Über die Komplexität der Multiplikation in eingeschränkten Branchingprogrammmodellen

Philipp Wölfel

Universität Dortmund, Lehrstuhl Informatik 2
philipp.woelfel@cs.uni-dortmund.de

Wie schwer ist es zu multiplizieren? Jeder hat in der Schule die Methode zum „schriftlichen“ Multiplizieren – die sog. „Schulmethode der Multiplikation“ – kennen gelernt. Unsere Schulkenntnisse reichen also aus, einen effizienten Algorithmus zum Multiplizieren von langen Zahlen anzugeben. Um zwei n-stellige Zahlen zu multiplizieren, benötigt man mit der Schulmethode $\Theta(n^2)$ Additionen und Multiplikationen einzelner Ziffern. Die Schulmethode ist recht einfach, aber bei weitem nicht optimal. Das asymptotisch schnellste bekannte Verfahren zur Multiplikation n-stelliger ganzer Zahlen stammt von von Schönhage und Strassen [SS71] und hat einen Zeitbedarf von $O(n \log n \log\log n)$. Aber ist dieser Algorithmus schon optimal?

In der Informatik besteht ein großes Interesse, die Komplexität von fundamentalen arithmetischen Funktionen zu bestimmen. Schließlich kommen sie in fast jedem Algorithmus auf die ein oder andere Weise vor und jeder Prozessor enthält Befehle zum Addieren, Subtrahieren, Multiplizieren und Dividieren. Während man bei oberen Schranken (z. B. für die Schaltkreisgröße und -tiefe) relativ gute Ergebnisse erzielen kann, ist man mit den heutigen Beweistechniken noch nicht in der Lage, z. B. im allgemeinen Schaltkreismodell vernünftige untere Schranken zu zeigen. So kennt man für keine explizit definierte boolesche Funktion in n Variablen – also auch nicht für Multiplizierer – eine bessere untere Schranke als $5n$ für die Schaltkreisgröße. Dieses Unvermögen, für Schaltkreise und andere allgemeine Rechenmodelle vernünftige untere Schranken für die Komplexität explizit definierter Funktionen zu zeigen, steht im Kontrast zu der bereits 1949 von Shannon bewiesenen Tatsache, dass die meisten booleschen Funktionen nicht durch Schaltkreise polynomieller Größe darstellbar sein können (weil es wesentlich mehr Funktionen als nicht äquivalente Schaltkreise polynomieller Größe gibt).

1 Branchingprogramme

In der Komplexitätstheorie unterscheidet man zwischen *uniformen* Rechenmodellen, die Eingaben beliebiger Länge erhalten können, und *nichtuniformen* Modellen, bei denen die Eingabelänge fest vorgegeben ist. Uniforme Rechenmodelle – die wichtigsten Beispiele sind Turing- und Registermaschinen – modellieren daher die Softwareebene, während

nichtuniforme Modelle der Hardwareebene entsprechen. Die meisten nichtuniformen Rechenmodelle (wie z. B. Schaltkreise) stellen eine *boolesche Funktion* $f\colon \{0,1\}^n \to \{0,1\}$ dar. Der Schaltkreis ist wegen des starken Bezugs zur Hardware sicherlich das bedeutendste nichtuniforme Rechenmodell und eines der wichtigsten Probleme der theoretischen Informatik besteht darin, Techniken zum Beweis vernünftiger unterer Schranken für so allgemeine nichtuniforme Modelle wie Schaltkreise zu entwickeln.

Unter anderem mit dem Ziel, den Beweis unterer Schranken voranzutreiben, aber auch als Datenstruktur für boolesche Funktionen wurden andere nichtuniforme Modelle intensiv untersucht. Zu den bedeutendsten gehören *Branchingprogramme*, auch *Binary Decision Diagrams* (kurz BDDs) genannt. Ein Branchingprogramm ist ein spezielles Entscheidungsdiagramm zur Darstellung einer booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$. Formal handelt es sich dabei um einen gerichteten azyklischen Graphen mit einem ausgezeichneten Startknoten (*Wurzel*), dessen innere Knoten mit Variablen $x_1, \ldots, x_n$ beschriftet sind. Jeder innere Knoten hat zwei ausgehende Kanten, eine *0-Kante* und eine *1-Kante*. So definiert jede Eingabe $(a_1, \ldots, a_n) \in \{0,1\}^n$ für die Funktion f einen *Berechnungspfad* im Branchingprogramm, der an der Wurzel beginnt und von einem mit einer Variablen x_i markierten Knoten über die mit a_i markierte Kante führt. Die Ausgabe des Branchingprogramms wird durch zwei Knoten ohne ausgehende Kanten, den *Senken* realisiert. Eine Senke ist mit 0 und eine Senke ist mit 1 beschriftet. Die Beschriftung derjenigen Senke, die auf dem Berechnungspfad einer Eingabe $(a_1, \ldots, a_n)$ erreicht wird, gibt den Funktionswert $f(a_1, \ldots, a_n)$ an.

