_j und R_i das Register R_j um $R_i \cdot f(x_1, \dots, x_n)$ ändert.

Definition 4.12. Die Zahl der *Variablenreferenzen* eines LBSPs ist die Zahl seiner Anweisungen der Form $R_j \leftarrow R_j \pm (R_i \cdot x_u)$. Ein LBSP heißt *read-once*, wenn jede

Variable x_u, $1 \leq u \leq n$, in höchstens einer Anweisung vorkommt. Entsprechend heißt ein LBSP *read-k-times*, wenn jede Variable x_u in höchstens k Anweisungen vorkommt.

Lemma 4.13. *Für jede boolesche Funktion $f(x_1, \ldots, x_n)$, die von einem read-once $\oplus$-BP mit ℓ Knoten berechnet wird , gibt es für alle $i, j \in \{0, \ldots, \ell\}$, $i \neq j$, ein read-4-times LBSP über dem Ring $\mathbb{Z}_2 = (\{0, 1\}, \oplus, \wedge, 0, 1)$ mit Breite $\ell + 1$, das R_j um $R_i \wedge f(x_1, \ldots, x_n)$ ändert.*

Beweis. Die Idee diese Beweises ist es, für jeden Knoten des $\oplus$-BPs die Zahl der Pfade zu berechnen, die an diesem Knoten starten und die Senke t erreichen. Es genügt, diese Rechnung im Ring $\mathbb{Z}_2$ durchzuführen, d. h. für jeden Knoten den Wert der (Sub-)Funktion zu berechnen, die der Knoten repräsentiert. Seien $R_0, \ldots, R_\ell$ die Register eines LBSPs mit Breite $\ell + 1$. Es genügt, wenn wir ein LBSP konstruieren, das R_1 um $R_0 \wedge f(x)$ ändert. Für ein Programm das für beliebige aber verschiedene $i, j \in \{0, \ldots, \ell\}$ R_j um $R_i \wedge f(x)$ ändert brauchen wir nur die Register geeignet umzubenennen. Wir fixieren eine topologische Ordnung der Knoten des gegebenen $\oplus$-BPs, sodass die Quelle s die kleinste Nummer 1 erhält und die Senke die größte Nummer ℓ. Für alle i, $1 \leq i \leq \ell$, weisen wir dem Knoten mit Nummer i das Register R_i zu.

Wir konstruieren das LBSP in drei Schritten, die drei aufeinanderfolgenden Phasen in der Rechnung des LBSPs entsprechen. So bezeichnen wir z. B. mit Phase 1 den Teil des LBSPs, der in Schritt 1 konstruiert wird. In Phase 1 und Phase 2 wird die eigentliche Rechnung durchgeführt, während Phase 3 die Registerinhalte aller Register, ausgenommen Register R_1, wiederherstellt.

Schritt 1. Besuche die Knoten des $\oplus$-BPs in topologischer Reihenfolge $i = 1, \ldots, \ell - 1$ und generiere für jede ausgehende Kante $i \xrightarrow{1} j$ (d. h. $i < j$) eine Anweisung $R_i \leftarrow R_i \oplus (R_j \wedge 1)$ und für jede ausgehende Kante $i \xrightarrow{x_u} j$ eine Anweisung $R_i \leftarrow R_i \oplus (R_j \wedge x_u)$.

Schritt 2. Generiere zuerst eine Anweisung $R_\ell \leftarrow R_\ell \oplus (R_0 \wedge 1)$ und füge dann alle Anweisungen aus Schritt 1 in umgekehrter Reihenfolge hinten an.

Schritt 3. Füge alle Anweisungen aus Schritt 1 und Schritt 2, ausgenommen Zuweisungen an Register R_1, in umgekehrter Reihenfolge hinten an.

Sei $f_i(x)$, $1 \leq i \leq \ell$, die (Sub-)funktion, die Knoten i repräsentiert, und sei r_i der Anfangswert in Register R_i. Wir zeigen per Induktion, dass für alle $i \in \{\ell, \ldots, 1\}$ Phase 1 und Phase 2 zusammen R_i um $R_0 \wedge f_i(x)$ ändern. Der Senkenknoten mit Nummer ℓ repräsentiert die Funktion $f_\ell(x) \equiv 1$. In Phase 1 gibt es keine Anweisung, die R_ℓ verändert. Die erste Anweisung in Phase 2 ändert, wie gewünscht, R_ℓ um $R_0 \wedge 1$, sodass $r_\ell \oplus (r_0 \wedge 1)$ der neue Wert in R_ℓ wird. Im Induktionsschritt

$(i \mapsto i-1)$ gilt $i < \ell$. Falls es eine Kante $i \xrightarrow{1} j$ gibt, dann gibt es in Phase 1 eine Anweisung $R_i \leftarrow R_i \oplus (R_j \wedge 1)$, die R_i um r_j ändert. In Phase 2 wir der neue Wert von R_j, $r_j \oplus (r_0 \wedge f_j(x))$, berechnet bevor das Programm zu Knoten i zurückkehrt, weil der Knoten j in der umgekehrten Reihenfolge vor Knoten i besucht wird. Wenn das Programm in Phase 2 zum Knoten i zurückkehrt, wird die Anweisung $R_i \leftarrow R_i \oplus (R_j \wedge 1)$ erneut ausgeführt, sodass R_i um $r_j \oplus (r_0 \wedge f_j(x))$ geändert wird. Beide Anweisungen für eine Kante $i \xrightarrow{1} j$ in Phase 1 und Phase 2 zusammen ändern R_i um $r_j \oplus (r_j \oplus (r_0 \wedge f_j(x)) = r_0 \wedge f_j(x)$. Mit denselben Argumenten wird für eine Kante $i \xrightarrow{x_u} j$ Register R_i in Phase 1 um $r_j \wedge x_u$ geändert und in Phase 2 um $(r_j \oplus (r_0 \wedge f_j(x))) \wedge x_u$ geändert. Beide Anweisungen zusammen ändern R_i um $(r_j \wedge x_u) \oplus ((r_j \oplus (r_0 \wedge f_j(x))) \wedge x_u)$. Falls $x_u = 0$, bleibt R_i also durch eine x_u-Kante unverändert und wird für $x_u = 1$ um $r_0 \wedge f_j(x)$ geändert. Für eine Eingabe a ändern alle ausgehenden Kanten eines Knotens i zusammen das Register R_i um

$$\bigoplus_{(i,j)\in E_a} \big(r_0 \wedge f_j(a)\big) = r_0 \wedge \Big(\bigoplus_{(i,j)\in E_a} f_j(a)\Big) = r_0 \wedge f_i(a).$$

Insbesondere haben wir gezeigt, dass in Phase 1 und Phase 2 Register R_1 um $r_0 \wedge f_1(x)$ geändert wird. Da Knoten 1 der Quellenknoten des Parity-Branchingprogramms ist, an dem $f(x) = f_1(x)$ berechnet wird, wird R_1 um $r_0 \wedge f(x))$ geändert.

