Verfahren zur Aufzählung von Polyedern in der Schleifenparallelisierung

Oliver Nyderle
Fakultät für Mathematik und Informatik
Universität Passau

betreut von Dr. Martin Griebl
Lehrstuhl für Programmierung
Professor C. Lengauer, Ph.D.

21. September 2000

Danksagungen

Ich möchte mich an dieser Stelle bei allen bedanken die mich bei der Erstellung dieser Arbeit unterstützt haben.

Ein besonderer Dank gilt dabei meinem Betreuer Dr. Martin Griebl, der immer ein offenes Ohr für mich hatte und mich unermüdlich mit Korrekturen und Anregungen versorgt hat. Danken möchte auch Prof. Christian Lengauer für seine Anregungen und Hinweise bei der Entstehung dieser Arbeit.

Ebenso danke ich meinen Eltern für die Unterstützung die sie mir während des Zeitraums der Entstehung gewährt. Mit Liebe danke ich meiner Freundin, die trotz eines enormen Zeitdrucks die Energie fand, sich der trockenen Substanz meiner Arbeit zu stellen.

Danken möchte ich außerdem all denen, die mich im Laufe der Entstehung immer wieder angespornt haben und meine Arbeit auch mehrfach haben.

Ein Dank gilt auch allen Mitarbeitern der Fakultät für Mathematik und Informatik der Universität Passau, ohne deren Arbeit ich mein Studium nie hätte durchführen können.

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Schleifenparallelisierung . . . 5

1.2 Übersicht . . . 7

2 Mathematische und algorithmische Grundlagen **8**

2.1 Mathematische Grundlagen . . . 8

2.1.1 Ordnungen . . . 8

2.1.2 Normalform für Ungleichungen . . . 9

2.1.3 Typisierung von Ungleichungssystemen . . . 9

2.2 Grundlagen für Polyeder . . . 9

2.2.1 Darstellungsformen . . . 10

2.2.2 In LooPo verwendete Form . . . 12

2.3 Das Polytopmodell . . . 12

2.3.1 Die Schleife . . . 12

2.3.2 Verschachtelung von Schleifen . . . 13

2.3.3 Schleifeninstanzen . . . 13

2.3.4 Abhängigkeiten von Instanzen . . . 14

2.3.5 Das Modell . . . 15

2.4 Die Polylib . . . 15

2.5 Die Eliminationsmethode nach Fourier und Motzkin . . . 16

2.5.1 Informelle Beschreibung der Idee . . . 16

2.5.2 Algorithmische Erklärung der Methode . . . 17

2.5.3 Komplexitätsbetrachtungen . . . 18

2.6 Berechnung der dualen Darstellung – Algorithmus von Chernikova . . . 19

2.6.1 Der Algorithmus . . . 21

2.6.2 Komplexität . . . 25

3 Das Eliminationsverfahren von V. Weispfenning **28**

3.1 Einleitung . . . 28

3.2 Informelle Beschreibung des Algorithmus . . . 28

3.2.1 Ein rechnerisches Beispiel . . . 30

3.3 Erweiterungen . . . 36

3.4 Komplexitätsbetrachtungen . . . 36

3.5 Umsetzung in LooPo . . . 37

3.6 Zusammenfassung . . . 40

4 Die Schleifengenerierung von F. Quilleré 42
4.1 Einleitung . . . 42
4.2 Grundlegende Notationen und Methoden . . . 42
4.2.1 Die Projektion von Polyedern . . . 43
4.2.2 Vereinfachung von Polyedern . . . 44
4.3 Das Aufzählen eines einzelnen Polyeders . . . 45
4.3.1 Problemstellung: Aufzählen eines Polyeders . . . 46
4.3.2 Grundlagen des Algorithmus . . . 46
4.3.3 Die Synthese der Schleifensätze . . . 48
4.3.3.1 Der Algorithmus . . . 49
4.3.3.2 Rechnerisches Beispiel für die Synthese . . . 50
4.3.4 Komplexitätsbetrachtungen . . . 52
4.3.5 Zusammenfassung . . . 52
4.4 Das Aufzählen mehrerer Polyeder . . . 53
4.4.1 Eine allgemeine Einführung . . . 53
4.4.2 Problem des Aufzählens mehrerer Polyeder . . . 54
4.4.3 Informelle Einführung in das Verfahren . . . 59
4.4.4 Formale Beschreibung des Algorithmus aus 4.4.3 . . . 63
4.4.4.1 Der Algorithmus . . . 63
4.4.4.2 Das Sortieren . . . 66
4.4.5 Komplexitätsbetrachtungen . . . 72
4.4.6 Zusammenfassung . . . 73
4.4.7 Erweiterungen und Probleme der Methode . . . 74

5 Zusammenfassung 76

1 Einleitung

1.1 Schleifenparallelisierung

Kann Parallelität die Ausführung von Programmen beschleunigen? Diese Frage wird sich jeder stellen, den Zeitprobleme bei der Abarbeitung eines Programmes dazu bringen, sich mit der parallelen Abarbeitung von Programmstrukturen auf Rechnernetzen oder Mehrprozessorsystemen zu befassen.

Wir können annehmen, daß das parallele Ausführen eines Algorithmus auf mehreren Prozessoren einen Geschwindigkeitsvorteil bringt, da die Berechnungen, die ein einzelner Prozessor nacheinander zu leisten hat, gleichzeitig von mehreren Prozessoren parallel erledigt werden können. In der Praxis treten hier jedoch einige Probleme auf. Nicht immer ist die parallele Ausführung eines Algorithmus auch die effizienteste und bringt den gewünschten Effekt. Der Vorteil, den wir durch eine parallele Ausführung gewinnen können, ist sehr stark von der Umsetzung eines Algorithmus in sein paralleles Pendant abhängig. Wird hier nicht die optimale Umsetzung erreicht, so kann dies den möglichen Geschwindigkeitsvorteil der parallelen Ausführung stark beeinträchtigen oder im ungünstigsten Fall sogar zu einer Verschlechterung des Laufzeitverhaltens führen.

Je nach Ausprägung des Ausgangsprogrammes gibt es für eine Parallelisierung verschiedene Vorgehensweisen. Welcher Weg zu einem parallelen Programm der effektivste ist, wissen wir jedoch vor der Parallelisierung nicht. Bisher ist eine solche Umwandlung noch mit sehr viel Handarbeit verbunden. Die sequentiellen Programme werden von Hand angepaßt und verfeinert, was natürlich sehr viel Zeit kostet und für eine Anwendung im größeren Rahmen in der Praxis wenig geeignet ist. Gebraucht werden deshalb automatische Parallelisierer, die entscheiden, wie ein Algorithmus effizient auf mehrere Prozessoren aufgeteilt werden kann. Es existieren inzwischen verschiedene Ansätze, einen sequentiellen Algorithmus automatisch in ein paralleles Programm umzuwandeln. Besonderes Augenmerk gilt hierbei Kontrollstrukturen, die Wiederholungen einer Berechnung beschreiben: den Schleifen. Einerseits ist hier eine effektive Parallelisierung wegen möglicher Datenabhängigkeiten eines jeden Durchlaufs von einem vorhergegangenen schwierig, da man nicht einfach eine beliebige Schleifeninstanz auf einem beliebigen Prozessor zur Ausführung geben kann. Andererseits vereinfachen Schleifen eine effektive Parallelisierung, da sie durch die Abhängigkeiten ihrer Durchläufe eine Struktur liefern, die Anhaltspunkte dafür gibt, wie die Instanzen auf die Prozessoren verteilt werden können, um einen effektiven Ablauf zu erreichen. Zudem garantiert eine Verteilung der einzelnen Berechnungsschritte auf mehrere Prozessoren unter Berücksichtigung aller Abhängigkeiten einen korrekten Programmablauf.

Theoretische Erkenntnisse zu dieser Problematik werden unter anderem vom Schleifenparallelisierer LooPo [LfP] (Loop Parallelizer in the Polyhedron Model) in die Praxis umgesetzt. Dieser am Lehrstuhl für Programmierung an der Universität Passau entwickelte Prototyp arbeitet vorgegebene sequentielle Schleifensätze für eine parallele Ausführung auf. Die theoretische Grundlage dafür liefert das Polytopmodell [Len93]. Die Ausführungsinstanzen der Schleifen, die den Indexraum dieser Schleifen bilden, werden dabei räumlich abgebildet und als Polyeder dargestellt. Die Berechnung einer Zuordnung der einzelnen Berechnungsschritte des Ursprungsprogrammes auf einen Punkt im Raum und in der Zeit ist bei diesem Modell ein zentrales Thema. Die parallel zueinander ausführbaren Instanzen werden in den Raum (also auf verschiedenen Prozessoren) und die nacheinander ausführbaren in die Zeit gelegt und durch parallele bzw. sequentielle Schleifen aufgezählt. Eine ausführlichere Einführung in diese Problematik ist in Abschnitt 2.3 zu finden. Dort werden auch die zum Verständnis notwendigen Grundlagen erläutert.

Aus diesem Ansatz ergeben sich zwei Probleme:

1. ein Schleifenraum wird durch ein Polyeder dargestellt; um es aufzählen zu können, müssen wir seine Grenzen bestimmen;

2. mehrere Schleifen, die nicht perfekt ineinander geschachtelt sind, generieren mehrere Polyeder, die sich überlappen können und für die eine korrekte Aufzählung gefunden werden muß.

Die für die Darstellung des berechneten Raumes benutzten Polyeder werden durch Ungleichungssysteme dargestellt. Für die Bearbeitung müssen diese aufgelöst werden, um so die Grenzen des beschriebenen Raumes zu berechnen. Hier kommen gängige Standardalgorithmen wie der von Fourier und Motzkin [DE73] zum Einsatz. Gerade das Lösen eines Ungleichungssystems verursacht einen nicht unerheblichen Berechnungsaufwand und ist schon bei relativ kleinen Systemen sehr rechenintensiv. Standardalgorithmen (wie z.B. nach Fourier-Motzkin) sind in der Anzahl der zu berechnenden Variablen doppelt exponentiell; das Arbeiten mit größeren Ungleichungssystemen wird dadurch erschwert bzw. unmöglich gemacht. Dies führt natürlich zu einer Einschränkung in der Benutzung von automatischen Parallelisierern, in unserem Falle von LooPo. Eine genauere Beschreibung der LooPo zugrundeliegenden Theorien und Methoden finden sich im Überblick bei [GL96] oder ausführlicher bei [Gri96].

Wir machen hier die Annahme, daß die in den hier vorgestellten Algorithmen benutzten Polytope auf reellen Werten basieren. Generell werden aber für die Ermittlung der Raum-Zeit-Koordinaten nur ganzzahlige Werte gewünscht bzw. sind schon durch die Formulierung der Probleme als Schleifen vorgegeben. Die hier vorgestellten Projektionsverfahren für solche Polytope setzten allerdings das Rechnen im reellen Raum voraus.

Das zweite große Problem bei der Umsetzung von sequentiellen Schleifen in ein parallel abarbeitbares Pendant ist das Aufzählen mehrerer in einem Programm vorkommender Schleifensätze. Jeder dieser Schleifensätze generiert ein Polyeder, so daß beim Aufzählen nicht nur ein Körper sondern eine Vereinigung mehrerer Polyeder betrachtet werden muß. Mit anderen Worten heißt das, daß für die Abarbeitung der einzelnen Schleifenrümpfe eine Reihenfolge gefunden werden muß, die auch mögliche Abhängigkeiten der Schleifensätze untereinander (geometrisch eine Überlappung zweier Polyeder) berücksichtigt.

Die bisher in der Praxis für die Aufzählung benutzten Verfahren haben im Zielprogramm oft einen erheblichen Aufwand an Rechenzeit verursacht. Bisher wurden die einzelnen Statements zum Aufzählen in einen Rahmen eingeschlossen, weshalb bei der Abarbeitung des Zielprogramms mehr Punkte im Raum als eigentlich nötig betrachtet werden müssen. Der zweite hier vorgestellte Algorithmus liefert in seiner Erweiterung eine neue Lösungsmöglichkeit für dieses Problem, indem nur die Bereiche des Raumes aufgezählt werden, die auch tatsächlich eine Instanz enthalten. In Kapitel 4.4.3 wird diese Problematik näher erläutert.

1.2 Übersicht

Im Rahmen dieser Arbeit sollen zwei Algorithmen zur Lösung von Ungleichungssystemen näher betrachtet und auf ihre Praktikabilität im Hinblick auf die Schleifenparallelisierung untersucht werden: der Eliminationsalgorithmus von Volker Weispfenning [Wei94] und der Ansatz von Fabien Quilleré [QRW99]. Der Ansatz von Weispfenning wendet das Prinzip des *Divide & Conquer* auf die Lösung von Ungleichungssystemen an. Quilleré erweitert dagegen bekannte Algorithmen wie z.B. die Fourier-Motzkin Elimination [DE73] oder den Algorithmus von Chernikova [Che65], welcher in der Polylib von Vincent Loechner [Loe99] implementiert ist. Die Grundlagen für die Implementation der Polylib stammen von Doran K. Wilde [Wil93].

Darüberhinaus soll das eigentliche Entwicklungsziel des Ansatzes von Quilleré näher erörtert werden. Die Intention der Arbeit Quillerés war nicht das Lösen eines Ungleichungssystems, sondern die Möglichkeit, mehrere sich überlappende Polyeder effizient aufzuzählen und eine fest vorgegebene Reihenfolge zu finden. Hierbei werden durch eine geschickte Zerlegung des Raumes im Gegensatz zu bisherigen Verfahren (z.B. [Wet95], [KPR95]) nur die Punkte im Raum betrachtet, die auch tatsächlich in einem der betrachteten Polyeder liegen.

Es soll außerdem untersucht werden, inwieweit sich diese Verfahren für einen praktischen Einsatz in LooPo eignen und wieviel Einsparungspotential sie im Gegensatz zu bisherigen Verfahren bieten. Dafür werden wir die theoretischen Aspekte der Arbeiten auf ihre praktische Umsetzbarkeit überprüfen.

Durch unterschiedliche Ausprägungen der Verfahren und unterschiedliche Zielsetzungen der Autoren ergeben sich unterschiedliche Aufgaben in der Praxis. Wir werden beispielsweise sehen, daß das Verfahren von Weispfenning angepaßt werden muß, um unsere Anforderungen vollständig erfüllen zu können. Andererseits wird nur ein Bruchteil der Möglichkeiten des Verfahrens genutzt. Der Ansatz von Quilleré hingegen ist schon durch seine Zielsetzung auf die Schleifenparallelisierung ausgerichtet.