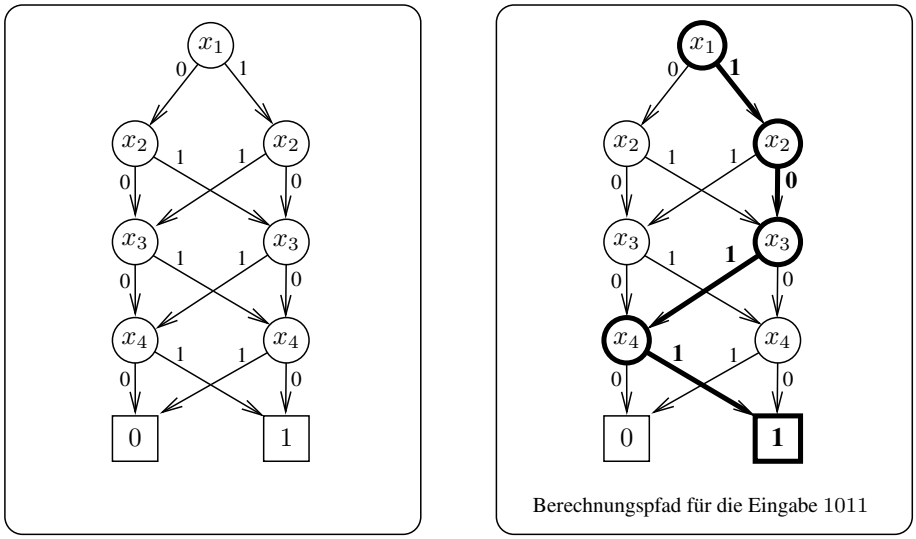

Abbildung 1: Branchingprogramm für die Abbildung $x_1x_2x_3x_4 \mapsto (x_1 + x_2 + x_3 + x_4) \bmod 2$

Das wichtigste Komplexitätsmaß bei Branchingprogrammen ist deren *Größe*, d. h. die Anzahl der Knoten im Graphen. Mit *Branchingprogrammgröße* einer Funktion f wird die Größe eines minimalen Branchingprogramms bezeichnet, das f darstellt. Seine Bedeutung erlangt dieses Komplexitätsmaß unter anderem dadurch, dass es ein Maß für den

Speicherplatzbedarf für die Berechnung der Funktion f in anderen Rechenmodellen darstellt. So kann man z. B. Turingmaschinen für Eingaben der Länge n, die Speicherplatz $s(n) \geq \log n$ benötigen, durch Branchingprogramme der Größenordnung $2^{s(n)}$ simulieren [Co66]. Umgekehrt ist der Logarithmus der Branchingprogrammgröße auch eine untere Schranke für den Platzbedarf einer nichtuniformen Turingmaschine. Auch zwischen Schaltkreiskomplexität und Branchingprogrammkomplexität gibt es einen engen Zusammenhang: Die Größe eines minimalen Branchingprogramms für eine boolesche Funktion liegt zwischen der Schaltkreisgröße und der Formelgröße (also der Größe des minimalen Schaltkreises mit Fan-out 1) der Funktion (vgl. [We87], S. 414 ff.).

Zwar konnten bis heute auch keine großen unteren Schranken für die Größe von allgemeinen Branchingprogrammen für explizit definierte Funktionen gezeigt werden, aber es gibt zahlreiche Möglichkeiten, die Mächtigkeit von Branchingprogrammen auf natürliche Weise einzuschränken. Die typischen Einschränkungen betreffen dabei entweder die Anzahl von Variablenzugriffen auf Berechnungspfaden (z. B. darf beim *Read-k-Branching-programm* auf jedem Berechnungspfad jede Variable höchstens k-mal vorkommen) oder die Reihenfolge, in der auf Variablen zugegriffen werden darf.