Wird eine Anweisung I eines LBSPs über $\mathbb{Z}_2$ zweimal, unmittelbar hintereinander ausgeführt, hebt sich die Wirkung von I gerade wieder auf. Die Anweisungen eines LBSPs über $\mathbb{Z}_2$ sind selbstinverse Abbildungen. Es gilt $A_I \cdot A_I = 1$. Seien $I_1, \ldots, I_m$ die Anweisungen, die in Schritt 1 und Schritt 2 generiert wurden. Angenommen wir würden in Schritt 3 diese Anweisungen in umgekehrter Reihenfolge hinten anfügen:

$$\underbrace{A_{I_1} \cdots \underbrace{A_{I_{m-1}} \cdot \underbrace{A_{I_m} \cdot A_{I_m}}_{1} \cdot A_{I_{m-1}}}_{1} \cdots A_{I_1}}_{1}\,.$$

Die zweite Hälfte des Programms würde die Wirkung der ersten Hälfte gerade aufheben, d. h. alle Registerinhalte würden wiederhergestellt. Deswegen werden in Schritt 3 Zuweisungen an Register R_1 ausgelassen, sodass R_1 in Phase 3 unverändert bleibt. Man möchte vielleicht vermuten, dass dadurch die korrekte Wiederherstellung der übrigen Register gefährdet wird. Glücklicherweise ist dies nicht der Fall. Weil R_1 dem Quellenknoten zugeordnet ist, der keine eingehenden Kanten besitzt, wird in Phase 1 und Phase 2 keine Anweisung erzeugt, die ein Vielfaches von R_1 zu einem anderen Register hinzuaddiert. Daher ist der Inhalt von R_1 in Phase 3 bedeutungslos.

Die behauptete read-4-times-Eigenschaft folgt aus der Tatsache, dass jede Kante des read-once $\oplus$-BP in Schritt 1 in genau eine Anweisung umgewandelt wird, die später in Schritt 2 einmal kopiert und in Schritt 3 dann noch zweimal kopiert wird. □

Lemma 4.14. *Jede read-once* $\{\oplus, \wedge\}$*-Formel der Größe* ℓ *und Tiefe* d *kann für jedes* $k \in \mathbb{N}$ *durch ein LBSP über dem Ring* $\mathbb{Z}_2 = (\{0,1\}, \oplus, \wedge, 0, 1)$ *mit Breite höchstens* $2^k + 2$ *simuliert werden, das höchstens* $4\ell \cdot \left(2^d\right)^{\frac{2}{k}}$ *Variablenreferenzen beinhaltet.*

Die Konstruktion im folgenden Beweis geht auf Cleve (1991) zurück. Cleve hat die Länge der konstruierten LBSPs analysiert, während wir hier die Zahl der Variablenreferenzen untersuchen. Lemma 4.9 und Lemma 4.13 dienen uns dabei als wichtige „Bausteine“.

Beweis. In diesem Beweis betrachten wir read-once Formeln als binäre Bäume, deren Wurzel der oberste Knoten auf Level 1 ist. Die Variablen und Konstanten sind die Blätter des Baumes. O.B.d.A. platzieren wir alle Blätter einer Formel der Tiefe d auf dem untersten Level $d+1$. Das bedeutet, dass Kanten nicht nur zwischen aufeinander folgenden Leveln verlaufen dürfen, sondern auch ein oder mehrere Level überspannen dürfen.

Sei also k eine beliebige natürliche Zahl. Es ist hilfreich, sich k zunächst als Konstante kleiner als d vorzustellen. Die gegebene Formel der Tiefe d berechne $f(x_1, \ldots, x_n)$ und $R_1, \ldots, R_{2^k+2}$ seien die verfügbaren Register mit Indexmenge $I = \{1, \ldots, 2^k + 2\}$. Wir werden für alle $i, j \in I$, $i \neq j$, LBSPs konstruieren, die R_j um $R_i \wedge f(x_1, \ldots, x_n)$ ändern. Zur Notation: $P_{j \oplus (i \wedge f)}$ bezeichnet im Folgenden das Programm, das R_j um $R_i \wedge f(x_1, \ldots, x_n)$ ändert. Dabei sind „$\oplus$“ und „$\wedge$“ im Index von P einfach nur syntaktische Zeichen, die *nicht* zu entsprechenden Rechnungen in $\mathbb{Z}_2$ auffordern sollen.

Unsere Konstruktion geht induktiv über die Formeltiefe d vor, wobei wir in jedem Schritt nicht nur ein Level der Formel, sondern gleich die jeweils k obersten Level in ein LBSP transformieren. Formeln der Tiefe $d = 0$ sind einzelne Variablen x_u oder Konstanten c und können durch read-once LBSPs mit nur einer Anweisung $R_j \leftarrow R_j \oplus (R_i \wedge x_u)$ bzw. $R_j \leftarrow R_j \oplus (R_i \wedge c)$ simuliert werden, deren Breite genau 2 ist.

Im Induktionsschritt ($d \mapsto d + k$) dürfen wir annehmen, dass wir schon für alle Formeln mit Tiefen bis zu d, die eine beliebige Funktion h berechnen, und alle möglichen Registerpaare $R_{i'}$ und $R_{j'}$, mit $i', j' \in I$ und $i' \neq j'$, entsprechende LBSPs $P_{j' \oplus (i' \wedge h)}$ konstruiert haben. Unsere Aufgabe ist es jetzt, Programme für Formeln mit Tiefe bis zu $d + k$ zu konstruieren, und zwar für alle möglichen Registerpaare R_j und R_i. Dazu trennen wir die obersten k Level der Formel ab (siehe Abbildung 4.4) und fassen den oberen Teil als Formel für eine Funktion g auf, die ehemals mit $m \leq 2^k$ Subformeln verbunden war. Diese m Subformeln für Funktionen $h_1 \ldots, h_m$ ersetzten wir durch m neue Variablen $y_1, \ldots, y_m$. Jeder Variablen y_u, $1 \leq u \leq m$, ist so genau eine Subfunktion h_u zugeordnet. Die Funktion

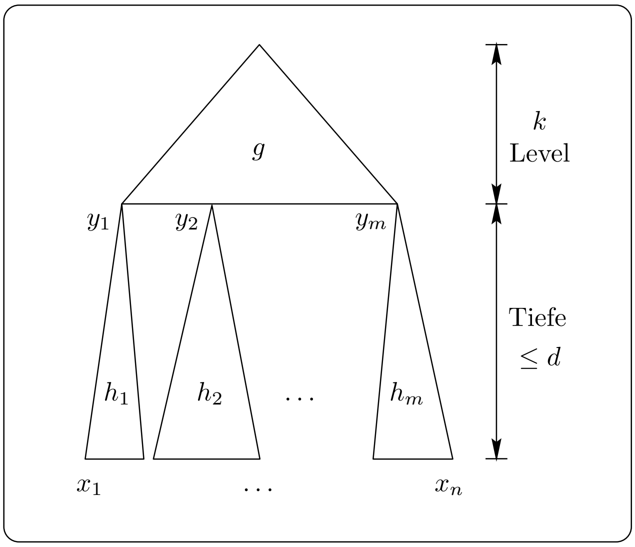


Abbildung 4.4: $f(x_1,\dots,x_n) = g\big(h_1(x_1,\dots,x_n),\dots,h_m(x_1,\dots,x_n)\big)$.