2 Mathematische und algorithmische Grundlagen

2.1 Mathematische Grundlagen

2.1.1 Ordnungen

Eine Ordnung auf beliebigen Mengen kann nach [Krö91] und [BS91] wie folgt definiert werden:

> Sei $p : M \times M \to \mathbb{B}$ ($\mathbb{B}$ bezeichnet die Menge der Wahrheitswerte *true* oder *false*). p heißt **partielle Ordnung**, wenn p folgendes erfüllt:
>
> $$\begin{array}{rcl} \forall\, x \in M: & x\,p\,x & \text{(reflexiv)} \\ \forall\, x,y,z \in M: & x\,p\,y \wedge y\,p\,z \to x\,p\,z & \text{(transitiv)} \\ \forall\, x,y \in M: & x\,p\,y \wedge y\,p\,x \to x = y & \text{(antisymetrisch)} \end{array}$$
>
> Wenn zusätzlich noch gilt:
>
> $$\forall x,y \in M: \quad x\,p\,y \vee y\,p\,x \quad \text{(linear)}$$
>
> wird p als **total** bezeichnet.

Eine **lexikographische Ordnung** wird wie folgt definiert:

> Sei A ein Alphabet mit einer totalen Ordnung $\prec$. Wir können diese Ordnung nun zu einer **lexikographischen Ordnung** $\prec_l$ erweitern, indem wir für beliebige $x = x_1...x_m$ und $y = y_1...y_m$ aus A^* definieren:
>
> $x \prec_l y$ genau dann, wenn gilt:
>
> (i) $|x| \leq |y|$ und $\forall i \in \mathbb{N}, 1 \leq i \leq |x| : \quad x_i = y_i$
>
> oder
>
> (ii) $$\begin{array}{r} \exists\, k \in \mathbb{N}, 1 \leq k \leq \min\left(|x|, |y|\right), \\ \forall i \in \mathbb{N}, 1 \leq i \leq k-1: \quad x_k \prec y_k \wedge x_i = y_i \end{array}$$
>
> ($\leq$ bezeichnet hier die übliche Ordnung auf den natürlichen Zahlen).

2.1.2 Normalform für Ungleichungen

Eine Ungleichung mit beliebigen Faktoren $a_1 \dots a_n \in \mathbb{R}$, einer Konstanten c aus $\mathbb{R}$ und den Unbekannten $x_1 \dots x_n$ ist in **Normalform**, wenn sie der folgenden Form entspricht:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n + c \geq 0.$$

2.1.3 Typisierung von Ungleichungssystemen

Ungleichungssysteme können nach [Wei94] folgendermaßen unterteilt werden:

1. parameterfreie Ungleichungssysteme,
2. schwach parametrische Ungleichungssysteme,
3. parametrische Ungleichungssysteme.

Als **parameterfrei** werden Ungleichungen bezeichnet, die nur Variablen mit konstanten (als Zahl gegebenen) Koeffizienten und **eine** Konstante als Offset haben.

Schwachparametrische Ungleichungssysteme können darüber hinaus noch symbolische oder konstante Parameter als Offset haben. Die einzelnen Variablen sind weiterhin parameterfrei.

In einem **parametrischen** Ungleichungssystem haben auch die Variablen außer einer möglichen Konstante noch zusätzliche unbekannte Parameter als Koeffizienten.

In Fernandez und Quinton [FQ88] wird außerdem nach **homogenen** und **inhomogenen** Ungleichungssystemen unterschieden.

Ein inhomogenes Ungleichungssystem der Form:

$$\begin{aligned} Ax &\geq b \\ x &\geq 0, \end{aligned} \tag{2.1}$$

wobei b ein n-spaltiger Vektor ist, kann allerdings in ein äquivalentes homogenes Ungleichungssystem umgeformt werden. Das dabei entstehende System sieht dann folgendermaßen aus:

$$\begin{aligned} [A| - b]\; x' &\geq 0 \\ x' &\geq 0, \end{aligned} \tag{2.2}$$

wobei hier $[A| - b]$ eine $m \times (n + 1)$-dimensionale Matrix und x' ein Vektor mit $(n + 1)$ Spalten ist.

2.2 Grundlagen für Polyeder

Ein Polyeder wird durch die Menge

$$D = \{x \mid Ax \geq b\}$$

definiert, wobei A und b eine konstante Matrix bzw. ein konstanter Vektor sind. Ein solches Ungleichungssystem kann mit Formel 2.2 homogenisiert werden.

In Anlehnung an parametrische Ungleichungen gibt es auch parametrische Polyeder. Hierbei handelt es sich um Familien $D(p)$ von Polyedern. Für jeden Wert des Parameters p existiert ein Polyeder. Man kann für die Darstellung den Vektor b durch eine affine Kombination mit den Parametern ersetzen:

$$D(p) = \{x \mid Ax \geq Bp + b\},$$

wobei A und B konstante Matrizen und b ein konstanter Vektor sind.

Ein solches parametrisiertes Polyeder kann wieder in ein nicht parametrisiertes Ungleichungssystem überführt werden. Dabei legt man die Indexräume zusammen, so daß gilt:

$$\begin{aligned} D(p) &= \left\{x \mid \begin{pmatrix} A & -B \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} \geq b\right\} \\ D' &= \left\{\begin{pmatrix} x \\ p \end{pmatrix} \mid A' \begin{pmatrix} x \\ p \end{pmatrix} \geq b\right\}. \end{aligned}$$

Das Polyeder wird also durch die Parameter in seiner Dimensionalität vergrößert. So erhalten wir wieder ein Konstrukt, das mit Formel 2.2 homogenisiert werden kann.

2.2.1 Darstellungsformen

Es existieren verschiedene Möglichkeiten, ein Polyeder mathematisch darzustellen. Wir wollen hier die Darstellungen

1. durch Halbraumschnitte (Ungleichungen)
2. über Scheitelpunkte und Strahlen (Generatoren)

einführen.

Beide Darstellungen werden verschiedentlich in der Literatur (z.B. bei [Wil93]) auch als eine implizite (1) und eine explizite (2) Darstellung von Polyedern bezeichnet. Sie sind äquivalent und können ineinander umgewandelt werden. Die entsprechenden Beweise lassen sich bei Nemhauser/Wolsey [NW88] oder Wilde [Wil93] nachlesen. Für die Umwandlung der impliziten Darstellung (Ungleichungen/Halbraumschnitte) in eine explizite (Strahlen und Scheitelpunkte) existieren verschieden Ansätze. Es wird dabei zwischen Methoden mit und ohne Pivotelement unterschieden. Auf einen Algorithmus des zweiten Typs, der die beiden Darstellungen ineinander überführt, wollen wir in Abschnitt 2.6 noch genauer eingehen. Es handelt sich dabei um den Algorithmus von Chernikova [Che65], wie er historisch trotz einiger Verbesserungen immer noch genannt wird.

Wegen der Äquivalenz der beiden Darstellungen und der Möglichkeit sie jeweils ineinander überführen zu können wird oft der Begriff der **dualen** Darstellung verwendet

Die **Halbraumdarstellung** schneidet bildlich gesehen den Raum rund um das darzustellende Polyeder ab. Es wird soweit eingeschränkt, bis nur noch die Punkte enthalten sind, die das Polyeder beschreiben. Ein n-dimensionaler Halbraum wird durch

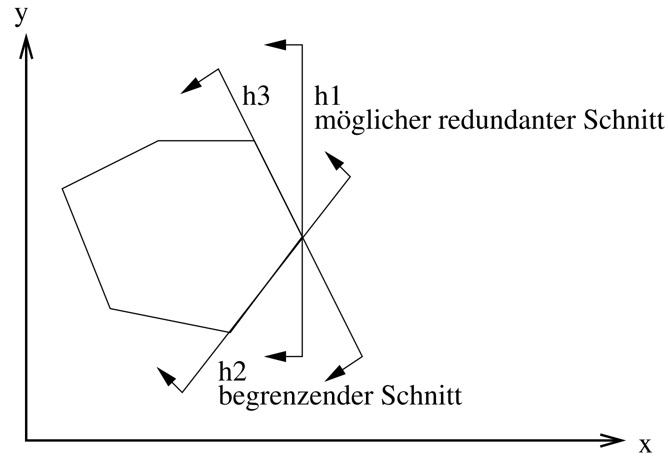


Abbildung 2.1: Mögliche Halbraumschnitte für die Polyederdarstellung

eine $(n-1)$-dimensionale Hyperebene begrenzt. Dies bezeichnet man als einen Halbraumschnitt. Jeder Halbraumschnitt kann mittels einer n-dimensionalen Ungleichung ausgedrückt werden. Ein n-dimensionales Ungleichungssystem mit m Ungleichungen beschreibt ein n-dimensionales Polyeder mit maximal m Begrenzungsflächen der Dimensionalität $n-1$. Das Polyeder kann dabei auch unbeschränkt sein. Nicht jede Ungleichung muß eine Begrenzungsfläche des Polyeders darstellen. Im Ungleichungssystem können auch Halbraumschnitte enthalten sein, die für die Beschreibung des Polyeders redundant sind. Die Definitionen der Darstellung können in [NW88, ab Seite 83] nachgelesen werden. Zur Veranschaulichung sei auf Abbildung 2.1 verwiesen. Die beiden Halbraumschnitte h_2 und h_3 begrenzen das Polyeder und beschreiben je eine begrenzende Seite des Polyeders. Wenn solche Schnitte für jede Seite eines Polyeders existieren, so ist es durch die Halbraumschnitte (Ungleichungen) vollständig beschrieben. Zusätzliche Halbraumschnitte wie h_1 im Bild sind redundant, können aber durchaus im beschreibenden Ungleichungssystem enthalten sein.

Die **Darstellung über Strahlen und Scheitelpunkte** beschreibt einen anderen Weg. Hier werden über die von einem Scheitelpunkt ausgehenden Strahlen Hyperebenen aufgespannt, die eine Fläche oder Kante bilden. Ausgehend von einem Scheitelpunkt wird mit $n-1$ Strahlen eine $(n-1)$-dimensionale Hyperebene aufgespannt. Ein n-dimensionales Polyeder läßt sich dann über die Eckpunkte und von den Eckpunkten ausgehende Strahlen beschreiben. Die Strahlen laufen dabei entlang der Kanten des Polyeders. Zur Veranschaulichung sei auf Abbildung 2.2 verwiesen, die dies für ein zweidimensionales Polyeder darstellt.

Die Grundlage hierfür liefert uns die Theorie von Minkowski (siehe [Sch94]) über die parametrische Darstellung von Polyedern. Sie besagt, daß ein beliebiges Polyeder durch eine Linearkombination von einer Menge von Linien L, Strahlen R und Scheitelpunkten

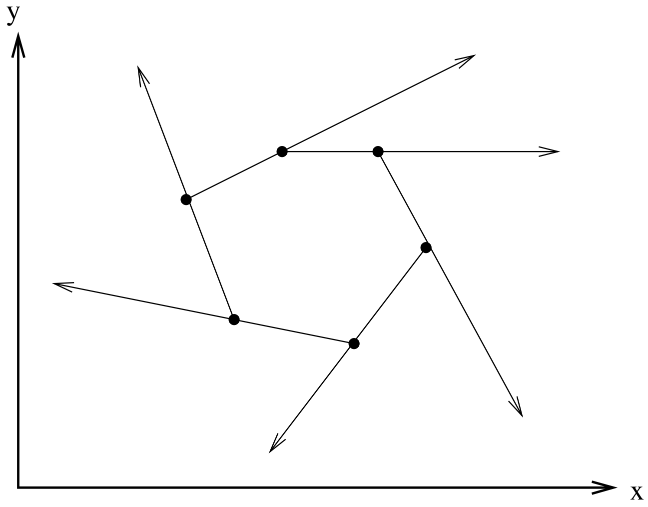


Abbildung 2.2: Darstellung eines Polyeders über Scheitelpunkte und Hyperebenen

V darstellbar ist:

$$P = \left\{ x \mid x = L\lambda + R\mu + V\nu,\ \mu,\nu \geq 0,\ \sum \nu = 1 \right\}. \tag{2.3}$$

2.2.2 In LooPo verwendete Form

Zur Beschreibung von Polyedern wird in LooPo die Darstellung mittels Halbraumschnitten bevorzugt. Das dazugehörige Ungleichungssystem besteht aus Ungleichungen in Normalform.

Üblicherweise schreiben wir in LooPo das Ungleichungssystem so, daß für x_i und x_j mit $i < j$ x_i eine weiter außenliegende Variable im korrespondierenden Schleifensatz ist. Im en ist x_j von x_i abhängig.

2.3 Das Polytopmodell

Grundlegend für Schleifenparallelisierung in LooPo ist das Polytopmodell. Es liefert die mathematischen Grundlagen für die Verschmelzung von Schleifenprogrammierung und Ungleichungssystemen für die Parallelisierung.

Dazu müssen wir allerdings erst ein paar grundlegende Begriffe der Schleifenparallelisierung einführen. Im folgenden wollen wir deshalb kurz die Begriffe Schleife, Verschachtelung, Instanz und Abhängigkeit einführen und das Polytopmodell vorstellen.

2.3.1 Die Schleife

Die Schleife ist ein strukturgebendes Element der Programmierung. Man unterscheidet die Klasse der zählenden und die Klasse der nicht zählenden Schleifen. Zu ersterer gehören beispielsweise die `For`-Schleifen, zu letzterer die `While`-Schleifen. Zur Vereinfachung betrachten wir hier die `For`-Schleifen, da sie durch ihre Semantik zur Übersetzungszeit klar definierte Grenzen bieten.

Eine `For`-Schleife der Form:

```
For i := u To o Do
  S1
  ⋮
  Sn
Endfor
```

beschreibt eine Schleife mit einem **Schleifenindex** i und den Statements $S_1, \ldots, S_n$, die innerhalb der Schleife auszuführen sind. Der Schleifenindex ist dabei durch seine Grenzen beschränkt. u bezeichnet die untere und o die obere Grenze des **Indexraumes**. Ein Schleifenprogramm besteht aus einer oder mehreren Schleifen. Der Indexraum kann je nach Beschaffenheit der ihn beschreibenden Schleifenkonstruktion auch mehrdimensional (z.B. im Falle eines verschachtelten Schleifensatzes) sein.