Sehr eingeschränkte Branchingprogrammmodelle erlangen eine besondere Bedeutung dadurch, dass sie als Datenstruktur für boolesche Funktionen verwendet werden können. Zu den wichtigsten Anwendungsbereichen solcher Branchingprogramme gehören das Model-Checking und die Verifikation von Schaltkreisen. Daher haben komplexitätstheoretische Untersuchungen für stark eingeschränkte Branchingprogramme eine ganz praktische Bedeutung. Sie können aufzeigen, für welche Probleme die Datenstruktur geeignet ist und wo ihre Grenzen liegen.

Viele der allgemeineren Branchingprogrammmodelle spielen als Datenstruktur für boolesche Funktionen keine wesentliche Rolle. Forschungsziel ist meist, die Mächtigkeit verschiedener Modelle zu vergleichen und Techniken zum Nachweis unterer Schranken für immer allgemeinere Modelle zu entwickeln. Wenn ein neues Berechnungsmodell untersucht wird, werden daher häufig zunächst untere Schranken für beliebige, aber explizit definierte Funktionen bewiesen. Oft werden für die ersten Beweise dann Funktionen gerade so definiert, dass sie gut zur Beweistechnik passen. Auch wenn dieses Vorgehen wichtig ist, um erst einmal Beweistechniken zu entwickeln, darf man nicht vergessen, dass solche künstlich definierten Funktionen nicht an sich von komplexitätstheoretischem Interesse sind. Stattdessen muss am Ende immer das Ziel stehen, mit den gelernten Techniken untere Schranken für wichtige und bedeutende Funktionen zu beweisen, deren Komplexität uns wirklich interessiert. Wenn man eine bestimmte ausgewählte Funktion betrachtet, die ihre Bedeutung unabhängig vom betrachteten Berechnungsmodell erlangt, so wird meist der Nachweis unterer oder oberer Schranken schwieriger sein, als wenn man die Funktion im Hinblick auf bekannte Beweistechniken geeignet definieren kann. So muss man im Fall einer vorgegebenen Funktion entweder die Eigenschaften, die man zur Anwendung einer bekannten Beweistechnik benötigt, erst nachweisen oder es müssen neue Beweistechniken gefunden bzw. bekannte Beweistechniken verändert werden, sodass sie auf die entsprechenden Eigenschaften der Funktion anwendbar sind. Während Ersteres dabei meist zu einem umfassenderen Verständnis der Funktion führt, kann Letzteres allgemeinere Beweistechniken oder sogar ein besseres Verständnis des Berechnungsmodells bewirken.

Abgesehen davon haben einige Funktionen, wie z. B. die fundamentalen arithmetischen Operationen, eine so große Bedeutung (nicht nur für die Informatik), dass es eines der vorrangigen wissenschaftlichen Ziele sein muss, möglichst viel über sie zu erfahren.

2 Das mittlere Bit der Multiplikation

Zu den wichtigsten Funktionen gehören ohne Frage die fundamentalen arithmetischen Operationen. Addition und Subtraktion sind zu „einfach“, als dass weitere komplexitätstheoretische Untersuchungen in Branchingprogrammmodellen lohnenswert erscheinen – schließlich haben schon sehr eingeschränkte Branchingprogramme für Addition oder Subtraktion nur lineare Größe. Anders ist die Situation bei der Multiplikation bzw. bei der booleschen Funktion, die ein (geeignet gewähltes) Ausgabebit des Produkts zweier Binärzahlen darstellt. Sie wird im Allgemeinen für „schwierig“ gehalten, aber es gibt bisher nur für sehr eingeschränkte Branchingprogrammmodelle untere Schranken und selbst diese waren bisher noch nicht zufrieden stellend.

Insbesondere die Multiplikation und die Division sollten also Schwerpunkt komplexitätstheoretischer Untersuchungen sein. Hier beschränken wir uns auf die Multiplikation; teilweise haben die aufgezeigten unteren Schranken aber auch Auswirkungen auf die Komplexität der Division: Mithilfe sog. *Read-Once-Projektionen* kann man nachweisen (vgl. [We93]), dass in den meisten Branchingprogrammmodellen – und bei allen in dieser Arbeit betrachteten – eine exponentielle untere Schranke für die Komplexität der Multiplikation auch eine exponentielle untere Schranke für die Komplexität der Division impliziert (wenn auch mit erheblich schwächeren Konstanten im Exponenten).