$f(x_1,\dots,x_n)$ kann nun als $g(h_1(x_1,\dots,x_n),\dots,h_m(x_1,\dots,x_n))$ dargestellt werden, wobei die Formel jeder Subfunktion h_u höchstens Tiefe d hat und nur von einem Teil der Variablen $x_1,\dots,x_n$ essentiell abhängt. Offensichtlich ist die Formel für $g(y_1,\dots,y_m)$ per Definition eine read-once Formel und hat Größe höchstens 2^k. Deswegen können wir nun Lemma 4.9 anwenden und erhalten ein read-once $\oplus$-BP für $g(y_1,\dots,y_m)$ mit höchstens 2^k+1 Knoten. Dann wenden wir Lemma 4.13 an und erhalten für alle $i,j \in I$, $i \neq j$, ein read-4-times LBSP $P_{j\oplus(i\wedge g)}$ dessen Breite durch 2^k+2 beschränkt ist. Jetzt benutzen wir die Induktionsvoraussetzung und ersetzen in jedem Programm $P_{j\oplus(i\wedge g)}$ für jedes $u \in \{1,\dots,m\}$ jede Anweisung $R_{j'} \leftarrow R_{j'} \oplus (R_{i'} \wedge y_u)$ durch das Programm $P_{j'\oplus(i'\wedge h_u)}$. Insgesamt haben wir für alle verschiedenen $i,j \in I$ LBSPs $P_{j\oplus(i\wedge f)}$ mit Breite höchstens 2^k+2 konstruiert, die R_j um $R_i \wedge g(h_1(x_1,\dots,x_n),\dots,h_m(x_1,\dots,x_n))$ ändern, d. h. $f(x_1,\dots,x_n)$ berechnen.

Die Häufigkeit, mit der eine Variable x_u in einem Programm $P_{j\oplus(i\wedge f)}$ referenziert wird, hängt von k ab. Eine Variable auf Level 1 wir nur einmal getestet, weil die gesamte Formel dann allein aus dieser Variablen besteht, die durch ein LBSP mit nur einer Anweisung simuliert wird. (Induktionsanfang). Eine Variable x_u auf einem der Level von 2 bis k wird höchstens 4-mal referenziert, weil das LBSP, das wir für die obersten k Level konstruieren, read-4-times ist (Abbildung 4.5). Auch eine Variable x_u auf Level $k+1$ ist read-4-times, weil x_u eine Subformel bildet, die zunächst durch ein read-once LBSP simuliert wird. Dieses read-once LBSP wird 4-mal als Ersatz für eine Anweisung im übergeordneten read-4-times LBSP für die obersten k Level benötigt. Eine Variable x_u auf einem der Level $k+2$ bis $2k$ wird in einem Programm für f höchstens 4^2-mal referenziert, weil x_u in eine Subformel, sagen wir für $h_{v(u)}$, fällt, deren zugehöriges LBSP $P_{v(u)}$ read-once oder read-4-times

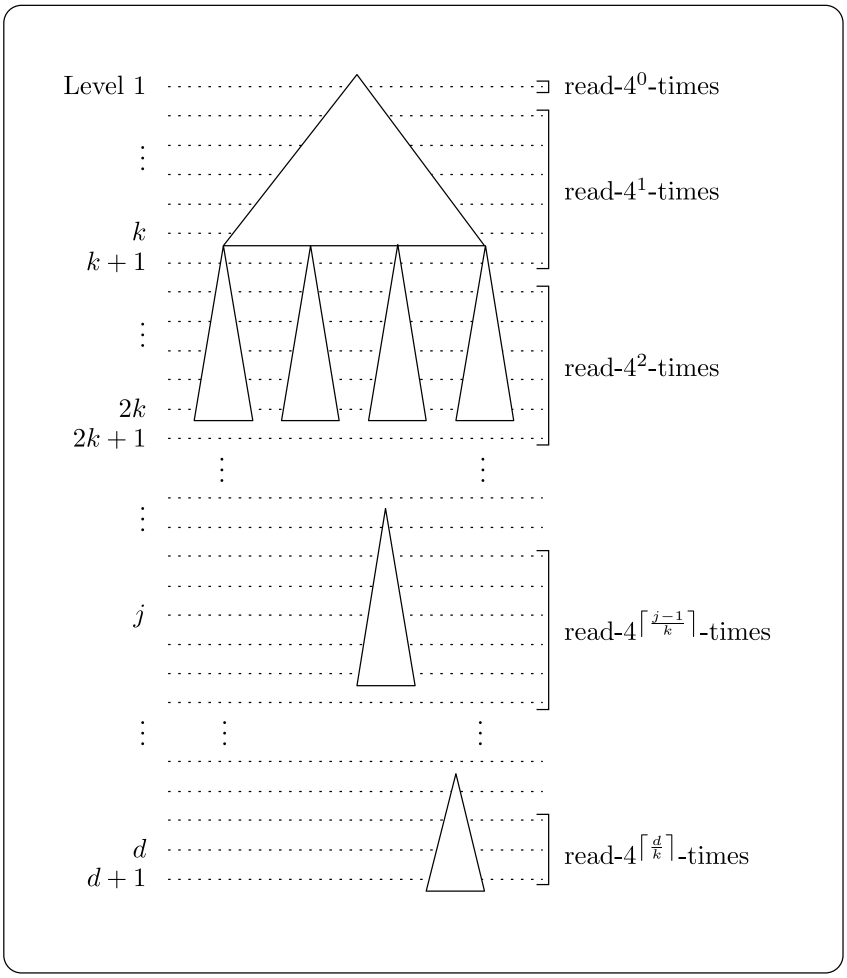


Abbildung 4.5: Variablenreferenzen.

ist und $P_{v(u)}$ 4-mal in das übergeordnete LBSP für die obersten k Level eingesetzt wird. Mit dem gleichen Argument wie oben sind auch Variablen auf dem Level $2k+1$ read-4^2-times. Allgemein wird eine Variable auf Level j, $1 \leq j \leq d+1$, höchstens $4^{\lceil \frac{j-1}{k} \rceil}$-mal in einem LBSP für f referenziert. Da wir alle Variablen auf dem untersten Level $d+1$ platzieren, analysieren wir den Worst-Case. Eine read-once Formel der Größe ℓ hat höchstens ℓ Variablen. Es gibt also in jedem unserer Programme $P_{j \oplus (i \wedge f)}$ höchstens

$$\ell \cdot 4^{\lceil \frac{d}{k} \rceil} \leq \ell \cdot 2^{2\left(\frac{d}{k}+1\right)} = 4\ell \cdot \left(2^d\right)^{\frac{2}{k}}$$

Variablenreferenzen. □

Mit den Ideen des Beweises von Satz 3.7 können wir erkennen, dass Lemma 4.14 gültig bleibt, wenn wir die Read-once-Eigenschaft fallen lassen: Wir entfalten eine beliebige Formel durch Duplizieren der Variablen und Konstanten zu einem Baum, ersetzen die Blätter des Baumes durch neue Variablen, wenden Lemma 4.14 an und machen die Ersetzung rückgängig.