2.3.2 Verschachtelung von Schleifen

Statements innerhalb einer Schleife können auch wieder Schleifen darstellen. Um eine solche Struktur von Schleifen zu charakterisieren, unterscheidet man diese ineinander geschachtelten Schleifensätze nach der Art ihrer Verschachtelung in **perfekt verschachtelte** oder **nicht perfekt** (oder auch **imperfekt**) **verschachtelte** Schleifensätze.

Als **perfekt verschachtelt** gelten ineinander geschachtelte Schleifensätze dann, wenn (bis auf die innerste) jede Schleife genau eine weitere Schleife und keine sonstigen Statements (z.B. Zuweisungen) enthält. In einem perfekt verschachtelten Schleifensatz kann also eine Schleife der Ebene i nur genau eine sie umgebenden Schleife der Tiefe $i-1$ und nur genau eine innere Schleife der Tiefe $i+1$ haben.

Im Gegensatz dazu gibt es bei den **imperfekt verschachtelten** Schleifen bezüglich der Verschachtelung keinerlei Einschränkungen. Innerhalb einer Schleife können beliebig viele weitere Schleifen liegen.

In einer verschachtelten Schleifenstruktur benutzen wir die Bezeichnungen **innere Schleifen** und **äußere Schleifen**, um für eine Variable x_i und ihre korrespondierende Schleife die umschließenden bzw. umschlossenen Schleifen zu beschreiben. In einer perfekten Verschachtelung werden alle Schleifen mit einem Index $j > i$ als **innere Schleifen** und alle Schleifen mit $j < i$ als **äußere Schleifen** bezeichnet. Im imperfekten Fall ist die Kategorisierung schwieriger, da es Schleifen geben kann, die in der gleichen Ebene wie x_i liegen. Aber auch hier haben innere Schleifen einen höheren Index als äußere.

2.3.3 Schleifeninstanzen

Informell erklärt, beschreibt uns eine Instanz eines Statements eine Ausführung eines durch eine oder mehrere Schleifen umschlossenen Statements S. Einer Schleife bzw. ihrem Statement kann also je nach Beschaffenheit ihrer Grenzen eine Menge von Instanzen zugeordnet sein.

Betrachten wir ein Statement S, das von mehreren Schleifen $l_1, \ldots, l_n$ umschlossen wird, wobei der Index k einer Schleife l_k den Grad ihrer Verschachtelung angibt (l_1 ist die äußerste und l_n die innerste Schleife). Der **Vektor** $\vec{i}$, der aus den Indizes der umschließenden Schleifen $i_1, \ldots, i_n$ besteht, beschreibt dann genau einen Ausführungspunkt des Schleifensatzes. Man bezeichnet diesen Vektor auch als den **Indexvektor**. Die Menge aller Indexvektoren beschreibt die vollständige Abarbeitung des gesamten Schleifensatzes und kann folgendermaßen definiert werden:

$$\Im = \{(i_1, \ldots, i_n) \mid u_1 \leq i_1 \leq o_1, \ldots, u_n \leq i_n \leq o_n\}.$$

Als eine **Instanz** bezeichnet man dann ein Tupel $\langle S, \vec{i} \rangle$, das eine Ausführung des Statements S an der Stelle $\vec{i}$ beschreibt.

2.3.4 Abhängigkeiten von Instanzen

Der Ablauf eines Programmes ist durch eine Menge von Instanzen gekennzeichnet, die voneinander abhängig sein können. Für eine bestehende Abhängigkeit τ zwischen zwei Instanzen I_1 und I_2 auf einer Speicherzelle s schreiben wir:

$$\begin{array}{rcll} I_1 & \xrightarrow{\tau}_s & I_2 & \qquad \text{oder auch} \\ \langle S_1, \vec{i} \rangle & \xrightarrow{\tau}_s & \langle S_2, \vec{j} \rangle, & \end{array}$$

wobei entweder die Statements S_1 und S_2 oder die Vektoren $\vec{i}$ und $\vec{j}$ gleich sein können. Das s kennzeichnet die Speicherzelle, die von beiden Instanzen referenziert wird, und τ bezeichnet die Art der Abhängigkeit. Damit eine Instanz I_1 von einer Instanz I_2 **abhängig** ist, müssen die ersten zwei der folgenden drei Bedingungen erfüllt sein:

1. In der vorgegebenen Reihenfolge der Ausführungen wird I_1 vor I_2 ausgeführt. Damit bezeichnen wir I_1 als **Quelle** und I_2 als **Ziel**.

2. Beide Instanzen greifen auf dieselbe Speicherzelle s zu, d.h. sie arbeiten auf denselben Daten.

3. Es existiert keine Instanz I_3, so daß gilt: $I_1 \xrightarrow{\tau}_s I_3$ und $I_3 \xrightarrow{\tau}_s I_2$ und I_3 modifiziert die Speicherzellen s.

Die Bedingung 3 ist optional.

Wir unterscheiden verschiedene Arten von Abhängigkeiten. Je nach Art des Zugriffs auf die Speicherzelle wird eine Abhängigkeit als **Input**-, **True**-, **Anti**- oder **Output**-Abhängigkeit bezeichnet. Einen Überblick über den Zusammenhang der Zugriffe liefert Tabelle 2.1. Die **Input**-Abhängigkeit wird vernachlässigt. Die **True**-Abhängigkeit wird, wenn Bedingung 3 erfüllt ist, auch als **Flow**- oder **Fluß**-Abhängigkeit bezeichnet.

Die Differenz $\vec{j} - \vec{i}$ der beiden Indexvektoren $\vec{i}$ und $\vec{j}$ nennt man den **Abhängigkeitsvektor** der beiden Instanzen.

Ziel	**Quelle**	
	liest	schreibt
liest	*Input*	*True*
schreibt	*Anti*	*Output*

Tabelle 2.1: Abhängigkeiten von Instanzen

2.3.5 Das Modell

Um eine Kontrollstruktur wie eine Schleife parallelisieren zu können, ist es notwendig, jede Instanz einer Schleifenstruktur in den Raum und in die Zeit auszulegen.

Das Verteilen im Raum bedeutet dabei, daß die Instanzen auf virtuelle Prozessoren verteilt werden. Es handelt sich um virtuelle Prozessoren, da wir uns in einem universellen Modell befinden, das zunächst von der eigentlichen parallelen Architektur, auf der wir das Programm später ausführen wollen, unabhängig sein soll und somit auch eine unbeschränkte Anzahl von Prozessoren erlaubt. Wir bezeichnen eine solche Zuordnung auf verschiedene Prozessoren als eine **Allokation**.

Das Auslegen in der Zeit bedeutet, daß die Instanzen nach ihren Abhängigkeiten auf – wiederum wegen der Unabhängigkeit bei der Wahl der Architektur – **virtuelle** Zeitpunkte verteilt werden. Eine solche zeitliche Verteilung nennt man einen **Schedule**.

Schedule und Allokation einer Schleife werden als Funktionen über die Schleifenindizes ausgedrückt. Die Instanzen sind so verteilt, daß die Abhängigkeiten zwischen Instanzen in der Zeit liegen, die gegebenen Abhängigkeitsvektoren also den Schedule bestimmen. Da Kommunikation zwischen zwei Prozessoren den größten Aufwand darstellt, ist eine ideale Verteilung dann erreicht, wenn alle abhängigen Instanzen in der Zeit und alle unabhängigen Instanzen im Raum ausgelegt sind. Dadurch wird Kommunikation zwischen den Prozessoren vermieden.

2.4 Die Polylib

In späteren Kapiteln wird an verschiedenen Stellen eine mathematische Bibliothek, namens *Polylib*, für verschiedene Berechnungen verwendet. Sie implementiert Operationen auf Polyedern. Entwickelt wurde die Bibliothek 1993 von Doran K. Wilde [Wil93], wobei die Arbeit inzwischen von Vincent Loechner [Loe99] fortgeführt wird.

In der Polylib sind neben dem Algorithmus von Chernikova (siehe Abschnitt 2.6) auch allgemeinere Berechnungen auf Polyedern implementiert. So finden sich hier Funktionen zur Berechnung des Schnitts, der Vereinigung oder der konvexen Hülle. Viele der Funktionen basieren dabei auf der Verwendung der impliziten/expliziten (oder auch dualen) Darstellung als Grundlage für die Berechnungen.

In den folgenden Kapiteln wird die Polylib mehrfach verwendet bzw. liefert die grundlegende Struktur für die Methoden in Kapitel 4.

2.5 Die Eliminationsmethode nach Fourier und Motzkin

Für das Lösen von Ungleichungssystemen (die Elimination von Variablen in einem Ungleichungssystem) wird in vielen Fällen der Algorithmus nach Fourier und Motzkin verwendet. Dieser basiert auf den Arbeiten von J.B.J. Fourier aus dem Jahre 1826, die von T.S. Motzkin in den 30er Jahren des zwanzigsten Jahrhunderts wiederentdeckt wurden. Man könnte sie als eine Erweiterung der Gauss-Elimination auf Ungleichungssysteme betrachten.

Im folgenden Abschnitt soll die Funktionsweise des Fourier-Motzkin Algorithmus zur Elimination von Variablen in Ungleichungssystemen kurz eingeführt werden. Wir werden dazu erst eine allgemeine informelle Beschreibung des Algorithmus geben, um dann den Leser in die einzelnen Schritte formaler einzuführen. Abschließend fassen wir ein paar grundlegende Gedanken in Bezug auf die Komplexität zusammen.

Als Referenz dienen die Ausführungen von Uptal Banerjee in [Ban93].

2.5.1 Informelle Beschreibung der Idee

Der Algorithmus von Fourier und Motzkin zielt auf die schrittweise Elimination der Variablen eines Ungleichungssystems ab. Der Grundgedanke des Verfahrens läßt sich graphisch mit einer Projektion des durch das Ungleichungssystem dargestellten Polyeders beschreiben. In jedem Eliminationsschritt wird das Polyeder um eine Dimensionalität reduziert, also in den $(n-1)$-dimensionalen Raum projiziert. Hat das neue Ungleichungssystem eine Lösung, so ist auch das Ausgangssystem lösbar. Dabei wird wie folgt vorgegangen:

> Gegeben sei ein Ungleichungssystem mit m Ungleichungen und n Variablen. Zur Elimination der Variablen wird zunächst aus den Variablen $x_1, \ldots, x_n$ ein x_i ausgewählt, das aus dem System eliminiert werden soll. Dazu werden die oberen und unteren Schranken für x_i berechnet. Wir erhalten also Ungleichungen der Form $x_i \leq \ldots$ und $x_i \geq \ldots$, bei denen x_i koeffizientenfrei auf der einen Seite der Ungleichung steht. Ungleichungen mit dem Koeffizienten 0 für x_i werden bei der Elimination von x_i nicht weiter betrachtet. Sie sind schon von x_i unabhängig. Die Berechnung der Schranken erfolgt durch einfache Umformung der entsprechenden Terme. Bei einer Ungleichung in Normalform:
>
> $$a_1x_1 + a_2x_2 + \cdots + a_nx_n + c \geq 0$$
>
> lassen sich die nach oben bzw. unten begrenzenden Schranken einer Variable x_i an ihrem Koeffizienten a_i erkennen. Gilt $a_i < 0$, so wird x_i durch die Ungleichung nach oben beschränkt. Durch eine Umformung entsteht eine Ungleichung der Form:
>
> $$\frac{a_1}{a_i}x_1 + \frac{a_2}{a_i}x_2 + \cdots + \frac{a_{i-1}}{a_i}x_{i-1} + \frac{a_{i+1}}{a_i}x_{i+1} + \cdots + \frac{a_n}{a_i}x_n + \frac{c}{a_i} \geq x_i. \quad (2.4)$$
>
> Im anderen Fall, $a_i > 0$, beschränkt die Ungleichung x_i nach unten und es

entsteht nach der Umformung eine Ungleichung folgender Form:

$$x_i \geq \frac{-a_1}{a_i}x_1+\frac{-a_2}{a_i}x_2+\cdots+\frac{-a_{i-1}}{a_i}x_{i-1}+\frac{-a_{i+1}}{a_i}x_{i+1}+\cdots+\frac{-a_n}{a_i}x_n+\frac{-c}{a_i} \quad (2.5)$$

oder

$$\frac{a_1}{a_i}x_1 + \frac{a_2}{a_i}x_2 + \cdots + \frac{a_{i-1}}{a_i}x_{i-1} + \frac{a_{i+1}}{a_i}x_{i+1} + \cdots + \frac{a_n}{a_i}x_n + \frac{c}{a_i} \geq -x_i. \quad (2.6)$$

Die so ermittelten Schranken werden nun einander gegenübergestellt. Jeder oberen Schranke wird paarweise jede untere Schranke gegenübergestellt bzw. die Ungleichungen (2.4) und (2.6) addiert. Das so entstandene Ungleichungssystem ist von x_i unabhängig. Als Ergebnis eines Eliminationsschritts, also der Elimination von x_i, erhält man ein neues Ungleichungssystem, das sich aus den neu entstandenen und den schon vorhandenen von x_i unabhängigen Ungleichungen (nämlich denen, deren Koeffizient $a_i = 0$ war) zusammensetzt.

Um alle Variablen zu eliminieren, müssen die vorherigen Schritte nun für jede verbleibende Variable wiederholt werden.

2.5.2 Algorithmische Erklärung der Methode

Im folgenden soll die Arbeitsweise des Eliminationsalgorithmus nach Fourier und Motzkin algorithmisch erklärt werden. Als Grundlage dient die Umsetzung von Uptal Banerjee in [Ban93]. Der Algorithmus wird hier für parameterfreie Ungleichungssysteme beschrieben. Eine Implementation dieses Algorithmus wurde von Sven Anders im Rahmen von LooPo erstellt [And96] und auf schwach parametrische Ungleichungssysteme erweitert. Zur weiteren Erläuterung der Vorgehensweise beschränken wir uns auf ein parameterfreies System.