Da Branchingprogramme normalerweise ein Modell zur Berechnung von Funktionen mit einem Ausgabebit darstellen, beschränken wir uns hier auf die Komplexität einzelner Ausgabebits der Multiplikation. Formal betrachten wir also die booleschen Funktionen

$$\mathrm{MUL}_{i,n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}, \quad (x_{n-1} \dots x_0,\, y_{n-1} \dots y_0) \mapsto z_i,$$

wobei $z_{2n-1} \dots z_0$ die Binärdarstellung des Produkts der Binärzahlen $x_{n-1} \dots x_0$ und $y_{n-1} \dots y_0$ ist. Es wird allgemein angenommen, dass das *mittlere Bit* der Multiplikation, also die Funktion $\mathrm{MUL}_{n-1,n}$, das am schwierigsten zu berechnende Ausgabebit ist. Tatsächlich kann man ein beliebiges Ausgabebit der n-Bit-Multiplikation dadurch erhalten, dass man die zu multiplizierenden Zahlen in zwei k-Bit-Zahlen, $k \leq 2n$, einbettet und von diesen Zahlen das mittlere Bit des Produkts berechnet. Wir beschränken uns daher darauf, die Komplexität des mittleren Bits der Multiplikation in verschiedenen Branchingprogrammmodellen zu untersuchen.

3 Verifizierbarkeit von Multiplizierern mit OBDDs

Der Name OBDD steht für *Ordered Binary Decision Diagram* und bezeichnet ein von Bryant 1985 [Br85] eingeführtes Branchingprogrammmodell, dessen Struktur zwei we-

sentlichen Einschränkungen unterliegt: Einerseits dürfen Variablen auf jedem Berechnungspfad eines OBDDs nur einmal vorkommen und andererseits ist die Reihenfolge, in der die Variablen auf den Berechnungspfaden vorkommen, durch eine feste *Variablenordnung* vorgegeben. Beim Branchingprogramm aus Abbildung 1 handelt es sich z. B. um ein OBDD mit der Variablenordnung (x_1, x_2, x_3, x_4).

Diese Einschränkungen führen dazu, dass sich OBDDs als Datenstruktur für boolesche Funktionen besonders eignen, da sich auf ihnen zahlreiche wichtige Operationen effizient durchführen lassen. Zu den wichtigsten Beispielen gehören die Syntheseoperation, die aus den OBDDs für zwei boolesche Funktionen f, g ein OBDD für die boolesche Funktion $f \odot g$ berechnet, wobei $\odot$ ein beliebiger binärer Operator ist, die Minimierung eines OBDDs, d. h. die Berechnung eines OBDDs minimaler Größe für die Funktion f gegeben als OBDD, oder der Erfüllbarkeitstest, der zu einer Funktion f, gegeben als OBDD, entscheidet, ob es eine Eingabe x mit $f(x) = 1$ gibt.

Für die Anwendung bei der Verifikation von Schaltkreisen kann man z. B. ausnutzen, dass eine Folge dieser drei Operationen einen Äquivalenztest liefert, der entscheidet, ob zwei Schaltkreise die gleiche Funktion darstellen. Jede einzelne für so eine Verifikation notwendige OBDD-Operation benötigt im Worst-Case nur quadratische Zeit bezüglich des größten beteiligten OBDDs. Trotzdem kann die Gesamtlaufzeit dieser Methode sehr schlecht sein, wenn zu große OBDDs entstehen. Mittlerweile wurden zahlreiche Techniken und Heuristiken entwickelt, um das Entstehen von großen OBDDs während der Syntheseoperationen zu verhindern und einfache Schaltkreise wie Addierer können mit OBDDs effizient verifiziert werden. Solche Methoden müssen jedoch versagen, wenn schon das OBDD für die Spezifikation zu groß ist. So konnte z. B. trotz zahlreicher Versuche erst vor wenigen Jahren für den gesamten 16-Bit-Multiplikationsschaltkreis $c6288$, einen der bekanntesten ISCAS-(International Symposium on Circuits and Systems)-Benchmark-Schaltkreise, ein OBDD berechnet werden [YCBO98]. Es sei angemerkt, dass das resultierende OBDD mithilfe mehrerer Wurzeln alle Ausgabebits des Multiplizierers darstellte. Es bestand aus mehr als 40 Millionen Knoten, das größte OBDD, das während der Syntheseoperationen entstand, hatte sogar mehr als 110 Millionen Knoten und der maximale Speicherplatzbedarf betrug 3803 Megabyte. Soweit dem Autor bekannt ist, konnte bis heute aus keinem Multiplikationsschaltkreis mit mehr als 16 Bit ein entsprechendes OBDD berechnet werden.