An dieser Stelle wollen wir unsere Zwischenergebnisse zusammenfassen. Wir starten mit einer beliebigen B_2-Formel mit Größe ℓ und erhalten mit Korollar 4.7 eine $\{\oplus, \wedge\}$-Formel mit Tiefe höchstens $3c \cdot \ln 2 \cdot \log(3\ell)$ und Größe höchstens $(3\ell)^\alpha$. Dann wenden wir Lemma 4.14 an und erhalten ein LBSP mit Breite höchstens $2^k + 2$. Die folgende Rechnung schätzt die Zahl der Variablenreferenzen ab.

$$\begin{aligned}
\#\text{Variablenreferenzen} &\leq 4 \cdot (3\ell)^\alpha \cdot \left(2^{3c \cdot \ln 2 \cdot \log(3\ell)}\right)^{\frac{2}{k}} \\
&= 4 \cdot 3^\alpha \cdot \ell^\alpha \cdot 2^{6\frac{c}{k} \cdot \ln 2 \cdot \log(3\ell)} \\
&= 4 \cdot 2^{\alpha \cdot \log 3} \cdot \ell^\alpha \cdot 2^{6\frac{c}{k} \cdot \ln 2 \cdot \log 3 + 6\frac{c}{k} \cdot \ln 2 \cdot \log \ell} \\
&= 2^{\alpha \cdot \log 3 + 2} \cdot \ell^\alpha \cdot 2^{6\frac{c}{k} \cdot \ln 2 \cdot \log 3} \cdot 2^{6\frac{c}{k} \cdot \ln 2 \cdot \log \ell} \\
&= 2^{\alpha \cdot \log 3 + 6\frac{c}{k} \cdot \ln 2 \cdot \log 3 + 2} \cdot \ell^\alpha \cdot \ell^{6\frac{c}{k} \cdot \ln 2} \\
&= 2^{(\alpha + 6\frac{c}{k} \cdot \ln 2) \cdot \log 3 + 2} \cdot \ell^{\alpha + 6\frac{c}{k} \cdot \ln 2}
\end{aligned}$$

Korollar 4.15. *Jede B_2-Formel der Größe ℓ kann für jedes $k \in \mathbb{N}$ und jedes beliebige $c \geq 2$ durch ein LBSP über dem Ring $\mathbb{Z}_2 = (\{0,1\}, \oplus, \wedge, 0, 1)$ mit Breite höchstens $2^k + 2$ simuliert werden, das höchstens $2^{\beta \log 3 + 2} \cdot \ell^\beta$ Variablenreferenzen beinhaltet, wo $\beta = 1 + \frac{1}{1 + \log(c-1)} + 6\frac{c}{k} \ln 2$.*

Straight-line Programme in Branchingprogramme konvertieren

Definition 4.16 (Barrington (1989)). Ein *Permutations-Branchingprogramm* (PBP) der Breite w und Länge ℓ ist durch eine Folge von Anweisungen der Form (u, ρ_0, ρ_1) gegeben, wobei u, $1 \leq u \leq n$, ein Variablenindex ist, der eine Variable $x_1, \ldots, x_n$ bezeichnet, und $\rho_0, \rho_1 \in S_w$ Permutationen sind. Es sei $\sigma_i = \rho_0$, falls $x_u = 0$ und $\sigma_i = \rho_1$, falls $x_u = 1$. Dann berechnet ein Permutations-Branchingprogramm für eine Variablenbelegung x die Permutation $\sigma(x) = \sigma_\ell \circ \cdots \circ \sigma_1$.

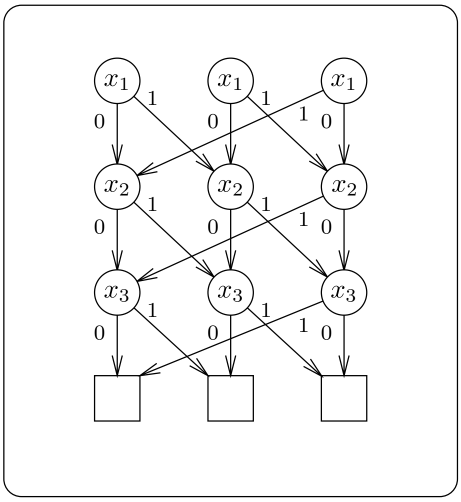
Die Graphdarstellung eines Permutations-Branchingprogramms ist eine rechteckige Anordnung von $(\ell + 1) \cdot w$ Knoten in $\ell + 1$ Zeilen und w Spalten. Jede Zeile (oder Level) i, $1 \leq i \leq \ell$, besitzt w Knoten $v_{i,1}, \ldots, v_{i,w}$, die alle mit derselben Variablen $x_{u(i)}$ markiert sind, und realisiert die i-te Anweisung des Programms. Auf den Leveln $1, \ldots, \ell$ hat jeder Knoten je eine ausgehende 0- und 1-Kante. Die 0-Kante des Knotens $v_{i,j}$ weist auf den Knoten $v_{i+1,\rho_0(j)}$, die 1-Kante auf den Knoten $v_{i+1,\rho_1(j)}$. Die Knoten auf Level $\ell + 1$ tragen keine Markierungen und haben keine ausgehenden Kanten. Sie heißen daher Senken. Die Knoten auf Level 1 heißen Quellen.

$\big(x_1, id, (1\ 2\ 3)\big)$,

$\big(x_2, id, (1\ 2\ 3)\big)$,

$\big(x_3, id, (1\ 2\ 3)\big)$

(a) Anweisungsfolge



(b) Graphdarstellung

Abbildung 4.6: PBP für $f(x_1, x_2, x_3) = [x_1 + x_2 + x_3 \equiv 1 \bmod 3]$ für $\tau = (1\ 2\ 3)$.

Definition 4.17. Ein Permutations-Branchingprogramm mit Breite w *berechnet* $f \in B_n$ für $\tau \in S_w$, $\tau \neq id$, wenn

$$\sigma(x) = \begin{cases} id & \text{für alle } x \in f^{-1}(0), \\ \tau & \text{für alle } x \in f^{-1}(1). \end{cases}$$

Beispiel 4.18. Abbildung 4.6 zeigt als Beispiel ein Permutations-Branchingprogramm, dass für $\tau = (1\ 2\ 3)$ (Zykelschreibweise) die Funktion $f(x_1, x_2, x_3) = [x_1 + x_2 + x_3 \equiv 1 \bmod 3]$ berechnet. Die Schreibweise „[*Ausdruck*]“ steht hier wieder für den booleschen Wert 1, falls *Ausdruck* wahr ist, und sonst für 0.

Wir haben schon gesehen, dass für eine feste Variablenbelegung jede Anweisung eines LBSPs mit Breite w eine bijektive, lineare Abbildung $h\colon \mathcal{R}^w \to \mathcal{R}^w$ berechnet. Die Anweisungen eines Permutations-Branchingprogramms mit Breite w beinhalten spezielle bijektiver Abbildungen, nämlich Permutationen $\sigma\colon \{1, \ldots, w\} \to \{1, \ldots, w\}$. Diese Gemeinsamkeit können wir ausnutzen.

Lemma 4.19. *Jedes LBSP über dem Ring* $\mathbb{Z}_2 = (\{0,1\}, \oplus, \wedge, 0, 1)$ *mit Breite* w*, das eine Funktion* $f \in B_n$ *berechnet und* r *Variablenreferenzen beinhaltet, kann durch ein Permutations-Branchingprogramm mit Länge* $\max\{r, 1\}$ *und Breite* 2^w *simuliert werden.*

Beweis. Sei $\{x_1, \ldots, x_n\}$ die Variablenmenge und ℓ die Länge eines gegebenen LBSPs. Sei $S = \{0,1\}^w$ die Menge aller möglichen Registerbelegungen eines LBSPs mit Breite w. Bei seiner Rechnung führt das LBSP eine Folge von ℓ Anweisungen

$I_1, \ldots, I_\ell$ aus. Die Rechnung beginnt mit einer beliebigen Registerbelegung s_0 und endet mit einer finalen Registerbelegung s_ℓ.

$$s_0 \xrightarrow{I_1} s_1 \xrightarrow{I_2} s_2 \longrightarrow \ldots \xrightarrow{I_\ell} s_\ell, \quad s_i \in S$$