Gegeben sei ein parameterfreies Ungleichungssystem der Dimension $n \times m$. Die Elimination beginnt bei der innersten Variable (also x_n). Der Algorithmus geht wie folgt vor:

1. Sortiere die Ungleichungen nach der zu eliminierenden Variable x_i; alle Ungleichungen mit positivem Koeffizienten für x_i stehen nach der Sortierung am Anfang des Ungleichungssystems gefolgt von den Ungleichungen mit negativen Koeffizienten für x_i; zuletzt folgen alle Ungleichungen, deren Koeffizient für x_i, $a_i = 0$ ist.

2. In den Zeilen des Ungleichungssystems wird nun x_i isoliert; dafür werden alle Koeffizienten einer Zeile durch den Koeffizienten von x_i dividiert; $a_{jk} = a_{jk}/a_{ik}$, wobei $1 \leq j, i \leq n$ und $1 \leq k \leq m$ ist.

3. Analyse der oberen und unteren Schranke für x_i; wir merken uns die berechneten Ungleichungen für unsere Lösung, indem wir sie in ein Lösungsungleichungssystem kopieren.

4. Eliminieren der Variable x_i in unserem Ungleichungssystem, indem wir jede untere Schranke allen oberen Schranken mit $\leq$ gegenüberstellen. Die so neu entstehenden Ungleichungen werden dem Ungleichungssystem zugefügt. Die oberen und unteren Schranken von x_i werden entfernt. Das Ungleichungssystem ist nun von x_i unabhängig.

5. Wenn $i > 1$, setze $i = i - 1$ und fahre mit Schritt 1 fort; wenn $i = 1$ ist, so ist die Berechnung der letzten Variable abgeschlossen; in diesem Falle sind wir am Ende der Rekursion angelangt und der Algorithmus terminiert.

Nach der Terminierung bleibt im generierten Ungleichungssystem das aufgelöste Ausgangsungleichungsystem stehen. Es müssen nun noch etwaige Redundanzen beseitigt werden, um eine möglichst minimale Beschreibung der Lösung zu erhalten.

2.5.3 Komplexitätsbetrachtungen

Der Algorithmus von Fourier und Motzkin ist durch die häufige Sortierung der Ungleichungen und ihre rasch anwachsende Anzahl sehr rechenaufwendig. Die Anzahl der Ungleichungen steigt in jedem Eliminationsschritt stark an, da jedesmal jede untere Schranke allen oberen gegenüber gestellt wird. Neben dem großen rechnerischen Aufwand haben wir also auch mit einem hohen Speicherverbrauch zu kämpfen. Für die Berechnung eines Systems mit n Variablen und m Ungleichungen werden im schlimmsten Falle nach dem ersten Berechnungsschritt $m' = \left(\frac{m-k}{2}\right)^2 + k$ Ungleichungen zu betrachten sein, wobei k die Anzahl der Ungleichungen bezeichnet, die in diesem Schritt wegen des Koeffizienten 0 für die zu eliminierende Variable nicht betrachtet wurden. Nach dem zweiten Berechnungsschritt sind es dann entsprechend $m'' = \left(\frac{m'-k'}{2}\right)^2 + k'$ Ungleichungen. Damit ergibt sich eine Komplexität von $O\left(m^{2^n}\right)$.

Die doppelt exponentielle Laufzeit macht den Algorithmus zur Lösung großer Probleme in der Praxis ungeeignet. Schon vergleichsweise kleine Probleme benötigen einen hohen Aufwand an Rechenzeit. Ein Beispiel:

> Nehmen wir an, wir wollten ein Ungleichungssystem mit 6 Unbekannten und 10 Ungleichungen lösen. Im schlimmsten Fall hätten wir im ersten Eliminationsschritt $\left(\frac{12}{2}\right)^2 = 36$ generierte Ungleichungen. Im zweiten Schritt sind es dann schon $\left(\frac{36}{2}\right)^2 = 169$, im dritten $\left(\frac{169}{2}\right)^2 = 7140$ (abgerundet), im vierten $\left(\frac{7140}{2}\right)^2 = 12744900$, im fünften $\left(\frac{12744900}{2}\right)^2 = 4.06 \cdot 10^{13}$ und im letzten Eliminationsschritt $\left(\frac{4.06 \cdot 10^{13}}{2}\right)^2 = 4.12 \cdot 10^{26}$ Ungleichungen, die betrachtet werden müssen.

Dieses Problem läßt sich nur durch eine Verminderung der Anzahl der in jedem Schritt neu entstehenden Ungleichungen verringern. Man könnte dies z.B. durch einen Adjazenztest/Redundanztest nach der Generierung erreichen, um redundante Informationen zu streichen und dadurch das Ungleichungssystem zu verkleinern. Ein Redundanztest über allen Ungleichungen ist allerdings sehr aufwendig, weshalb man sich im allgemeinen auf einfachere Tests beschränkt (Eliminieren von „gleichen” Ungleichungen, Eliminieren von Zeilen, die für alle Variablen den Koeffizienten 0 beinhalten, usw.).

Im weiteren werden wir die Abkürzung „FM” für diesen Algorithmus verwenden.

2.6 Berechnung der dualen Darstellung – Algorithmus von Chernikova

In diesem Abschnitt wollen wir ein Verfahren zur Berechnung der dualen Darstellung eines Polyeders vorstellen. Es existieren zwei Ansätze, zwei Klassen von Algorithmen für eine solche Berechnung: die Methoden mit und die Methoden ohne Pivotstrategie. Die Methoden, die mit Pivotstrategien arbeiten, sind von der Simplex-Methode abgeleitet. Sie finden neue Scheitelpunkte, die mit den bereits bekannten zusammenhängen, indem sie Simplex-Pivot-Operationen benutzen.

Das hier vorgestellte Verfahren basiert allerdings auf der ersten Klasse von Algorithmen, den Methoden ohne Pivotstrategie. Die duale Darstellung wird hier folgendermaßen berechnet:

- Als Erstes wird ein Tableau initialisiert, in dem ein initiales Polyeder (z.B. das universelle, also der gesamte Raum) gleichzeitig in beiden Darstellungen (siehe Abschnitt 2.2.1) repräsentiert wird.
- Als Nächstes wird rekursiv in jedem Schritt zum vorherigen Polyeder eine neue Ungleichung (oder auch Gleichung) hinzugefügt. Die beiden Darstellungen des neuen Polyeders werden berechnet, bis man schließlich am Ende der Rekursion, wenn alle Ungleichungen (oder Gleichungen) des Ausgangspolyeders der neuen Darstellung hinzugefügt wurden, das gewünschte Polyeder erhält.

Die Reihenfolge, in der die Begrenzungen hinzugefügt werden, hat keine Auswirkungen auf das Ergebnis, allerdings beeinflußt sie erheblich die Laufzeit des Algorithmus. Die Komplexität dieses Problems ist $O\left(n^{\lfloor \frac{m}{2} \rfloor}\right)$, wobei m die Anzahl der Beschränkungen (Ungleichungen) und n die Dimensionalität bezeichnen. Eine Begrenzung oder Beschränkung eines Polyeders kann dabei eine Ungleichung oder eine Gleichung sein.

Das hier beschriebene Verfahren wird gemeinhin als der Algorithmus von Chernikova bezeichnet. Wir orientieren uns vor allem an der Umsetzung von Doran K. Wilde in der Polylib [Wil93], da genau diese Implementation in einem der hier vorgestellten Verfahren benutzt wird. Nähere Erläuterungen zum Verfahren findet man auch bei Hervé Le Verge [Le 92].

Die Bezeichnung des Algorithmus als der „Algorithmus von Chernikova” hat historische Gründe, da die Methoden, die nicht auf Pivotstrategien zurückgehen, eigentlich auf einer Arbeit von T.S. Motzkin basieren. N.V. Chernikova hatte diese dann 1965 [Che65] verfeinert und bekannt gemacht.

T.S. Motzkin stellte schon 1953 in Motzkin et al. [MRTT53] einen allgemeinen Algorithmus vor, der das Problem der Berechnung der dualen Darstellung für polyhedrische

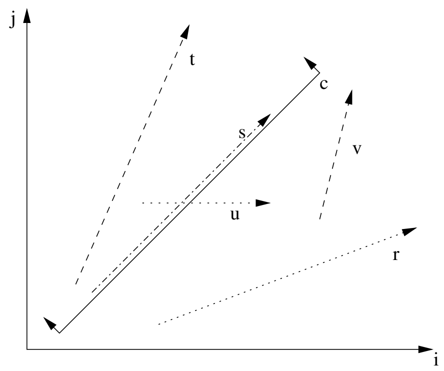


Abbildung 2.3: Einteilung der Strahlen nach der Beschränkung

Kegel[1] iterativ löst. Ausgehend von der Darstellung des universellen Polyeders wird dabei in jeder Iteration eine neue Beschränkung des zu berechnenden Polyeders zum aktuellen Kegel hinzugefügt. Der Kegel besteht dabei aus Strahlen und Linien. Die Strahlen, die den Kegel definieren, lassen sich in drei Mengen unterteilen:

1. R^+ sind die Strahlen, die eine Beschränkung erfüllen.
2. R^0 sind die Strahlen, die eine Beschränkung durchdringen (also auf einer Beschränkung liegen).
3. R^- sind die Strahlen, die einer Beschränkung des Polyeders widersprechen (also in einer der Beschränkung entgegengesetzten Richtung liegen).

Die Kategorisierung der Strahlen erfolgt dabei anhand ihrer Richtungsvektoren und ist unabhängig von ihrer Lage im Raum. Betrachten wir dazu die Abbildung 2.3. In unserem Beispiel erfüllt der Strahl t die Beschränkung (Ungleichung) c und ist deshalb in R^+. Der Strahl s liegt auf der Beschränkung und gehört deshalb offensichtlich zu R^0. Der Strahl r widerspricht eindeutig der Beschränkung und ist somit ein Element von R^-. Die beiden Strahlen u und v sind wegen ihrer Lage im Raum auf den ersten Blick nicht so einfach zuzuordnen. Wegen ihrer Richtung kann allerdings u eindeutig R^+ und v eindeutig R^- zugeordnet werden.

Der neu berechnete Kegel, der die Beschränkung einhält, wird dann aus diesen Mengen konstruiert.

Das Hauptproblem der auf dieser Grundlage aufbauenden Methoden ist die Tatsache, daß sie eine nicht minimale Menge von Strahlen generieren, da nicht nur Extrema sondern auch redundante Strahlen erzeugt werden. Ohne Einschränkungen würde dann

[1] Kegel sind ein Spezialfall der Polyeder: In Nemhauser und Wolsey [NW88] heißt es dazu: „A polyhedron $Ax \leq 0$ is a cone." Da sich Polyeder in eine solche Darstellung umwandeln lassen, ist dies keine Einschränkung der Gültigkeit des Verfahrens.

die Anzahl der Strahlen exponentiell ansteigen und dementsprechend Probleme sowohl beim Speicherbedarf als auch bei der Geschwindigkeit des Algorithmus verursachen.

Motzkin bietet einen einfachen und eleganten Test zur Lösung dieses Problems an. Er zeigte, daß genau dann die konvexe Kombination eines Paares von Strahlen (hier $r^+ \in R^+$ und $r^- \in R^-$) einen maximalen Strahl für den neuen Kegel als Ergebnis liefert, wenn die kleinste Begrenzungsfläche, die beide Strahlen enthält, folgendes erfüllt:

1. ihre Dimensionalität ist um eins größer als die Dimensionalität von r^+ und r^-, und

2. sie enthält **nur** die Strahlen r^+ und r^-.

Dieser Test verhindert, daß zusätzliche Strahlen generiert werden und bringt die Lösung in eine minimale Form.

Später wurde dieser Algorithmus von anderen Forschern erweitert und verbessert. N.V. Chernikova stellte 1965 [Che65] einen ähnlichen Algorithmus vor, der eine spezielle Variante des Problems löst, nämlich den Fall, in dem die Variablen alle nicht negativ sind ($x \geq 0$). Die von der Autorin benutzte Methode entspricht der Vorgehensweise von Motzkin, allerdings verwendet sie ein kleineres und weiterentwickeltes Tableau. Später wurde die Methode von Chernikova durch F. Fernández und P. Quinton [FQ88] verbessert. Sie beseitigten die Einschränkung bezüglich der Größe der Variablen (> 0) und erweiterten den Algorithmus um eine Heuristik, welche die Ungleichungen sortiert und dadurch den Algorithmus beschleunigen soll. Ein Großteil der Ausführungszeit des Algorithmus wird für den Adjazenztest benötigt. H. Le Verge [Le 92] nahm sich dieser Problematik an. Er verbesserte die Geschwindigkeit des in [FQ88] verwendeten Redundanztests.

2.6.1 Der Algorithmus

Nachdem wir im vorherigen Abschnitt über einige grundlegende Aspekte der Methode geschrieben haben, wollen wir nun den Algorithmus etwas näher erläutern. Wir beziehen uns dabei auf die Implementation in der Polylib. Die Programmcodefragmente stammen aus [Wil93].

Wie schon im vorherigen Ansatz angedeutet, wird bei der vorgestellten Methode in jedem Iterationsschritt das vorliegende Polyeder (vom vorhergehenden Berechnungsschritt) durch die Addition einer neuen Beschränkung (Ungleichung) verfeinert. Das resultierende Polyeder spiegelt diese neue Beschränkung in seinem Aussehen wieder.

Eine Ungleichung der Form

$$ax \geq 0$$

kann durch einen Halbraum H^+

$$H^+ = \{x : ax \geq 0\}$$

geometrisch interpretiert werden. H^+ enthält dann alle Punkte des Raumes, für die diese Ungleichung erfüllt ist. Bei jedem Iterationsschritt des Algorithmus wird nun eine neue Ungleichung (dargestellt durch ihren Halbraum H^+) zum System hinzugefügt.

Das Polyeder $P = L + R$ [2] wird durch einen Schnitt mit dem Halbraum durch diese neue Beschränkung verändert. Es entsteht ein neues Polyeder $P' = L' + R'$, welches im nächsten Iterationsschritt weiter verfeinert wird. Ausgangspunkt der Berechnung ist ein Polyeder, das dem universellen Polyeder entspricht. Es enthält deshalb noch keine Strahlen in R, und in L genau die Linien, welche die n Dimensionsachsen des zu berechnenden n-dimensionalen Polyeders beschreiben. Dieses wird dann rekursiv von den beschränkenden Ungleichungen des umzuwandelnden Polyeders verfeinert.