Natürlich bedeuten diese Erfahrungswerte noch nicht, dass es nicht auch kleine OBDDs für 16-Bit- oder sogar 32-Bit-Multiplizierer geben kann. Die Größe des minimalen OBDDs für eine Funktion hängt stark von der gewählten Variablenordnung ab – tatsächlich gibt es viele Funktionen, die bei geeigneter Wahl der Variablenordnung durch polynomiell große OBDDs dargestellt werden können, während bei schlecht gewählter Variablenordnung die OBDD-Darstellung exponentiell groß wird (mehrere Beispiele finden sich in der Monographie [We00]). So könnte es möglich sein, dass bei den Versuchen, OBDDs für Multiplizierer zu berechnen, immer nur die falschen Variablenordnungen gewählt bzw. heuristisch gefunden wurden.

Wenn wir aber beweisen können, dass es in Wirklichkeit gar keine kleinen OBDDs für Multiplizierer gibt, kann man auf weitere Versuche, OBDDs zur Verifikation einzusetzen, verzichten und sich auf die Suche nach alternativen Lösungen konzentrieren. Wenn man

die Gründe versteht, aus denen so eine Datenstruktur für die Darstellung und Verifikation wichtiger Funktionen ungeeignet ist, so kann dies auch dabei helfen, bessere Datenstrukturen und Methoden zu entwickeln.

Bryant hat bereits 1986 gezeigt, dass die Darstellung der Multiplikation durch OBDDs problematisch ist, indem er bewies, dass die Darstellung der Funktion $\mathrm{MUL}_{n-1,n}$ OBDDs mit mindestens $2^{n/8}$ Knoten erfordert [Br86]. Diese exponentielle untere Schranke impliziert, dass die OBDD-Größe für n-Bit-Multiplizierer sehr stark in n wächst, also dass die OBDD-Verifikation für Multiplizierer theoretisch schnell an ihre Grenzen stößt. Allerdings ist dieses Ergebnis für die heute typischen Multiplikationsschaltkreise in der Praxis kaum von Relevanz. Denn selbst für 128-Bit-Multiplizierer impliziert es nur eine untere Schranke von 65 536 OBDD-Knoten. OBDDs dieser Größenordnung sind mit heutigen Mitteln noch sehr gut handhabbar.

Die oben geschilderten Erfahrungen aus der Praxis, nach denen selbst 16-Bit-Multiplizierer schon OBDDs mit mehr als 40 Millionen Knoten erforderten, ließen aber erwarten, dass die wahre OBDD-Größe der Multiplikation wesentlich größer ist. So äußern z. B. Bollig und Wegener (s. [BW99], S. 3) die Vermutung, dass jedes OBDD für $\mathrm{MUL}_{n-1,n}$ mindestens 2^n Knoten hat. Im Rahmen der Dissertation wurde nun erstmals folgende untere Schranke bewiesen, die für die Verifikation von Multiplizierern tatsächlich praxisrelevant ist.

Theorem 1 ([Wo01]) *Jedes OBDD, das die Funktion* $\mathrm{MUL}_{n-1,n}$ *darstellt, besteht aus mindestens* $2^{n/2}/61 - 4$ *Knoten.*

Somit ist die Vermutung, dass OBDDs zur Verifikation von Multiplizierern schon bei den heutzutage üblichen Wortbreiten kaum geeignet sind, erstmals auch theoretisch belegt: Für das mittlere Bit eines 64-Bit-Multiplizierers werden mehr als 70 Millionen Knoten benötigt – bei einem 128-Bit Multiplizierer sind es schon mehr als $3{\cdot}10^{17}$ Knoten. Der Versuch, 64-Bit-Multiplikationsschaltkreise mit OBDDs zu verifizieren, benötigt daher schon eine extrem große Menge an Ressourcen und für 128-Bit-Multiplizierer ist dies mit heutigen Mitteln unmöglich.

Obwohl die neue untere Schranke einen Fortschritt bei der Bestimmung der OBDD-Größe der Multiplikation darstellt, handelt es sich bei der Größenordnung von $2^{n/2}$ vermutlich nicht um die Wahrheit. Wie man am Unterschied zu Bryants unterer Schranke von $2^{n/8}$ sieht, kann der konstante Faktor im Exponenten schon bei kleinen Wortbreiten n entscheidende Auswirkungen haben. Möglichst gute Approximationen für die Wahrheit zu finden, bedeutet daher auch, obere Schranken für die OBDD-Größe der Multiplikation herzuleiten. Bisher wurde allerdings noch nicht versucht, für allgemeine Werte n OBDDs für die Funktion $\mathrm{MUL}_{n-1,n}$ zu konstruieren; allerdings ist bekannt, dass man jede boolesche Funktion in $2n$ Variablen mit einem OBDD der Größe $O(2^{2n}/n)$ darstellen kann (vgl. [We00]). In der Dissertation wird hingegen eine OBDD-Konstruktion für $\mathrm{MUL}_{n-1,n}$ vorgestellt, die mit $(7/3) \cdot 2^{4n/3}$ Knoten auskommt.