Für eine konkrete Variablenbelegung a berechnet jede einzelne Anweisung I_j des LBSPs eine bijektive, lineare Abbildung $h_{I_k} : S \to S$ der w Register auf sich selbst. Wir simulieren jede einzelne Anweisung I_k, $1 \leq k \leq \ell$, des LBSPs durch genau eine Anweisung $(x_{u(k)}, \sigma_{k,0}, \sigma_{k,1})$ eines Permutations-Branchingprogramms mit Breite 2^w. In dem Ring $\mathbb{Z}_2$ ist „$+ = \oplus$“ und auch „$- = \oplus$“. Daher gibt es nur drei verschiedene Anweisungtypen in LBSPs über $\mathbb{Z}_2$ (vgl. Definition 4.10), nämlich

$$\begin{aligned} R_j &\leftarrow R_j \oplus (R_i \wedge 0), && (1)\\ R_j &\leftarrow R_j \oplus (R_i \wedge 1) \qquad \text{und} && (2)\\ R_j &\leftarrow R_j \oplus (R_i \wedge x_u). && (3) \end{aligned}$$

Anweisungen der Form (1) bewirken keine Veränderung. Wir setzen $\sigma_{k,0} = \sigma_{k,1} = id$ und wählen irgendeine Variable für $x_{u(k)}$ aus. Typ-(2)-Anweisungen simulieren wir durch $\sigma_{k,0} = \sigma_{k,1} = \rho_{j,i}$, wo

$$\rho_{j,i}(r_w, \ldots, r_j, \ldots, r_i, \ldots, r_1) = (r_w, \ldots, r_j \oplus r_i, \ldots, r_i, \ldots, r_1)$$

und $(r_w, \ldots, r_j, \ldots, r_i, \ldots, r_1) \in S$ eine Registerbelegung beschreibt. Dabei sind r_i und r_j die Inhalte der Register R_i bzw. R_j. Man kann leicht einsehen, dass die Abbildung $\rho_{j,i}$ eine Permutation ist. Ein Zustand mit $r_i = 0$ wird auf sich selbst abgebildet, während Zustände mit $r_i = 1$, $(\ldots, r_j, \ldots, 1, \ldots)$, wechselseitig mit Zuständen $(\ldots, \overline{r}_j, \ldots, 1, \ldots)$ vertauscht werden. Daher ist σ_k ein Produkt von Transpositionen und somit eine Permutation. Bei der Form (3) brauchen wir zwei verschiedene Permutationen $\sigma_{k,0} = id$ und $\sigma_{k,1} = \rho_{j,i}$ für $x_{u(k)} = 0$ bzw. $x_{u(k)} = 1$. Insgesamt berechnet das Permutations-Branchingprogramm $\sigma = \sigma_{\ell,x_{u(\ell)}} \circ \cdots \circ \sigma_{1,x_{u(1)}}$ und simuliert so jede einzelne Anweisung des LBSPs.

Für Anweisungen des Typs (1) und (2) haben wir im Permutations-Branchingprogramm Zeilen der Form $(x_{u(k)}, \sigma_k, \sigma_k)$ erzeugt. Solche Zeilen können mit ihrer Vorgänger- oder Nachfolgeranweisung verschmolzen werden:

$$\begin{matrix} (x_{u(k-1)}, \sigma_{k-1,0}, \sigma_{k-1,1}), \\ (x_{u(k)}, \sigma_k, \sigma_k) \end{matrix} \quad \rightsquigarrow \quad (x_{u(k-1)}, \sigma_k\sigma_{k-1,0}, \sigma_k\sigma_{k-1,1})$$

$$\begin{matrix} (x_{u(k)}, \sigma_k, \sigma_k) \\ (x_{u(k+1)}, \sigma_{k+1,0}, \sigma_{k+1,1}), \end{matrix} \quad \rightsquigarrow \quad (x_{u(k+1)}, \sigma_{k+1,0}\sigma_k, \sigma_{k+1,1}\sigma_k).$$

Das Ergebnis einer Verschmelzung ist nur dann eine Anweisung von der Form $(x_{u(k)}, \sigma_{k,0}, \sigma_{k,1})$, $\sigma_{k,0} \neq \sigma_{k,1}$, wenn die beteiligte Vorgänger- bzw. Nachfolgeranweisung von dieser Form war. Ansonsten erhält man erneut eine Anweisung von der

Form $(x_{u(k)}, \sigma_k, \sigma_k)$, die dann ihrerseits wieder verschmolzen werden kann. Die Zahl der Anweisungen vom Typ $(x_{u(k)}, \sigma_{k,0}, \sigma_{k,1})$ im Permutations-Branchingprogramm ist zu Beginn der Verschmelzungen gleich der Zahl der Typ-(3)-Anweisungen im LBSP, also gleich der Zahl der Variablenreferenzen r. Bei einer Verschmelzung bleibt diese Zahl unverändert. Alle anderen Anweisungen können durch wiederholtes Verschmelzen eliminiert werden, vorausgesetzt, dass es mindestens eine Typ-(3)-Anweisung im LBSP gibt. Falls $r = 0$ lassen sich alle Anweisungen zu einer einzigen Anweisung verschmelzen.

Das gegebene LBSP berechnet eine Funktion f, d. h. es ändert für gewisse Register R_i und R_j das Register R_j um $R_i \wedge f(x_1, \ldots, x_n)$:

$$\begin{aligned} s_0 &= (r_{w-1}, \ldots, r_j, \ldots, r_i, \ldots, r_0) \\ &\vdots \\ s_l &= (r_{w-1}, \ldots, r_j \oplus (r_i \wedge f(x_1, \ldots, x_n)), \ldots, r_i, \ldots, r_0). \end{aligned}$$

Das konstruierte Permutations-Branchingprogramm simuliert diese Rechnung und berechnet die Permutation

$$\sigma(r_{w-1}, \ldots, r_j, \ldots, r_i, \ldots, r_0) = (r_{w-1}, \ldots, r_j \oplus (r_i \wedge f(x_1, \ldots, x_n)), \ldots, r_i, \ldots, r_0).$$

Bei einer nicht akzeptierenden Rechnung, $f(x_1, \ldots, x_n) = 0$, ist

$$\sigma_{\text{reject}}(r_{w-1}, \ldots, r_j, \ldots, r_i, \ldots, r_0) = (r_{w-1}, \ldots, r_j, \ldots, r_i, \ldots, r_0),$$

d. h. $\sigma_{\text{reject}} = id$, bei einer akzeptierenden Rechnung, $f(x_1, \ldots, x_n) = 1$, ist

$$\sigma_{\text{accept}}(r_{w-1}, \ldots, r_j, \ldots, r_i, \ldots, r_0) = (r_{w-1}, \ldots, r_j \oplus r_i, \ldots, r_i, \ldots, r_0).$$

Es ist $\sigma_{\text{accept}} \neq \sigma_{\text{reject}}$. Wir wählen $\tau = \sigma_{\text{accept}}$. Jede Variablenbelegung $a \in f^{-1}(1)$ führt zu einer akzeptierenden Rechnung des LBSPs und wählt gleichzeitig in unserem Permutations-Branchingprogramm eine Hintereinanderschaltung von Permutationen aus, die zusammen τ ergeben. Andersherum akzeptiert unser Permutations-Branchingprogramm auch nur dann, wenn es für die erzeugte Permutation eine Variablenbelegung gibt, für die auch das LBSP akzeptiert, weil es dessen Rechnung simuliert. □