Betrachten wir jetzt die einzelnen Bestandteile des Algorithmus. Der Algorithmus `Dual` beschreibt die Methode von Motzkin, mit welcher wir die duale Repräsentation einer gegebenen Menge von Ungleichungen bestimmen können. Bei der Ausführung von `Dual` werden drei Prozeduren aufgerufen, die das Polyeder modifizieren. In `ConstrainL` wird der Raum der Linien mit einer neuen Beschränkung verändert. Mit `AugmentR` erhalten wir ein Verfahren, welches die Dimension des Strahlenraums vergrößert. Durch `ConstrainR` wird schließlich der Strahlenraum beschränkt. Die Abbildungen 2.4–2.8 zeigen die Pseudocodelistings dieser Prozeduren. Sie werden später noch im einzelnen erläutert.

Wir haben als Ausgangssituation einen unbeschränkten n-dimensionalen Raum in L, der durch die n Linien der Dimensionsachsen des darzustellenden n-dimensionalen Polyeders gebildet wird. Der Strahlenraum R ist leer. Der Algorithmus soll mittels der Beschränkungen a den Raum so schneiden, daß nach Betrachtung aller a in A die Kombination des Linien- und des Strahlenraumes dem Polyeder entspricht, das durch die Beschränkungen in A beschrieben wird.

Die Abarbeitung besteht aus zwei Phasen. In der ersten Phase wird zunächst der Linienraum in seiner Dimensionalität verkleinert (entsprechend der Beschränkung geschnitten) und seine eliminierte Dimension wird in den Strahlenraum übernommen. Dadurch vergrößert sich Schritt für Schritt die Dimensionalität des Strahlenraumes. Die Kombination beider Räume repräsentiert damit immer einen n-dimensionalen Körper. Wenn keine Beschränkungen, die den Linienraum weiter eingrenzen können, mehr vorliegen, oder keine Linien mehr im Linienraum existieren, wird die zweite Phase ausgeführt. Hierbei wird nun der Strahlenraum mit den Beschränkungen, die keine Auswirkungen auf den Linienraum haben, geschnitten und dadurch weiter verfeinert. Auf diesem Wege werden im Verlauf der beiden Phasen alle Beschränkungen nach und nach zum Strahlenraum hinzugefügt, so daß nach Ablauf des Algorithmus durch Kombination von L und R das Ausgangspolyeder beschrieben wird.

Wenden wir uns den einzelnen Prozeduren zu:
Konzentrieren wir uns zunächst auf die Ausgangsprozedur `Dual` (siehe Abbildung 2.4). Als erstes werden hier L und R nach unseren Vorgaben so generiert, daß sie das universelle Polyeder darstellen. Für alle Beschränkungen a konstruiert man nun mit `ConstrainL` aus

[2] Nach [NW88] sind Polyeder, die der Form $Ax \geq 0$ entsprechen, Kegel. Die von uns verwendeten Polyeder P entsprechen dieser Definition. Nach [Wil93] wird ein Kegel durch eine Linearkombination von Strahlen R und Linien L definiert und stellt ein Polyeder mit nur einem Scheitelpunkt dar. Ohne Einschränkung der Allgemeinheit ist dieser Scheitelpunkt der Nullpunkt. Die ursprüngliche Definition für die parametrische Darstellung von Polyedern von Minkowski (siehe Abschnitt 2.3, Seite 12) kann deshalb zu der hier verwendeten Zerlegung vereinfacht werden. Siehe dazu [Wil93] oder [Sch94].

```
Procedure Dual(A), Returns L, R

        L :=Basis für einen n-dimensionalen linearen Raum
        R :=Nullpunkt
        For alle Ungleichungen a ∈ A Do
            r_new :=ConstrainL (L, a)
            If r_new ≠ 0 Then AugmentR (R, a, r_new) Else ConstrainR (R, a)
        End
        Return L, R
```

Abbildung 2.4: Algorithmus zu Berechnung von `Dual`(A)

```
Procedure Combine(r1, r2, a), Returns r3

        D := GGT(ar1, ar2)
        λ1 := ar2 ÷ D
        λ2 := −ar1 ÷ D
        r3 := λ1r1 + λ2r2
        Return r3
```

Abbildung 2.5: Prozedur zur Berechnung von `Combine`(r_1, r_2, a)

den Linien in L einen neuen Strahl r_{new}, der die Beschränkung erfüllt. Läßt sich ein solcher Strahl finden, so wird er mit `AugmentR` der Strahlenmenge hinzugefügt. Andernfalls wird durch `ConstrainR` der bisherige Strahlenraum mit der neuen Beschränkung a verfeinert. Die Algorithmen `ConstrainL` und `AugmentR` gehören zu der oben beschriebenen Phase 1 und `ConstrainR` zur Phase 2 des Algorithmus. Als Ergebnis von `Dual` erhalten wir eine durch Strahlen und Linien beschriebene duale Darstellung des zu berechnenden Polyeders.

Nun wollen wir näher auf die in `Dual` verwendeten Funktionen eingehen.

Als erstes widmen wir unsere Aufmerksamkeit der Prozedur `Combine` (siehe Abbildung 2.5), da sie in allen Teilprozeduren des Algorithmus verwendet wird. Sie erledigt die aktuellen Berechnungen, die in `ConstrainL`, `ConstrainR` und `AugmentR` gemacht werden, nämlich das Kombinieren zweier Stahlen. Als Eingabe bekommt die Funktion die zu kombinierenden Strahlen r_1 und r_2 übergeben. Außerdem benötigt sie die aktuelle Beschränkung a. Es wird dann ein Strahl $r^=$ errechnet, der eine Linearkombination der beiden Strahlen r_1 und r_2 ist und zudem die Beschränkung a durchdringt. Es gilt:

$$r^= = \lambda_1 r_1 + \lambda_2 r_2 \text{ und } ar^= = 0.$$

`Combine` gibt diesen Strahl als Rückgabewert an die aufrufende Prozedur zurück.

```
Procedure ConstrainL(L, a), Modifies L, Returns r_new

    Finde ein l_1 ∈ L, so daß al_1 ≠ 0 (l_1 liegt nicht auf der Beschränkung a)
    If l_1 existiert nicht Then (L ∩ H = L und r_new = ∅) Return 0.
    L' := ∅
    For alle Linien l_2 ∈ L mit l_2 ≠ l_1 Do
        L' := L'+Combine(l_1, l_2, a)
    End
    If al_1 > 0 (l_1 bestätigt die Beschränkung a) Then
        generiere Strahl r_new entsprechend zu l_1
    Else (al_1 < 0)
        generiere Strahl r_new entsprechend zu −l_1
    L := L'
    Return r_new
```

Abbildung 2.6: Prozedur zur Berechnung von `ConstrainL`(L, a)

`ConstrainL` (siehe Abbildung 2.6) ist die erste Funktion der Phase 1 und verändert den Linienraum L, indem sie ihn mit einer neuen Beschränkung a schneidet. Dafür wählt man eine beliebige Linie l_1 aus L, welche die zu berechnenden Beschränkung nicht durchdringt (d.h. nicht auf der Beschränkung liegt). Existiert keine solche Linie (d.h. alle Linien in L entsprechen der Beschränkung), so können alle in L enthaltenen Linien in den nächsten Berechnungsschritt übernommen werden und es wird kein neuer Strahl erzeugt. Der Algorithmus fährt in diesem Fall mit Phase 2 fort. Wurde allerdings eine solche Linie l_1 gefunden, so wird L von der Ungleichung a beschränkt und die Dimensionalität von L um eins vermindert, indem alle Linien in L mittels `Combine` so mit l_1 verbunden werden, daß sie die Beschränkung a durchdringen. Daraus wird ein neuer Strahl r_{new} erzeugt, der die Richtung zeigt, in der L beschränkt wurde. Als Ergebnis liefert der Algorithmus den neuen Strahl r_{new} zurück und verändert die Menge der Linien.

> *Anmerkung:* Die Funktion `Combine` ist auf Strahlen definiert, kann aber ohne Veränderung auch auf L angewandt werden. In diesem Fall wird ein Strahl mit der durch ihn definierten Linie assoziiert: ein Strahl r mit einem Richtungsvektor $\vec{v}_r$ spannt mit einem zweiten Strahl r' mit dem Richtungsvektor $-\vec{v}_r$ eine Linie auf; die Informationen für r' sind somit implizit in r enthalten. Für die verwendeten Berechnungen werden diese Informationen nicht benötigt, da sich die Berechnungen nur auf die relative Lage der Vektoren zueinander und nicht auf die absolute Lage des Strahls im Raum beziehen.

Die beiden anderen verwendeten Funktionen beeinflussen den Strahlenraum R.

Die erste, die Prozedur `AugmentR` (siehe Abbildung 2.7), verwendet dabei das Ergebnis der vorher ausgeführten Routine `ConstrainL` und fügt den dort generierten neuen Strahl

```
Procedure AugmentR(R, a, r_new), Modifies R

    R' := ∅
    For jeden Strahl r ∈ R Do
        If ar = 0 Then R' := R' + r
        If ar > 0 Then R' := R'+Combine(r, −r_new, a)
        If ar < 0 Then R' := R'+Combine(r, r_new, a)
    End
    If a ist eine Ungleichung Then R' := R' + r_new
    R := R'
```

Abbildung 2.7: Prozedur zur Berechnung von `AugmentR`(R, a, r_{new})

r_{new} dem Strahlenraum R hinzu. Sie ist die zweite Prozedur der Phase 1. Die Dimension von R erhöht sich durch das Hinzufügen des Strahls um eins.

Mit den beiden bisherigen Methoden werden nur so viele neue Strahlen hinzugefügt, wie das Polyeder Dimensionen hat. Ein Polyeder verfügt im allgemeinen aber über mehr Beschränkungen als Dimensionen.

Findet sich mit `ConstrainL` kein neuer Strahl (alle Dimensionen des Polyeders sind schon durch Strahlen des Strahlenraums dargestellt), welcher der Darstellung hinzugefügt werden kann, so muß die hinzuzufügende neue Beschränkung mit den schon bestehenden Strahlen kombiniert werden (dies entspricht der vorher erwähnten Phase 2). Dafür wird die Operation `ConstrainR` (siehe Abbildung 2.8) benötigt. Sie schränkt den Strahlenraum R mit der Beschränkung a ein. Dafür wird R mit dem Halbraum H^+ (entspricht der Ungleichung) geschnitten und der Bereich außerhalb dieser Beschränkung entfernt. Die Oberfläche des Polyeders wird danach neu berechnet. Die Strahlen, welche die von H^+ vorgegebenen Beschränkungen erfüllen, werden zu der Lösung hinzugefügt. Strahlen, die diese Bedingung nicht erfüllen, werden mit adjazenten Strahlen, die diese Bedingung erfüllen, kombiniert, um einen neuen Strahl zu generieren, welcher die Bedingung erfüllt. Für den Adjazenztest wird der entsprechende Algorithmus, der ebenfalls von Motzkin stammt, benutzt. Es werden alle Paare getestet, um diejenigen zu finden, durch deren Kombination ein neuer extremer (d.h. beschränkender) Strahl entsteht. Dafür werden erst die Bedingungen berechnet, welche von beiden Strahlen gemeinsam erfüllt werden. Dann wird sichergestellt, daß kein weiterer Strahl diese Bedingungen erfüllt. Somit ist die generierte Liste der Strahlen garantiert minimal (nicht redundant), da nur der eine jeweils neu entstande Strahl diese Menge von Bedingungen erfüllt.

2.6.2 Komplexität

Betrachten wir nun die Komplexität des Algorithmus. Ausschlaggebend für die Komplexität des gesamten Algorithmus sind die einzelnen Prozeduren, die von der Hauptroutine `Dual` aufgerufen werden.

`Procedure ConstrainR`(R, a), Modifies R

```
Partition R = R⁺ + R⁰ + R⁻
   R⁰ := {r | r ∈ R, ar = 0},
      die Strahlen, die auf der Beschränkung a liegen
   R⁺ := {r | r ∈ R, ar > 0},
      die Strahlen, die die Bedingung a erfüllen
   R⁻ := {r | r ∈ R, ar < 0},
      die Strahlen, die die Bedingung a nicht erfüllen
If Beschränkung a ist eine Ungleichung Then R' := R⁺ + R⁰
Else (die Beschränkung a ist eine Gleichung) R' := R⁰
For jeden Strahl r⁺ ∈ R⁺ Do
   For jeden Strahl r⁻ ∈ R⁻ Do
      Adjazenztest auf (r⁺, r⁻)
       c :=Menge der gemeinsamen Beschränkungen,
          die sowohl von r⁺ als auch r⁻ erfüllt werden
       For jeden Strahl r ∈ R mit r ≠ r⁺, r ≠ r⁻ Do
          If r erfüllt alle Beschränkungen in c Then
             (r⁺ und r⁻ sind nicht adjazent)
             weiter zum nächsten r⁻
       End
       (andernfalls sind r⁺ und r⁻ adjazent)
       R' := R'+Combine(r⁺, r⁻, a)
   End
End
R := R'
```

Abbildung 2.8: Prozedur zur Berechnung von `ConstrainR`(R, a)

`Combine` braucht nur einen konstanten Aufwand, da nur eine Berechnung ausgeführt wird. Der Algorithmus `ConstrainL` arbeitet linear in der Anzahl der Linien. `AugmentR` hat eine in der Anzahl der zu betrachtenden Strahlen in R lineare Komplexität. Der Algorithmus `ConstrainR` hat eine Zeitkomplexität von $O\left(n^3k\right)$, wobei n die Anzahl der Strahlen in R und k die Menge der Beschränkungen darstellen. Ein großer Teil dieser Berechnungszeit entfällt natürlich auf den Adjazenztest.

Der Zeitverbrauch des Algorithmus `Dual` ist in der Anzahl der Strahlen polynomial, da er von der Laufzeit von `ConstrainR` bestimmt wird. Die Anzahl der Strahlen allerdings ist in der Anzahl der Beschränkungen exponentiell, weshalb auch der gesamte Algorithmus einen in der Anzahl der Beschränkungen exponentiellen Aufwand benötigt.

Ausführlichere Informationen zu einer genauen Implementierung und dem Laufzeitverhalten können in [Le 92] nachgelesen werden.