4 FBDDs

Free Binary Decision Diagrams, oder kurz FBDDs, sind Branchingprogramme, bei denen auf jedem Berechnungspfad jede Variable höchstens einmal vorkommen darf. Gegenüber OBDDs fällt also die Variablenordnung weg, die für alle Berechnungspfade die Reihenfolge der gelesenen Variablen vorschreibt. Da FBDDs wesentlich weniger restriktiv als OBDDs sind, kann man erwarten, dass die FBDD-Größe von vielen Funktionen kleiner als deren OBDD-Größe ist. So ist es nicht überraschend, dass es zahlreiche Beispiele für boolesche Funktionen mit exponentieller OBDD-Größe gibt, die durch polynomiell große FBDDs dargestellt werden können (vgl. [We00]). Eine größere Darstellungskraft einer Datenstruktur birgt jedoch meist den Nachteil, dass Operationen auf dieser Datenstruktur schwieriger durchzuführen sind. So sind für FBDDs viele Operationen nicht effizient realisierbar, die mit OBDDs in polynomieller Zeit durchgeführt werden können. Insbesondere wird z. B. vermutet, dass es keinen Algorithmus gibt, der in polynomieller Zeit testet, ob zwei FBDDs die gleiche Funktion darstellen. Prinzipiell ist jedoch auch mit FBDDs eine Schaltkreisverifikation nach dem im vorigen Abschnitt beschriebenem Muster möglich, wenn man fordert, dass alle in der Prozedur vorkommenden FBDDs gewisse Gemeinsamkeiten aufweisen [SW95, GM94].

Während die ersten exponentiellen unteren Schranken für FBDDs bereits seit den 80er Jahren bekannt sind (s. z. B. [We88]), konnte erst 1995 durch Ponzio eine schwach exponentielle untere Schranke von $2^{\Omega(\sqrt{n})}$ für die FBDD-Größe des mittleren Bits der Multiplikation nachgewiesen werden (vgl. [Po98]). Aus komplexitätstheoretischer Sicht war dieses Resultat bedeutend, da es zum ersten Mal eine superpolynomielle untere Schranke für die Funktion $\mathrm{MUL}_{n-1,n}$ in einem Branchingprogrammmodell aufzeigte, bei dem die Reihenfolge, in der die Variablen auf den Berechnungspfaden vorkommen dürfen, nicht eingeschränkt ist. Da aber auch für die FBDD-Größe eine Größenordnung von $2^{\Omega(n)}$ vermutet wurde, kann diese Schranke noch nicht als zufrieden stellend betrachtet werden. Aus praktischer Sicht ist erwähnenswert, dass die von Ponzio notierte untere Schranke keine Auskunft über die Konstanten im Exponenten gibt. Dies zeigt schon, dass man hier noch weit davon entfernt ist, für praktisch relevante Wortbreiten n belegen zu können, dass sich n-Bit-Multiplizierer nicht durch FBDDs darstellen lassen. Im Rahmen der Dissertation konnte nun – in Zusammenarbeit mit Beate Bollig – eine echt exponentielle untere Schranke für die FBDD-Größe der Multiplikation bewiesen werden.

Theorem 2 ([BW01]) *Jedes* FBDD*, das die Funktion* $\mathrm{MUL}_{n-1,n}$ *darstellt, hat eine Größe von mindestens* $2^{\lfloor (n-7)/4 \rfloor}$.

Erwähnt sei noch, dass die hier vorgestellten unteren Schranken für die OBDD- und FBDD-Größe der Multiplikation auf einer völlig neuen Beweisidee beruhen. Es wurde erstmals die Tatsache ausgenutzt, dass man mithilfe der Multiplikation sog. *universelle Hashklassen* konstruieren kann (vgl. [DHKP97, Wo99]). Dabei handelt es sich um Funktionsklassen, die in zahlreichen randomisierten Algorithmen verwendet werden, wenn es darum geht, eine Menge von Elementen möglichst zufällig mithilfe einer (Hash)funktion auf Tabellenplätze abzubilden. Interessant ist dabei, dass dieses algorithmische „Werkzeug“ hier eine eher unerwartete komplexitätstheoretische Anwendung findet.