Lemma 4.20. *Jedes Permutations-Branchingprogramm mit Breite w und Länge ℓ, das für ein $\tau \in S_w$, $\tau \neq id$, eine Funktion $f \in B_n$ berechnet, kann durch ein geschichtetes Branchingprogramm mit Tiefe ℓ und Breite höchstens w simuliert werden.*

Beweis. Sei $\sigma(x) \in \{id, \tau\}$ die Permutation, die ein gegebenes Permutations-Branchingprogramm bei Eingabe $x \in \{0,1\}^n$ berechnet. Da $\tau \neq id$, gibt es ein $k \in \{1, \ldots, w\}$, sodass $\tau(k) \neq k$. Es genügt also zu entscheiden, ob ein Berechnungspfad, der an der k-ten Quelle des Permutations-Branchingprogramms startet,

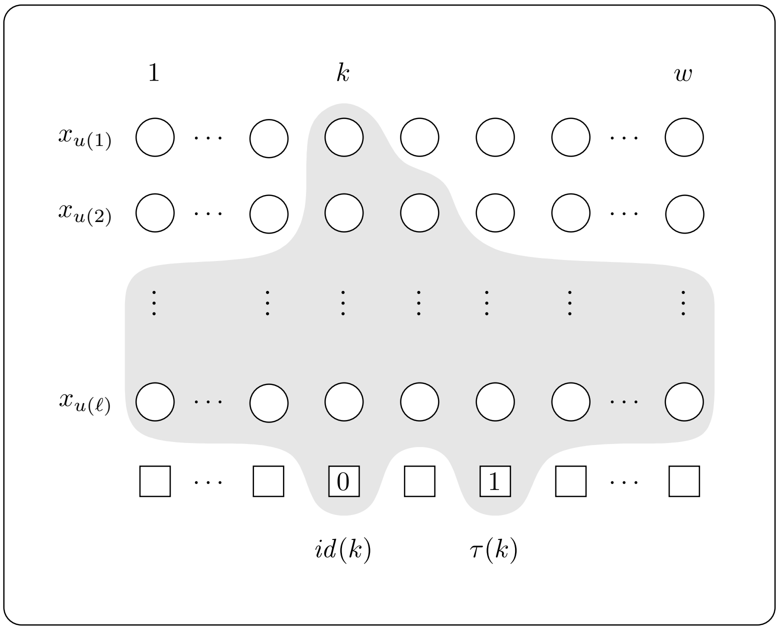


Abbildung 4.7: Branchingprogramm aus Permutations-Branchingprogramm.

bei Eingabe x die $\tau(k)$-te Senke oder die k-te Senke erreicht (Abbildung 4.7). Die k-te Quelle des Permutations-Branchingprogramms wird zur Quelle des Branchingprogramms. Die k-te Senke des Permutations-Branchingprogramms wird mit 0 markiert, die $\tau(k)$-te Senke mit 1. Alle übrigen Senken sind zwar auf Berechnungspfaden nicht erreichbar, aber es kann inkonsistente Pfade geben, auf denen man sie von der Quelle des Branchingprogramms erreichen kann. Wir verschmelzen diese unmarkierten Senken beliebig mit der 0- oder 1-Senke. Zum Schluss eliminieren wir alle inneren Knoten, die von der Quelle des Branchingprogramms nicht erreichbar sind und erhalten ein geschichtetes Branchingprogramm mit Tiefe ℓ und Breite höchstens w.

□

Branchingprogrammgröße für Formeln

Satz 4.21. *Jede boolesche Funktion $f \in B_n$, die durch eine B_2-Formel mit Größe ℓ berechnet wird, kann für jedes $k \in \mathbb{N}$ und jedes $c \geq 2$ durch ein geschichtetes Branchingprogramm mit*

$$\begin{aligned} \textit{Tiefe} &\leq 2^{\beta \log 3 + 2} \cdot \ell^{\beta}, \\ \textit{Breite} &\leq 2^{2^k + 2} \qquad \textit{und daher} \\ \textit{Größe} &\leq 2^{2^k + \beta \log 3 + 4} \cdot \ell^{\beta} \end{aligned}$$

für $\beta = 1 + \frac{1}{1+\log(c-1)} + 6\frac{c}{k}\ln 2$ *berechnet werden.*

Beweis. Wir wenden nacheinander Korollar 4.15, Lemma 4.19 und Lemma 4.20 an.

$$\begin{tikzcd}
B_2\text{-Formel: Größe} = \ell \arrow[r, "\text{Satz 4.21}"] \arrow[d, "\text{Korollar 4.15}"'] & \text{BP: } \begin{array}{c} \text{Tiefe} \le 2^{\beta \log 3+2} \cdot \ell^{\beta} \\ \text{Breite} \le 2^{2^k+2} \end{array} \\
\text{LBSP: } \begin{array}{c} \#\text{Var.-Ref.} \le 2^{\beta \log 3+2} \cdot \ell^{\beta} \\ \text{Breite} \le 2^k+2 \end{array} \arrow[r, "\text{Lemma 4.19}"'] & \text{PBP: } \begin{array}{c} \text{Länge} \le 2^{\beta \log 3+2} \cdot \ell^{\beta} \\ \text{Breite} \le 2^{2^k+2} \end{array} \arrow[u, "\text{Lemma 4.20}"']
\end{tikzcd}$$

$\square$

In Satz 4.21 können $k \in \mathbb{N}$ und $c \geq 2$ frei gewählt werden. Wenn wir z. B. $k := \lceil c^2 \rceil$ wählen, erhalten wir ein β das nur von c abhängt. Es ist dann

$$\lim_{c \to \infty} \beta(c) = \lim_{c \to \infty} \left(1 + \frac{1}{1 + \log(c-1)} + 6 \frac{c}{\lceil c^2 \rceil} \cdot \ln 2 \right) = 1.$$

Korollar 4.22. *Jede boolesche Funktion $f \in B_n$, die durch eine B_2-Formel mit Größe $L(f)$ berechnet wird, kann für jedes $\varepsilon > 0$ durch ein geschichtetes Branchingprogramm mit Größe $O(L(f)^{1+\varepsilon})$ berechnet werden.*

Damit haben wir die Branchingprogrammgröße für Funktionen mit Formelgröße $\mathrm{L}(f)$ von unten durch $\Omega(\mathrm{L}(f))$ und von oben durch $O(\mathrm{L}(f)^{1+\varepsilon})$ eingegrenzt. Das Problem, die maximal notwendige Branchingprogrammgröße für Formeln zu bestimmen, bleibt dennoch offen. So ist z. B. $\mathrm{BP}(f) = \Theta(\mathrm{L}(f) \log \mathrm{L}(f))$ weiterhin möglich.

5 Bewertung der neuen Schranke

Wir vergleichen in diesem Kapitel das neue Resultat aus Kapitel 4 mit dem bisher besten Resultat. Welches ist die kleinste Formelgröße $\ell = \mathrm{L}(f)$, sodass die neue Schranke für die Branchingprogrammgröße aus Satz 4.21

$$\mathrm{BP}(f) \leq 2^{2^k+\beta\log 3+4} \cdot \ell^\beta, \quad \beta = 1 + \frac{1}{1+\log(c-1)} + 6\frac{c}{k}\ln 2 \tag{5.1}$$

kleiner als die Schranke von Sauerhoff, Wegener und Werchner

$$\mathrm{BP}(f) \leq 1{,}360 \cdot \ell^\delta, \quad \delta = \log_4(3+\sqrt{5}) < 1{,}195 \tag{5.2}$$

wird? Wir suchen die Formelgröße ℓ_0^* als das minimale $\ell_0 \in \mathbb{N}$, für das gilt:

$$\forall \ell \geq \ell_0 \;\; \exists c \geq 2 \;\; \exists k \in \mathbb{N}\colon 2^{2^k+\beta\log 3+4} \cdot \ell^\beta < 1{,}360 \cdot \ell^\delta. \tag{5.3}$$

Für alle c und für alle k ist $2^{2^k+\beta\log 3+4} > 16 > 1{,}360$. Daher können nur solche Werte für c und k die Bedingung erfüllen, für die $\ell^\beta < \ell^\delta$ und somit $\beta < \delta$ ist. Wir betrachten also ab jetzt nur noch solche β, für die gilt

$$1 < \beta < \delta. \tag{5.4}$$

Die Bedingung (5.3) lässt sich, da $\ell \geq 1$, mit Hilfe der Potenzgesetze wie folgt umformen.