3 Das Eliminationsverfahren von V. Weispfenning

3.1 Einleitung

In Kapitel 1 wurden einige grundlegende Probleme der Schleifenparallelisierung eingeführt. Die Grundlage für den dort angesprochenden Prototyp eines Schleifenparallelisierers LooPo [LfP] ist das Polytopmodell [Len93] (siehe Abschnitt 2.3). Die Polyeder sind in diesem Modell in Form von Ungleichungssystemen gespeichert. Im Laufe der Parallelisierung müssen diese Ungleichungssysteme gelöst werden, um die Schleifen, die von dem entsprechenden Polyeder beschrieben werden, aufzählen zu können. Für diese Berechnung wird der in Kapitel 2.5 vorgestellte Fourier-Motzkin-Algorithmus verwendet.

Das größte Problem des FM-Algorithmus ist die im schlimmsten Fall doppelt exponentielle Komplexität, die durch das schnelle Anwachsen der Anzahl der Schranken/Ungleichungen entsteht. Hier kommt die Idee von Weispfenning zum Tragen. Sein Ansatz ist im schlimmsten Fall einfach exponentiell. Dies bringt insbesondere bei großen Ungleichungssystemen eine signifikante Verbesserung des Laufzeitverhaltens mit sich. Die dem Ansatz zugrundeliegende Idee verbindet die Vorteile des *Divide & Conquer* mit dem Lösungsweg des FM.

Die vorgestellte Methode [Wei94] geht auf die Arbeiten V. Weispfennings zur Quantifikatorenelimination für lineare Probleme in den reellen Zahlen [Wei88] zurück. Sie wurde von Loos und Weispfenning [LW93] 1993 weiter verfeinert und von T. Sturm 1993 in Redlog [LfMA] implementiert. Die grundlegende Idee ist hier, die Lösbarkeit eines parametrischen linearen Ungleichungssystems anhand einer endlichen Auswahl parametrischer Testpunkte zu testen.

3.2 Informelle Beschreibung des Algorithmus

Wir wollen zunächst den Algorithmus informell vorstellen. Eine anschauliche Einführung bietet die Beispielrechnung im nachfolgenden Abschnitt. Der hier beschriebene Algorithmus ist eine etwas vereinfachte Version des von T. Sturm implementierten Originals. Bei der Beschreibung beschränken wir uns auf parameterfreie Ungleichungssysteme.

Ein Grundgedanke der Implementation ist die Idee eines rekursiven *Divide & Conquer*. Im Gegensatz zu FM wird das zu berechnende Polytop nicht vergrößert, sondern durch das *Divide* des Verfahrens klein gehalten. Dafür entsteht aber in jedem Eliminationsschritt eine Menge von neuen Polytopen.

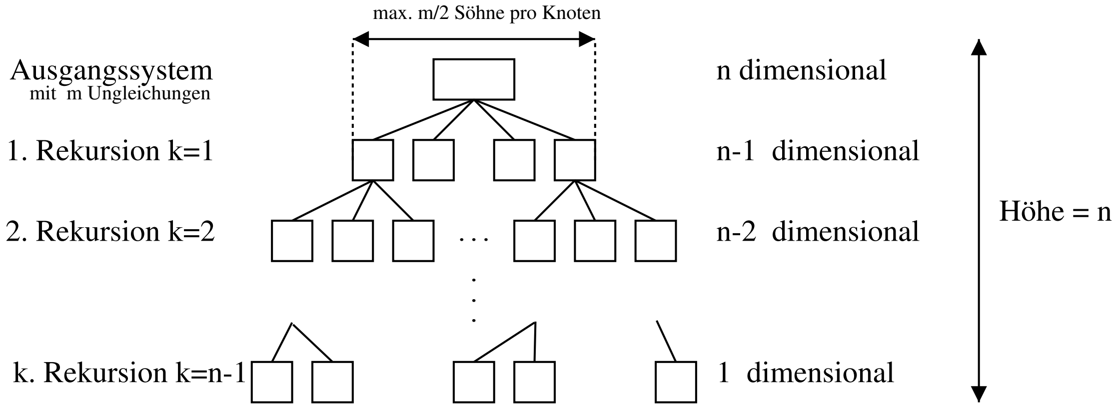


Abbildung 3.1: Rekursionsbaum

Das Polytop wird in jedem Berechnungsschritt um eine Dimension reduziert, oder genauer, vom n-dimensionalen Raum in den $n-1$ dimensionalen Raum projiziert. Die Projektion erfolgt dabei anhand von Testgeraden – den Ungleichungen, die das Polytop nach oben bzw. unten beschränken. Durch die Projektion entsteht für jede dieser Ungleichungen wieder ein neues Ungleichungssystem.

Die Ungleichungssysteme sind in der Anzahl ihrer Zeilen beschränkt: hatte das Ausgangssystem m Zeilen, so hat jedes der neu generierte System höchstens $m-1$ Zeilen. Die -1 rührt daher, daß **eine** Ungleichung für die Projektion ausgewählt wird, die allen anderen gegenübergestellt wird. Dadurch entsteht **eines** der neuen Ungleichungssysteme. Für die Anzahl der neu enstehenden Ungleichungssysteme gibt es eine klare Grenze, nämlich die Mächtigkeit der ausgewählten Menge von Schranken (obere oder untere). Bei FM hingegen wird in einem Eliminationsschritt jede Ungleichung allen anderen Ungleichungen gegenübergestellt und so statt mehrerer kleiner ein neues sehr großes Ungleichungssystem errechnet.

Um eine der beiden Mengen (obere oder untere Schranken) zur Projektion auszuwählen, vergleicht man die beiden Mengen miteinander. Zur Bestimmung der besten Auswahl werden von V. Weispfenning außer der einfachsten Möglichkeit (der Bestimmung der Mächtigkeit der einzelnen Mengen) noch weitere Methoden vorgeschlagen. Im weiteren verwenden wir diese einfachste Methode und wählen die Menge mit der geringeren Mächtigkeit für die weitere Bearbeitung aus. Die gewählte Menge hat deshalb maximal $\frac{m}{2}$ Elemente (falls beide Mengen gleich groß sind). Pro Reduktionsschritt entstehen somit höchstens $\frac{m}{2}$ neue Systeme, da jede der Schranken genau ein neues System generiert. Die Anzahl der Rekursionen ist auf die Anzahl der Variablen (also n) beschränkt. Abbildung 3.1 zeigt einen durch das *Divide & Conquer* aufgespannten Baum. Die Höhe wird durch die Anzahl der Variablen n beschränkt. Jeder Knoten hat maximal $\frac{m}{2}$ Söhne. Die maximale Breite das Baumes beträgt also höchstens $\left(\frac{m}{2}\right)^n$.

Für ein n-dimensionales Ungleichungssystem in Normalform, wie es in LooPo verwendet wird, geht der Algorithmus nun wie folgt vor:

1. Wähle die zu eliminierende Variable x_i aus; wir beginnen mit der innersten Schleifenvariable, die in unserem Ungleichungssystem von x_n repräsentiert wird.
2. Reduziere den Koeffizienten a_i in jeder Zeile j durch Division, so daß $a_i = 1$ ist. Ungleichungen, deren Koeffizient $a_i = 0$ ist, werden ignoriert.
3. Bestimme die Ungleichungen, die x_i nach oben bzw. unten begrenzen.
4. Wähle eine der beiden Mengen als Projektionskandidaten aus.
5. Stelle jedes Element der ausgewählten Menge allen anderen Ungleichungen des Ausgangssystems gegenüber (Projektion). Es entsteht für jede Schranke ein neues Ungleichungssystem, in dem x_i reduziert ist.
6. Wiederhole für jedes entstandene Ungleichungssystem die Schritte 1 bis 5, wobei $i = i - 1$ sei, bis alle Varaiblen betrachtet wurden.

Im letzten Rekursionsschritt liegen dann nur noch eindimensionale Ungleichungssysteme vor. Hier findet sich somit eine mögliche Lösung für die äußerste Variable des Polyeders. Eine mögliche Lösung für alle anderen Variablen kann daraus durch Rücksubstitution gewonnen werden. Liefert ein Zweig kein Ergebnis, so wird er nicht weiter betrachtet.

Eine einfache Implementation des Algorithmus generiert bildlich einen Baum von Ungleichungen. Jeder Knoten dieses Baumes besteht aus einem Ungleichungssystem und hat wegen der maximalen Anzahl der Projektionskandidaten maximal $\frac{m}{2}$ Söhne. Ein Pfad von der Wurzel bis zum Blatt beschreibt eine mögliche Lösung des Ungleichungssystems. Existiert kein solcher Pfad, dann ist das System unlösbar.

3.2.1 Ein rechnerisches Beispiel

Hier soll nun anhand eines kleinen Beispiels die Funktionsweise erklärt werden.

Gegeben sei ein parameterfreies lineares Ungleichungssystem u mit drei Variablen (x_1, x_2 und x_3), einer Konstanten (c) und zehn Ungleichungen in Normalform:

$$\begin{array}{rrrrrcl}
1) & x_1 & +x_2 & & -5 & \geq & 0 \\
2) & & x_2 & & -1 & \geq & 0 \\
3) & -x_1 & +x_2 & & +5 & \geq & 0 \\
4) & -x_1 & -x_2 & & +11 & \geq & 0 \\
5) & x_1 & -x_2 & & +1 & \geq & 0 \\
6) & & x_2 & +2x_3 & -7 & \geq & 0 \\
7) & & x_2 & -2x_3 & +11 & \geq & 0 \\
8) & & -x_2 & -3x_3 & +24 & \geq & 0 \\
9) & & -x_2 & +3x_3 & -3 & \geq & 0 \\
10) & & -x_2 & & +6 & \geq & 0
\end{array} \tag{3.1}$$

welches dem Polyeder aus Abbildung 3.2 entspricht. Für die Abbildung wurde zwecks

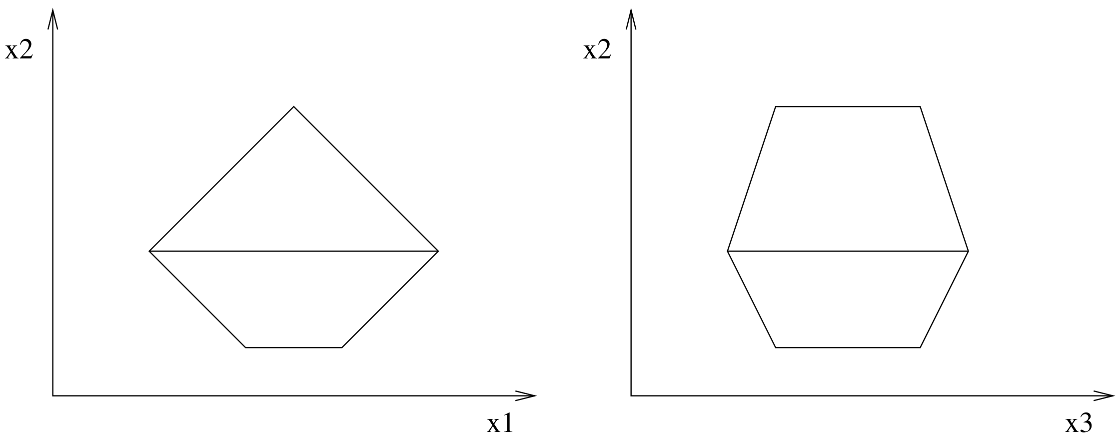


Abbildung 3.2: Polyeder für das Ungleichungssystem 3.1, dargestellt in den Dimensionsachsen x_1 und x_2 bzw. x_2 und x_3.

besserer Übersichtlichkeit eine Darstellung über zwei Abbilder in jeweils zwei der drei Dimensionsachsen gewählt. In einer Darstellung in homogenen Koordinaten, wie sie LooPo verwendet, würde das System in Form einer Matrix folgendermaßen aussehen:

$$\begin{pmatrix} 1 & 1 & 0 & -5 \\ 0 & 1 & 0 & -1 \\ -1 & 1 & 0 & 5 \\ -1 & -1 & 0 & 11 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & 2 & -7 \\ 0 & 1 & -2 & 11 \\ 0 & -1 & -3 & 24 \\ 0 & -1 & 3 & -3 \\ 0 & -1 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \geq 0$$

Wir wollen nun anhand dieses Beispiels den bisher nur informell eingeführten Algorithmus vorrechnen. Der besseren Übersichtlichkeit wegen haben wir ein relativ einfaches symmetrisches Polyeder gewählt. Eine weitere Besonderheit ist hier, daß dieses parallel zu den Koordinatenachsen ausgerichtet ist, womit einige Ungleichungen von x_3 bzw. x_1 unabhängig sind.

Als erstes wird das Ungleichungssystem, das uns hier schon in Normalform vorliegt, zu der zu eliminierenden Variable hin aufgelöst. Die Elimination beginnt bei diesem Algorithmus mit der innersten Variable des Ungleichungssystems. In unserem Beispiel formen wir unser Ungleichungssystem also nach x_3 um. Die Zeilen 1)–5) und 10) des Ungleichungssystems 3.1 bleiben unverändert, da sie kein x_3 enthalten. Daraus resultieren für x_3 die folgenden Schranken:

$$\begin{array}{rrrcl} a) & -\frac{x_2}{2} & +\frac{7}{2} & \leq & x_3 \\ b) & \frac{x_2}{2} & +\frac{11}{2} & \geq & x_3 \\ c) & -\frac{x_2}{3} & +\frac{24}{3} & \geq & x_3 \\ d) & \frac{x_2}{3} & +1 & \leq & x_3 \end{array} \tag{3.2}$$

Wir erhalten die in Ungleichungssystem 3.2 dargestellten unteren (a, d) und oberen Schranken (b, c) für x_3. Die Schranken lassen sich auch schneller ohne ein wirkliches Umformen erkennen. Bei oberen Schranken steht allgemein immer ein negativer Faktor vor der zu eliminierenden Variable. So läßt sich auf einen Blick erkennen, daß in System 3.1 7) und 8) die oberen Schranken und 6) und 9) die unteren Schranken für x_3 sind. Damit ergibt sich für die Menge der oberen Schranken O und die Menge der unteren Schranken U folgendes Bild:

$$\begin{aligned} O &= \left\{\frac{x_2}{2} + \frac{11}{2}, -\frac{x_2}{3} + \frac{24}{3}\right\} \\ U &= \left\{-\frac{x_2}{2} + \frac{7}{2}, \frac{x_2}{3} + 1\right\}. \end{aligned}$$

Für die weitere Abarbeitung wird eine der beiden Mengen ausgewählt. Die Elemente dieser Menge S sind die Kandidaten, anhand derer die Projektion auf die nächste Ebene (in den $n-1$-dimensionalen Raum) stattfindet. In diesem Fall haben beide Mengen O und U die gleiche Mächtigkeit, weshalb wir eine beliebige auswählen. Wir entscheiden uns für die Menge der unteren Schranken $(S = U)$. Durch die Auswahl der beiden Schranken als Testkandidat für x_3 werden zwei neue Ungleichungssysteme erzeugt. Für die erste Schranke $\left(-\frac{x_2}{2} + \frac{7}{2}\right)$ entsteht das folgende Ungleichungssystem:

$$\begin{array}{rrrrcl} 1.1) & x_1 & +x_2 & -5 & \geq & 0 \\ 1.2) & & x_2 & -1 & \geq & 0 \\ 1.3) & -x_1 & +x_2 & +5 & \geq & 0 \\ 1.4) & -x_1 & -x_2 & +11 & \geq & 0 \\ 1.5) & x_1 & -x_2 & +1 & \geq & 0 \\ 1.6) & & 0x_2 & +0 & \geq & 0 \\ 1.7) & & 2x_2 & +4 & \geq & 0 \\ 1.8) & & x_2 & +27 & \geq & 0 \\ 1.9) & & -x_2 & +3 & \geq & 0 \\ 1.10) & & -x_2 & +6 & \geq & 0 \end{array} \tag{3.3}$$

In dem neuen Ungleichungssystemen 3.3 ist die innerste Schleifenvariable eliminiert worden. Die Zeile 1.6 kann entfernt werden, da sie nur $0 \geq 0$ enthält.