5 Allgemeinere Branchingprogrammmodelle

Obwohl auch für allgemeinere Branchingprogrammmodelle als FBDDs (wie z. B. Read-k-Branchingprogramme, bei denen auf jedem Berechnungspfad jede Variable k-mal vorkommen darf) bereits seit langem Techniken zum Beweis unterer Schranken bekannt sind, konnte bisher in keinem solchen Modell eine exponentielle untere Schranke für die Multiplikation nachgewiesen werden. Nach den Erkenntnissen über die Multiplikation, die bei dem Beweis der unteren Schranken für OBDDs und FBDDs gewonnen wurden, war es nun ein nahe liegendes Ziel, auch allgemeinere Modelle zu untersuchen. Dabei sollte einerseits die Auswirkung von Nichtdeterminismus untersucht werden und andererseits sollten Berechnungsmodelle betrachtet werden, bei denen die gleiche Variable auf einzelnen Berechnungspfaden mehrmals vorkommen darf.

Beim $(1,+k)$-BP handelt es sich um ein Branchingprogramm, bei dem auf jedem Berechnungspfad k Variablen beliebig oft vorkommen dürfen, alle anderen Variablen jedoch (wie beim FBDD) nur einmal. Interessant ist dieses Modell unter anderem deswegen, weil ein $(1,+k)$-BP für $k = n$ ein uneingeschränktes Branchingprogramm ist (wenn n die Anzahl der Variablen darstellt). Mithilfe neuer Techniken zum Beweis unterer Schranken in diesem Berechnungsmodell konnte im Rahmen der Dissertation bewiesen werden, dass es keine polynomiell großen $(1,+k)$-BPs für $\mathrm{MUL}_{n-1,n}$ geben kann, wenn k nicht mindestens eine Größenordnung von $n/\log n$ hat [Wo02]. Asymptotisch bessere untere Schranken sind auch für keine andere explizit definierte Funktion bekannt.

Nichtdeterminismus ist eines der fundamentalsten Konzepte der theoretischen Informatik und es ist daher nicht verwunderlich, dass man auch nichtdeterministische Branchingprogrammmodelle in Analogie zu anderen nichtdeterministischen Rechenmodellen untersucht hat. Ein nichtdeterministisches Branchingprogramm enthält zusätzliche nichtdeterministische Knoten mit Ausgangsgrad 2, an denen sich ein Berechnungspfad in zwei Berechnungspfade aufteilt, die über die beiden ausgehenden Kanten des nichtdeterministischen Knoten verlaufen. Auf diese Weise erhält man mehrere Berechnungspfade und der Funktionswert $f(x)$ einer Eingabe x ist nun 1, wenn es einen Berechnungspfad gibt, der zur 1-Senke führt.

Da die Erkenntnisse über die Multiplikation noch nicht ausreichten, um exponentielle untere Schranken für allgemeine nichtdeterministische FBDDs zu beweisen, wurden im Rahmen der Dissertation zunächst FBDDs mit eingeschränktem Nichtdeterminismus untersucht. Bei einem $(\vee,k)$-FBDDs handelt es sich um ein nichtdeterministisches FBDD mit höchstens k nichtdeterministischen Knoten, die auf den Berechnungspfaden alle vor den mit Variablen markierten Knoten vorkommen müssen. Ähnlich wie bei $(1,+k)$-BPs konnte auch hier gezeigt werden, dass es keine polynomiell großen $(\vee,k)$-FBDDs für $\mathrm{MUL}_{n-1,n}$ geben kann, wenn k nicht mindestens eine Größenordnung von $n/\log n$ hat [Wo02]. Es sei angemerkt, dass es bis heute außer den Beweistechniken für unbeschränkte nichtdeterministische FBDDs keine andere Beweistechnik gibt, die für $(\vee,k)$-FBDDs mit $k \geq n/\log n$ superpolynomielle untere Schranken liefert.