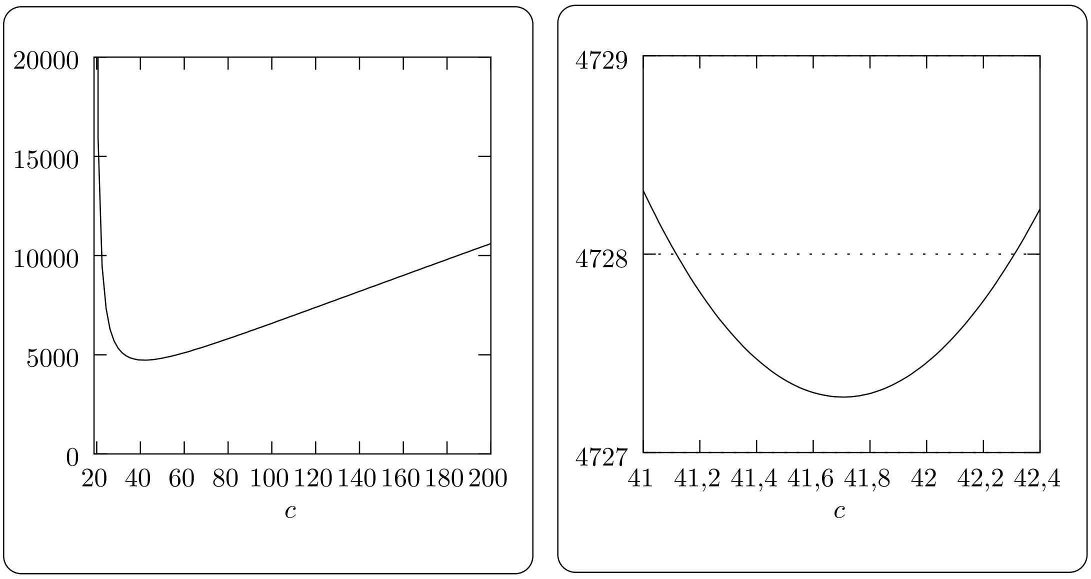
$$\begin{aligned} \frac{\ell^\delta}{\ell^\beta} &> \frac{2^{2^k+\beta\log 3+4}}{1{,}360} \\ \ell^{\delta-\beta} &> 2^{2^k+\beta\log 3+4-\log 1{,}360} \end{aligned}$$

Wegen Ungleichung (5.4) gilt $\delta - \beta > 0$. Exponenzieren mit $\frac{1}{\delta-\beta}$ liefert

$$\ell > 2^{\frac{2^k+\beta\log 3+4-\log 1{,}360}{\delta-\beta}}. \tag{5.5}$$

Wenn wir in die rechte Seite von Ungleichung (5.5) Werte für c und k einsetzen, die die Nebenbedingungen erfüllen, erhalten wir ein ℓ_0, das eine obere Schranke für das gesuchte minimale ℓ_0^* ist. Zuvor schränken wir den zu untersuchenden Wertebereich für c und k mit Hilfe der Nebenbedingung $1 \leq \beta \leq \delta$ geeignet ein. In

$$\beta = 1 + \tfrac{1}{1+\log(c-1)} + 6\tfrac{c}{k}\ln 2 < \delta \tag{5.6}$$



(a) (b)

Abbildung 5.1: $g(c)$.

kommen ausschließlich positive Terme vor. Wenn wir den Term $6\frac{c}{k}\ln 2$ in der letzten Ungleichung streichen, wird die linke Seite der Ungleichung nur kleiner. Daher gilt auch

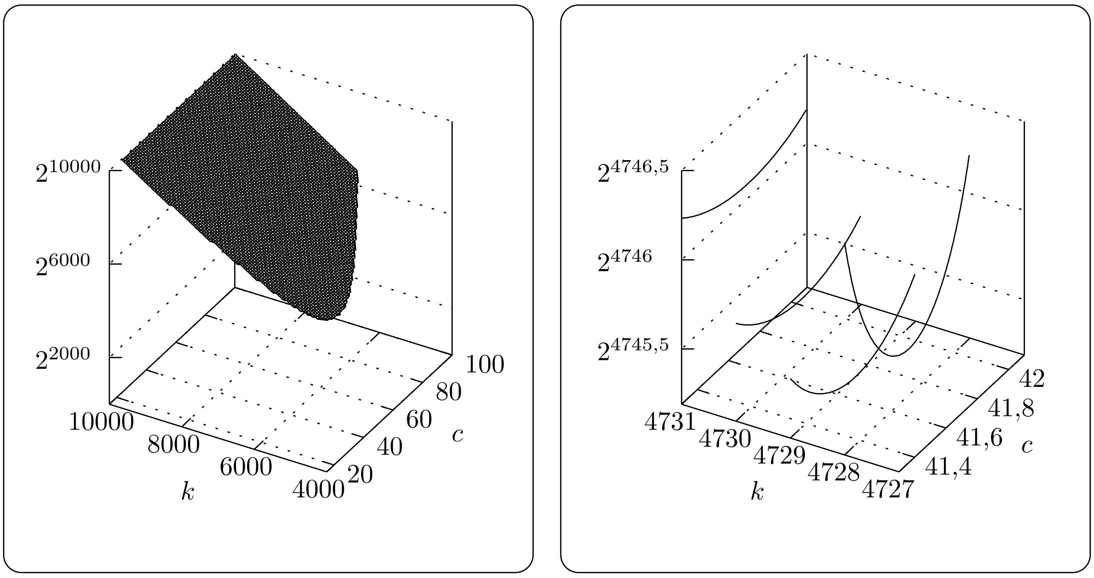
$$\begin{aligned}\frac{1}{1+\log(c-1)} &< \delta - 1\\ \log(c-1) &> \frac{1}{\delta-1} - 1\\ c &> 2^{\frac{1}{\delta-1}-1} + 1\end{aligned}$$

und $c_{\min} = 2^{\frac{1}{\delta-1}-1} + 1 > 18{,}731$ ist eine untere Schranke für c. Für eine untere Schranke für k lösen wir Ungleichung (5.6) nach k auf:

$$k > \frac{6c\ln 2}{\delta - 1 - \frac{1}{1+\log(c-1)}} =: g(c) \tag{5.7}$$

Für $c > c_{\min}$ ist der Nenner in Ungleichung (5.7) größer als 0 und somit $g(c)$ positiv. Abbildung 5.1(a) zeigt den Graphen von $g(c)$. Das kleinste $k \in \mathbb{N}$, sodass es ein $c > c_{\min}$ gibt, welches Ungleichung (5.7) erfüllt, ist $k_{\min} = 4728$ (Abbildung 5.1(b)). Wenn wir zu diesem $k = k_{\min}$ etwa $c = 41{,}7$ wählen, ist $\beta < \delta$ und wir können durch Einsetzen in Ungleichung (5.5) ℓ_0 berechnen und erhalten so eine erste Abschätzung für l_0^*:

$$\ell_0^* \le \ell_0 \le 2^{2^{4745{,}450}}.$$



(a) (b)

Abbildung 5.2: $f(c, k)$.