Durch Anwendung der zweiten unteren Schranke $\left(\frac{x_2}{3} + 1\right)$ auf das Ungleichungssystem 3.1 wird als nächstes das folgende System erzeugt:

$$
\begin{array}{rrrrcl}
2.1) & x_1 & +x_2 & -5 & \geq & 0 \\
2.2) & & x_2 & -1 & \geq & 0 \\
2.3) & -x_1 & +x_2 & +5 & \geq & 0 \\
2.4) & -x_1 & -x_2 & +11 & \geq & 0 \\
2.5) & x_1 & -x_2 & +1 & \geq & 0 \\
2.6) & & \frac{5}{3}x_2 & -5 & \geq & 0 \\
2.7) & & \frac{1}{3}x_2 & -9 & \geq & 0 \\
2.8) & & -2x_2 & +21 & \geq & 0 \\
2.9) & & 0x_2 & +0 & \geq & 0 \\
2.10) & & -x_2 & +6 & \geq & 0
\end{array}
\tag{3.4}
$$

Hier kann die Zeile 2.9 entfernt werden, da sie nur 0 enthält.

Im nächsten Schritt wollen wir x_2 eliminieren. Wir müssen dafür wieder in den beiden neu entstandenen Ungleichungssystemen die oberen und unteren Schranken bestimmen. Anhand der Vorzeichen erkennen wir, daß im Ungleichungssystem 3.3 die Zeilen 1.1–1.3, 1.7 und 1.8 die unteren Schranken darstellen. Demnach sind 1.4, 1.5, 1.9 und 1.10 die oberen Schranken. Auch im zweiten neu entstandenen System 3.4 bestimmen wir gleich die oberen (2.4, 2.5, 2.8, 2.10) und die unteren (2.1–2.3, 2.6, 2.7) Schranken. So können wir abschätzen, daß in beiden Fällen die Menge der oberen Schranken eine geringere Mächtigkeit aufweist und wählen sie als die neue Menge von Testkandidaten aus. Natürlich ist diese Übereinstimmung zufällig und durch die Symmetrie unseres Ausgangssystems bedingt. Im allgemeinen ergeben sich für die verschiedenen Systeme auch unterschiedliche Mächtigkeiten der Mengen der unteren und oberen Schranken. Für das Ungleichungssystem 3.3 erhalten wir als Menge der Projektionsgeraden:

$$S_1 = O_1 = \{3,\, 6,\, x_1 + 1,\, -x_1 + 11\}\,. \tag{3.5}$$

Unter Verwendung dieser Schranken entstehen dann die folgenden Ungleichungssysteme:

$$\text{mit } x_2 \le 3 \quad : \quad \begin{array}{lrrcl} 1.1.1) & x_1 & -2 & \ge & 0 \\ 1.1.2) & -x_1 & +8 & \ge & 0 \\ 1.1.3) & -x_1 & +8 & \ge & 0 \\ 1.1.4) & x_1 & -2 & \ge & 0 \end{array} \tag{3.6}$$

$$\text{mit } x_2 \le 6 \quad : \quad \begin{array}{lrrcl} 1.2.1) & x_1 & +1 & \ge & 0 \\ 1.2.2) & -x_1 & +11 & \ge & 0 \\ 1.2.3) & -x_1 & +5 & \ge & 0 \\ 1.2.4) & x_1 & -5 & \ge & 0 \end{array} \tag{3.7}$$

$$\text{mit } x_2 \le x_1 + 1 \quad : \quad \begin{array}{lrrcl} 1.3.1) & x_1 & -2 & \ge & 0 \\ 1.3.2) & x_1 & & \ge & 0 \\ 1.3.3) & -x_1 & +5 & \ge & 0 \\ 1.3.4) & x_1 & +3 & \ge & 0 \\ 1.3.5) & x_1 & +14 & \ge & 0 \\ 1.3.6) & -x_1 & +2 & \ge & 0 \\ 1.3.7) & -x_1 & +5 & \ge & 0 \end{array} \tag{3.8}$$

$$\text{mit } x_2 \le -x_1 + 11 \quad : \quad \begin{array}{lrrcl} 1.4.1) & -x_1 & +10 & \ge & 0 \\ 1.4.2) & -x_1 & +8 & \ge & 0 \\ 1.4.3) & x_1 & -5 & \ge & 0 \\ 1.4.4) & -x_1 & +13 & \ge & 0 \\ 1.4.5) & -x_1 & +38 & \ge & 0 \\ 1.4.6) & x_1 & +8 & \ge & 0 \\ 1.4.7) & x_1 & -5 & \ge & 0 \end{array} \tag{3.9}$$

Schließlich sind wir bei der letzten Variable angelangt. Ein weiterer Rekursionsschritt ist hier nicht nötig. Nur x_1 muß noch ausgewertet werden. Da wir nur noch konstante Werte vorliegen haben, brauchen wir x_1 nur nach oben und unten abzuschätzen. Dafür wählen wir jeweils das Minimum der oberen und das Maximum der unteren Schranken. Daraus folgt:

$$\max(U_i) \le x_1 \le \min(O_i) \text{ ist eine Lösung des Ungleichungssystems.}$$

Wir erhalten also für jedes Ungleichungssystem, das sich nach *true* auflösen läßt, eine mögliche Lösung für das Ausgangssystem. In unserem Beispiel erhalten wir aus 3.6

$$2 \le x_1 \le 8,$$

aus 3.7 folgt

$$5 \le x_1 \le 5,$$

aus 3.8 bekommen wir die Lösung

$$2 \le x_1 \le 2$$

und aus 3.9

$$8 \le x_1 \le 8.$$

Die gleiche Berechnung muß nun noch für die zweite Ableitung des Ausgangssystems gemacht werden. Wir erinnern uns an das Ungleichungssystem 3.4. Auch hier müssen wir wieder die Schranken bestimmen. Beim näheren Betrachten des Systems erkennt man eine starke Ähnlichkeit zu 3.3, da alle im Ausgangssystem von x_3 unabhängigen Ungleichungen in beiden neuen Systemen vorhanden sind. Dies ist die Folge der Parallelität des Ausgangssystems zu den Koordinatenachsen. Auch hier ist die Menge der oberen Schranken wieder von geringerer Mächtigkeit und wird deshalb für die weitere Bearbeitung ausgewählt. Ein Teil der oberen Schranken entspricht den Schranken aus 3.3 (vergleiche 3.5):

$$S_2 = O_2 = \left\{\frac{21}{2},\, 6,\, x_1+1,\, -x_1+11\right\}. \tag{3.10}$$

Wir generieren nun aus diesen Schranken und dem System 3.4 wieder die neuen von x_2 unabhängigen Ungleichungssysteme:

$$\text{mit } x_2 \leq \frac{21}{2} \quad : \quad \begin{array}{lrrcl} 2.1.1) & x_1 & +\frac{11}{2} & \geq & 0 \\ 2.1.2) & -x_1 & +\frac{31}{2} & \geq & 0 \\ 2.1.3) & -x_1 & +\frac{1}{2} & \geq & 0 \\ 2.1.4) & x_1 & -\frac{19}{2} & \geq & 0 \end{array} \tag{3.11}$$

$$\text{mit } x_2 \leq 6 \quad : \quad \begin{array}{lrrcl} 2.2.1) & x_1 & +1 & \geq & 0 \\ 2.2.2) & -x_1 & +11 & \geq & 0 \\ 2.2.3) & -x_1 & +5 & \geq & 0 \\ 2.2.4) & x_1 & -5 & \geq & 0 \end{array} \tag{3.12}$$

$$\text{mit } x_2 \leq x_1+1 \quad : \quad \begin{array}{lrrcl} 2.3.1) & x_1 & -2 & \geq & 0 \\ 2.3.2) & x_1 & & \geq & 0 \\ 2.3.3) & -x_1 & +5 & \geq & 0 \\ 2.3.4) & x_1 & -2 & \geq & 0 \\ 2.3.5) & x_1 & -26 & \geq & 0 \\ 2.3.6) & -x_1 & +\frac{19}{2} & \geq & 0 \\ 2.3.7) & -x_1 & +5 & \geq & 0 \end{array} \tag{3.13}$$

$$\text{mit } x_2 \leq -x_1+11 \quad : \quad \begin{array}{lrrcl} 2.4.1) & -x_1 & +10 & \geq & 0 \\ 2.4.2) & -x_1 & +8 & \geq & 0 \\ 2.4.3) & x_1 & -5 & \geq & 0 \\ 2.4.4) & -x_1 & +8 & \geq & 0 \\ 2.4.5) & -x_1 & -16 & \geq & 0 \\ 2.4.6) & x_1 & -\frac{1}{2} & \geq & 0 \\ 2.4.7) & x_1 & -5 & \geq & 0 \end{array} \tag{3.14}$$

Beim näheren Betrachten dieser Ungleichungssysteme sehen wir, daß sie bis auf 3.12 beim Auswerten $false$ ergeben. In diesem Falle haben wir also nur eine Lösung

$$5 \leq x_1 \leq 5.$$

Auf diese Weise wird durch die rekursive Ausführung ein Ungleichungsbaum generiert, dessen Knoten jeweils ein Ungleichungssystem enthalten. Abbildung 3.3 zeigt den

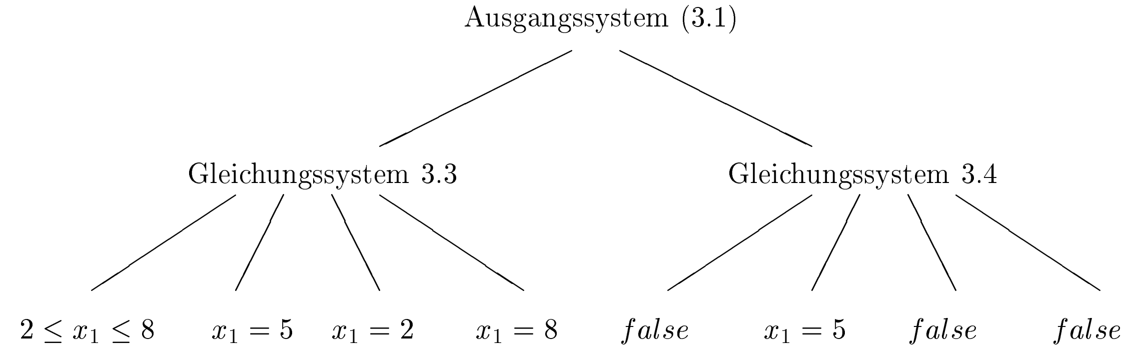


Abbildung 3.3: Lösungsbaum des Beispiels

Lösungsbaum für unser Beispiel. Jeder Sohn eines Knotens enthält ein Polyeder, das im Vergleich zu seinem Vater um eine Dimension (um eine weiter innen liegenden Schleifenvariable) verringert wurde, also vom n-dimensionalen Raum in den $n-1$-dimensionalen projiziert wurde. In jedem Projektionsschritt wird das Polyeder A mit $dim_A = n$ in m Polyeder der Dimensionalität $dim_B = n-1$ zerlegt. Dabei gilt: $m = (|U| \Leftrightarrow |U| < |O|)$ oder $m = (|U| \Leftrightarrow |O| > |U|)$. In den Blättern finden wir die Systeme, in denen nur noch die äußerste Schleifenvariable erhalten geblieben ist. Wird ein solches System vereinfacht, so erhält man eine obere bzw. untere Schranke für die äußerste Schleifenvariable. Durch Rückwärtssubstitution entlang der Knoten erhält man daraus eine Lösung des Ungleichungssystems. Jeder Pfad von der Wurzel zum Blatt beschreibt eine mögliche Lösung. Das vorgestellte Verfahren kommt also, falls das System lösbar ist, zu einer Lösung.

Der Lösungsraum des Ungleichungssystems wird im ursprünglichen Algorithmus von Weispfenning nicht betrachtet, da sich das Verfahren nur mit der Lösbarkeit eines Systems beschäftigt.

3.3 Erweiterungen

In der oben vorgestellten Form löst der Algorithmus lineare Ungleichungssysteme. Er kann aber auch auf die Berechnung quadratischer Systeme erweitert werden. Da in der Schleifenparallelisierung jedoch keine quadratischen Systeme verwendet werden, wird auf diese mögliche Erweiterung nicht eingegangen. Die theoretischen Grundlagen und Beweise für den hier vorgestellten Algorithmus und seine Erweiterung finden sich z.B. bei [Wei94].

3.4 Komplexitätsbetrachtungen

Die Komplexität des Algorithmus wird im wesentlichen durch die *Divide & Conquer* Architektur des Algorithmus bestimmt. Im Gegensatz zu Fourier-Motzkin steigt hier mit jedem Reduktionsschritt die Anzahl der zu berechnenden Ungleichungen nicht an.