6 Zusammenfassung und Ausblick

Wir können nicht erwarten, auf die eingangs gestellte Frage nach der Komplexität der Multiplikation in naher Zukunft für irgendein allgemeines Rechenmodell wie Schaltkreise oder Branchingprogramme eine vollständige Antwort zu finden. Wir können aber versuchen, nach und nach zu umfassenderen Erkenntnissen über die Multiplikation zu gelangen und neue Techniken zum Nachweis oberer und unterer Schranken zu entwickeln, um auf diese Weise die Komplexität der Multiplikation besser einzugrenzen. Die Ergebnisse der Dissertation stellen einen weiteren Schritt in diese Richtung dar. Dass dies nicht der letzte war, zeichnet sich schon an einer Reihe weiterführender Erkenntnisse über die Branchingprogrammkomplexität der Multiplikation ab. So konnten z. B. in [BWW02] und [BW] exponentielle untere Schranken in weiteren eingeschränkten nichtdeterministischen FBDD-Modellen nachgewiesen werden und kürzlich zeigten Sauerhoff und Woelfel exponentielle untere Schranken für nichtdeterministische Branchingprogramme, bei denen jede Variable auf jedem graphtheoretischen Pfad konstant oft vorkommen darf [SW03].

Literatur

[Br85] Bryant, R. E.: Symbolic manipulation of boolean functions using a graphical representation. *Proceedings of the 22nd ACM/IEEE Design Automation Conference (DAC)*. S. 688–694. 1985.

[Br86] Bryant, R. E.: Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*. C-35(8):677–691. 1986.

[BW] Bollig, B. und Woelfel, P.: A lower bound technique for nondeterministic graph-driven read-once-branching programs and its applications. Erscheint in *Theory of Computing Systems*.

[BW99] Bollig, B. und Wegener, I.: Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. Technical Report TR99-048. Electronic Colloquium on Computational Complexity (vgl. auch [BW00]). 1999.

[BW00] Bollig, B. und Wegener, I.: Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. In: *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP)*. S. 187–198. 2000.

[BW01] Bollig, B. und Woelfel, P.: A read-once branching program lower bound of $\Omega(2^{n/4})$ for integer multiplication using universal hashing. In: *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*. S. 419–424. 2001.

[BWW02] Bollig, B., Waack, S., und Woelfel, P.: Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. In: *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science (IFIP TCS)*. S. 83–94. 2002.

[Co66] Cobham, A.: The recognition problem for the set of perfect squares. In: *Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory*. S. 78–87. 1966.

[DHKP97] Dietzfelbinger, M., Hagerup, T., Katajainen, J., und Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*. 25:19–51. 1997.

[GM94] Gergov, J. und Meinel, C.: Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*. 43:1197–1209. 1994.

[Po98] Ponzio, S.: A lower bound for integer multiplication with read-once branching programs. *SIAM Journal on Computing*. 28:798–815. 1998.

[SS71] Schönhage, A. und Strassen, V.: Schnelle Multiplikation großer Zahlen. *Computing*. 7:281–292. 1971.

[SW95] Sieling, D. und Wegener, I.: Graph driven BDDs – a new data structure for Boolean functions. *Theoretical Computer Science*. 141:283–310. 1995.

[SW03] Sauerhoff, M. und Woelfel, P.: Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In: *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*. S. 186–195. 2003.

[We87] Wegener, I.: *The Complexity of Boolean Functions*. Wiley-Teubner. 1987.

[We88] Wegener, I.: On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*. 35(2):461–471. 1988.

[We93] Wegener, I.: Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions. *Information Processing Letters*. S. 85–87. 1993.

[We00] Wegener, I.: *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM. 2000.

[Wo99] Woelfel, P.: Efficient strongly universal and optimally universal hashing. In: *Mathematical Foundations of Computer Science: 24th International Symposium (MFCS)*. Lecture Notes in Computer Science 1672. S. 262–272. 1999.

[Wo01] Woelfel, P.: New bounds on the OBDD-size of integer multiplication via universal hashing. In: *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. Lecture Notes in Computer Science 2010. S. 563–574. 2001.

[Wo02] Woelfel, P.: On the complexity of integer multiplication in branching programs with multiple tests and in read-once branching programs with limited nondeterminism. In: *Proceedings of the 17th Annual IEEE Conference on Computational Complexity (CCC)*. S. 80–89. 2002.

[YCBO98] Yang, B., Chen, Y.-A., Bryant, R. E., und O'Hallaron, D. R.: Space- and time-efficient BDD construction via working set control. In: *Proceedings of the Asia South-Pacific Design Automation Conference (ASPDAC)*. S. 423–432. 1998.

Philipp Wölfel wurde 1972 in Würzburg geboren. Er studierte Wirtschaftsinformatik von 1993 bis 1995 an der Universität zu Köln und Informatik (Diplom) von 1995 bis 2001 and der Universität Dortmund. Er promovierte 2003 in Dortmund bei Prof. Dr. Ingo Wegener. Zurzeit ist er wissenschaftlicher Mitarbeiter am Lehrstuhl für effiziente Algorithmen und Komplexitätstheorie in Dortmund.