Um ℓ_0^* zu bestimmen, müssen wir die rechte Seite von Ungleichung (5.5) minimieren. Dazu genügt es, den Exponenten für $c > c_{\min}$ und $k \geq k_{\min}$ zu minimieren:

$$f(c, k) := \frac{2^k + \beta \log 3 + 4 - \log 1{,}360}{\delta - \beta} \to \min, \qquad 1 < \beta < \delta.$$

Die gängigen Methoden der Analysis sind leider nicht unmittelbar anwendbar. Die Funktion $f(c, k)$ ist zwar in c differenzierbar aber in k nicht stetig, da $k \in \mathbb{N}$. Für die Relaxation des Problems mit reellwertigem k scheint es zudem nicht möglich, die Nullstellen der partiellen Ableitungen nach c oder k als geschlossene Formel zu bestimmen. Der Graph von $f(c, k)$ gibt aber Aufschluss über das vermutlich eindeutig bestimmte Minimum. Abbildung 5.2(a) zeigt einen Ausschnitt des Graphen von $f(c, k)$. Es sind nur Punkte eingetragen, an denen $f(c, k)$ definiert ist, d. h. $1 < \beta < \delta$ erfüllt ist. Auf der nach oben gerichteten Achse sind die Funktionswerte auf einer logarithmischen Skala aufgetragen. (Auf einer linearen Skala ist der Graph nicht nur in Richtung der k-Achse, sondern auch in Richtung der c-Achse so stark nach oben gekrümmt, dass er sich nur noch auf sehr kleinen Ausschnitten sinnvoll darstellen lässt.) Trotzdem kam man in Abbildung 5.2(a) ein Minimum erahnen, das auf dem „Rand“ für $k = k_{\min} = 4728$ zu liegen scheint. Die nähere Ansicht in Abbildung 5.2(b) zeigt, dass dieses Minimum in Wirklichkeit nicht auf dem „Rand“ liegt, sondern erst für $k = 4729$ und c etwa bei 41,7 angenommen wird. Mit der Newtonmethode lässt sich $c \approx 41{,}715$ für dieses Minimum genauer ermitteln. Mit diesen Werten für c und k erhält man ein ℓ_0', das eine bessere obere Schranke für ℓ_0^* liefert:

$$\ell_0^* \leq \ell_0' \leq 2^{2^{4745,193}}.$$

Diese Zahl ist leider viel zu groß, als dass Formeln dieser Größenordnung heute (und wohl auch in absehbarer Zukunft) in Rechnern explizit gespeichert werden könnten.[1] Dies gilt erst recht für die mit der neuen Methode konstruierten Branchingprogramme. Wir haben hier zwar keine unteren Schranken für das Verfahren gezeigt, aber die obere Schranke in Ungleichung (5.1) enthält schon für $k = k_{\min} = 4728$ eine multiplikative Konstante, die größer als $2^{2^{4728}}$ ist, und kann uns nicht ermutigen entsprechende Branchingprogramme wirklich zu konstruieren. Daher ist das Resultat aus Korollar 4.22 eher von theoretischer Bedeutung. Dagegen kann das Konstruktionsverfahren von Sauerhoff, Wegener und Werchner auch praktisch eingesetzt werden, da es neben seinen kleinen Konstanten in der zugehörigen Schranke (Ungleichung (5.2)) auch sehr einfach und effizient umgesetzt werden kann.

---

[1] Physiker schätzen die Anzahl der Atome im Universum auf etwa $10^{79} < 2^{2^{8,036}}$.

Abkürzungen und mathematische Symbole

$\wedge$	logische Konjunktion, AND
$\vee$	logische Disjunktion, OR
$\neg$	logische Negation, NOT
$\oplus$	logische Parität, EXOR
$\oplus$-BP	Parity-Branchingprogramm (→Definition 4.8)
B_2	Menge aller booleschen Funktionen $f\colon \{0,1\}^2 \to \{0,1\}$ (→Tabelle 1.1)
B_2^*	Teilmenge von B_2 (→Tabelle 1.1)
BP	Branchingprogramm (→Definitionen 1.9 und 1.10)
$\mathrm{BP}(F)$, $\mathrm{BP}(f)$	Branchingprogrammgröße (→Definition 1.12)
$\mathrm{C}(S)$, $\mathrm{C}(f)$	Schaltkreisgröße (→Definition 1.3)
$\mathrm{D}(S)$, $\mathrm{D}(f)$	Schaltkreistiefe (→Definition 1.3)
id	identische Abbildung
ite	$ite(x,y,z) := xy \vee \overline{x}z$ ("if x then y else z")
$\mathrm{L}(F)$, $\mathrm{L}(f)$	B_2-Formelgröße (→Definition 1.6)
$\mathrm{L}^*(F)$, $\mathrm{L}^*(f)$	$\{\wedge,\vee,\neg\}$-Formelgröße (→Definition 1.6)
LBSP	linear bijection straight-line Programm (→Definition 4.10)
ln	Logarithmus zur Basis e
log	Logarithmus zur Basis 2
$\mathbb{N}$	Menge der natürlichen Zahlen
PBP	Permutations-Branchingprogramm (→Definition 4.16)
S_n	Symmetrische Gruppe bestehend aus allen Permutationen der Elemente von $\{1,\ldots,n\}$
U_2	Teilmenge von B_2 (→Tabelle 1.1)
$\mathbb{Z}_2$	Ring über $\{0,1\}$ mit $\oplus$ als additiver und $\wedge$ als multiplikativer Verknüpfung

Bibliographie

Barrington, D. A. (1989). Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. Journal of Computer and System Sciences 38, 150–164.

Bonet, M. L. und Buss, S. R. (1994). Size-depth tradeoffs for boolean formulae. Information Processing Letters 49, 151–155.

Brent, R. P. (1974). The parallel evaluation of general arithmetic expressions. Journal of the ACM 21, 201–206.

Cleve, R. (1991). Towards optimal simulations of formulas by bounded-width programs. Computational Complexity 1, 91–105.

Karchmer, M. und Wigderson, A. (1993). On span programs. In *Proc. of the 8th Annual Conference on Structure in Complexity Theory (SCTC).* S. 102–111.

Pratt, V. R. (1975). The effect of basis on size of boolean expressions. In *Proc. of the 16th IEEE Symp. on Foundations of Computer Science (FOCS).* S. 119–121.

Sauerhoff, M., Wegener, I. und Werchner, R. (1999). Relating branching program size and formula size over the full binary basis. In *Proc. of the 16th Symp. on Theoretical Aspects of Computer Science (STACS)*, LNCS 1563. S. 57–67.

Wegener, I. (1987). *The Complexity of Boolean Functions.* Wiley-Teubner.

Wegener, I. (1989). *Effiziente Algorithmen für grundlegende Funktionen.* B. G. Teubner.

Wegener, I. (1993). Theorie des Logikentwurfs. Vorlesungsskript, Universität Dortmund.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams – Theory and Applications.* Monographs on Discrete and Applied Mathematics. SIAM.