Es werden zwar ähnlich zu FM auch $\frac{m}{2}$ neue Ungleichungen generiert (wobei m die Anzahl der Ungleichungen des Ungleichungssystems ist), allerdings wächst die Anzahl der Ungleichungen nicht in jedem Schritt quadratisch an. Jeder Knoten im Baum hat eine maximale Größe von $m_i = m - i$ Ungleichungen, wobei i die Tiefe der Rekursion ist.

Zeitkomplexität:

> Die Höhe des Baumes ist durch die Anzahl der Variablen, hier also die Dimension des Ungleichungssystems n, beschränkt. Dies ist gleichzeitig die maximale Rekursionstiefe des Algorithmus. Ein Berechnungsschritt des Algorithmus errechnet die Schranken eines Ungleichungssystems und seine Söhne für den nächsten Rekursionsschritt (falls es ein innerer Knoten des Baumes ist). Ein Berechnungsschritt für einen inneren Knoten im Baum benötigt dafür eine Zeit von $m - i + \left(\frac{m-i}{2}\right) \cdot (m - i - 1)$. Ein Blatt braucht zu seiner Berechnung nur maximal $\frac{m-n}{2}$ Schritte. In jeder Ebene $i > 0$ des Baumes müssen höchstens $\left(\frac{m-i+1}{2}\right)^i$ Knoten berechnet werden. Es sind somit maximal $\left(\frac{m}{2}\right)^n \left(\frac{m-n}{2}\right) + \sum_{i=1}^{n} \left(\frac{m-i+1}{2}\right)^i \cdot \left(m - i + \frac{m-i}{2} \cdot (m - i)\right)$ Berechnungsschritte zum Erzeugen der Knoten und Blätter des Baumes erforderlich.
>
> Die Zeitkomplexität ist demnach einfach exponentiell.

Speicherkomplexität:

> In jeder Ebene, also in jedem Eliminationsschritt, werden im schlimmsten Falle $\frac{m}{2}$ neue Ungleichungssysteme erzeugt. Da durch die Zerlegung aber jedes neu generierte Ungleichungssystem maximal $m - 1$ Ungleichungen enthalten kann, ergibt sich für die neu generierte Ebene ein Speicherbedarf von $\left(\frac{m}{2}\right) \cdot (m - 1)$. Der Speicherplatzbedarf einer Berechnung eines Ungleichungssystems der Problemgröße n und der Mächtigkeit m läßt sich nach oben mit $m + \sum_{i=1}^{n} \left(\frac{m-i+1}{2}\right)^i (m - i)$ abschätzen.
>
> Es ergibt sich demnach eine Speicherkomplexität von $O(m^n)$.
>
> Die Speicherkomplexität kann noch in Hinblick auf das reine Testen der Lösbarkeit optimiert werden. Dabei werden nicht alle Ungleichungssysteme im Baum gespeichert, sondern nur die Pfade von der Wurzel zum Blatt, die tatsächlich zu einer Lösung führen.

3.5 Umsetzung in LooPo

Das von Weispfenning entwickelte Verfahren und der dazugehörige Algorithmus von Kappert leisten mehr als wir für eine Integration in LooPo brauchen. Das Quantoreneliminationsverfahren ist für alle Arten von Ungleichungssystemen geeignet. Es lassen sich parameterfreie, schwach parametrisch und auch parametrische Ungleichungen lösen. Es beschränkt sich nicht auf lineare Ungleichungssysteme, sondern ist auch auf nicht linearen – quadratischen – Ungleichungssystemen anwendbar. Die von uns in der Schleifenparallelisierung verwendeten Ungleichungssysteme sind in aller Regel schwach parametrisch und linear. Dadurch vereinfacht sich eine Umsetzung des Verfahrens erheblich. Wir wollten eine möglichst einfache, schlanke und effiziente Lösung implementieren, die auf unsere Erfordernisse ausgerichtet ist.

Ein sich ergebendes Problem ist der etwas andere Ansatz des Quantoreneliminationsverfahrens. Eigentlich wird vom Verfahren von Weispfenning die Lösbarkeit eines Ungleichungssystems überprüft und eine Menge von möglichen Lösungen berechnet. Wenn die Rekursion bis zum Blatt des Baumes in der untersten Ebene absteigen kann, ist eine mögliche Lösung gefunden. Um den Algorithmus aber in LooPo verwenden zu können, brauchen wir als Ergebnis alle möglichen Lösungen, also den gesamten Lösungsraum. Die Sicherstellung einzelner Lösungen erfüllt unsere Forderung nicht. Wir müssen also im Rahmen dieser Arbeit den Algorithmus so erweitern, daß er uns nicht nur die Einzellösungen, sondern den Lösungsraum liefert.

Aus Effizienzgründen verzichtet Weispfennings Eliminationsverfahren in seiner Implementierung auf eine vollständige Speicherung aller in der Rekursion gesammelten Informationen. Der exponentielle Speicherverbrauch kann so verringert werden.

Eine Speicherung dieser Informationen ist allerdings für eine Verwendung in LooPo notwendig. Darum wird beim Eliminieren der entstehende Baum vollständig in einer Datenstruktur gehalten. Das wird möglich, da die Problemgrößen von LooPo dies noch mit einem vertretbaren Speicheraufwand erlauben und keine Steigerung des Speicherverbrauchs im Vergleich zu den schon benutzten Implementationen (FM) festgestellt werden kann.

Unser Ausgangssystem stellt einen n-dimensionalen Polyeder dar. Nach der ersten Rekursion entsteht eine Menge von Projektionen in den $n-1$-dimensionalen Raum. Diese Projektionen sind entlang der nach oben oder unten beschränkenden Hyperebenen des Ausgangspolyeders entstanden. Dabei wird bildlich jede der Projektionsebenen mit allen anderen geschnitten. Es entsteht durch die direkt angrenzenden Hyperebenen wieder ein Polyeder sowie zusätzliche Informationen, die durch den Schnitt mit nicht angrenzenden Hyperebenen erzeugt wurden. Diese sind für die Beschreibung des neu entstandenen Polyeders redundant. Alle in einer Ebene k generierten Projektionskörper zusammen beschreiben das Projektionsbild des ursprünglichen Polyeders aus der Ebene $k+1$. Das Projektionsbild eines Polyeders ist wieder ein Polyeder. Die Vereinigung der entstandenen Projektionsbilder entspricht der Projektion des Ausgangspolyeders.

Vergleichen wir unsere Knoten mit den Informationen, die FM für seine Lösung speichert. In einem Eliminationsschritt werden bei FM die berechneten Schranken für die zu eliminierende Variable x_i in die Lösung übernommen. Bei dem hier vorgestellten Eliminationsverfahren bleiben die vereinfachten Ungleichungssysteme (der Koeffizient a_i von x_i ist 1) in jedem Knoten gespeichert.

Im Gegensatz zu FM ist damit der Lösungsraum über den ganzen Baum verteilt.

Betrachten wir nun ein einzelnes Polyeder in einer Ebene des Baumes genauer. In einer Ebene k sind die Ungleichungssysteme der Dimensionalität l abgelegt, die ein l-dimensionales Polyeder beschreiben. Die Ungleichungen sind durch Projektion entlang der $(l+1)$-ten Achse und einer begrenzenden Ungleichung aus ihrem Vaterknoten ent-

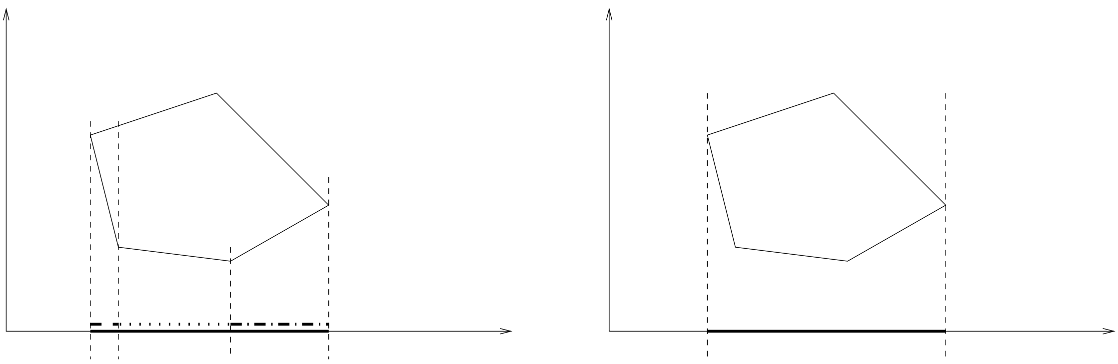


Abbildung 3.4: Projektion eines Polyeders

standen. Jedes der Projektionsbilder des Vaterknotens ist wieder ein Polyeder, beschreibt aber nur einen projizierten Ausschnitt des ursprünglichen Polyeders. Bei einer Projektion aller begrenzenden Ungleichungen entsteht ein vollständiges Bild des Polyeders aus den Vaterknoten. Betrachten wir nun einzeln alle Teilbilder des Polyeders aus dem Vaterknoten, so liefern sie zusammengenommen das Projektionsbild des Vaterknotens (siehe Abbildung 3.4). Allerdings entstehen bei einer Projektion natürlich auch redundante Informationen, da bei der Projektion alle Schnittpunkte der Testgeraden mit allen anderen Geraden des Polyeders berechnet werden (siehe Abbildung 3.5). Bezüglich der Testgeraden p entstehen 4 Schnittpunkte $(x_1 \cdots x_4)$. Davon sind x_1 und x_2 die unteren und x_3 und x_4 die oberen Schranken für das neu entstanden 1-dimensionale Polyeder. Folgende Ungleichungen entstehen:

$$\begin{array}{rcl} x & \leq & x_4 \\ x & \leq & x_3 \\ x & \geq & x_2 \\ x & \geq & x_1. \end{array}$$

Da aber $x_1 \leq x_2$ und $x_3 \leq x_4$ gilt, sind die zwei Ungleichungen $x \geq x_1$ und $x \leq x_4$ redundant.

Nur die beiden inneren Schnittpunkte beinhalten also die gewünschte Information. Die restlichen Ergebnisse der Projektion sind im Ungleichungssystem redundant.

Das vollständige Projektionsbild des Ausgangspolyeders kann durch eine Vereinigung der einzelnen Projektionen berechnet werden. Dafür ist es allerdings nötig, daß vorher die redundanten Informationen der Einzelpolyeder eliminiert werden.

Eine Redundanzüberprüfung für ein Ungleichungssystem ist sehr aufwendig, weshalb sie aus Laufzeitgründen eigentlich nicht praktikabel ist. Die Berechnung des Lösungsraums erwies sich bei der Implementierung als das größte Problem. Es wurden verschiedene Lösungswege versucht. Jeder Lösungsweg brachte jedoch immer wieder neue Probleme mit sich. Entweder erwies sich die Berechnung als sehr aufwendig, oder der

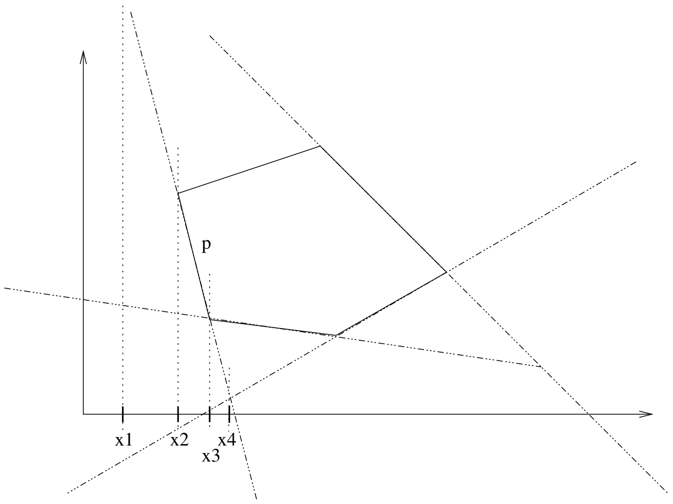


Abbildung 3.5: Redundante Informationen, die bei der Projektion entstehen

Speicherbedarf wurde zu groß, um den Algorithmus noch effizient einsetzen zu können. Schließlich beschäftigten wir uns während der Arbeit an Quilleré mit der Polylib. Es zeigte sich, daß die neue Version (im Gegensatz zu der auch bisher schon in LooPo vorhandenen älteren Fassung) an Variabilität und auch Korrektheit gewonnen hatte und deshalb nun für einen Einsatz durchaus geeignet schien. Die Idee war nun, eventuell auf dem Umweg über die duale Darstellung eine effektivere Lösung zu finden. Beim Durcharbeiten der Grundlagen und Funktionalitäten der Polylib zeigte sich, daß die für unsere Zwecke nötigen Funktionen enthalten waren. Nach Umwandlung des Ungleichungssystems (der Halbraumdarstellung) in die duale Darstellung mittels Scheitelpunkten und Strahlen durch den Algorithmus von Chernikova sind keine redundanten Beschränkungen in der Polyederdarstellung vorhanden. Eine Vereinigung der Ungleichungssysteme ist nun problemlos durchführbar. Die Vereinigung kann also problemlos über Funktionen der Polylib berechnet werden (siehe dazu auch [Wil93, Kap. 4]).

Mit dieser Erweiterung erhalten wir in unserer prototypischen Implementierung als Lösung für das Ungleichungssystem nicht nur eine Menge von Einzellösungen, sondern den gewünschten Lösungsraum.

Ursprünglich war auch eine Erweiterung auf parametrische Ungleichungssysteme geplant, jedoch wurden die Probleme durch das nötige „Einsammeln" der Lösung im Baum so groß, daß an eine effiziente Implementierung nicht mehr zu denken war.

3.6 Zusammenfassung

In den vorherigen Abschnitten haben wir uns ausführlich mit dem Eliminationsverfahren von Volker Weispfenning beschäftigt. Wir haben gesehen, daß das Verfahren in seiner Funktionalität viel mächtiger und universeller einsetzbar ist, als wir es in der Schleifenparallelisierung benötigen. Umgekehrt mußten wir feststellen, daß die Lösung, die uns das Verfahren liefert, nicht ganz unseren Erwartungen entspricht. Es liefert uns als Ergebnis einer Berechnung nicht den Lösungsraum des Ungleichungssystems, sondern nur eine nicht unbedingt vollständige Menge von Einzellösungen. Dieses Problem haben wir mit der soeben vorgestellten Methode und mit Hilfe der Polylib lösen können.

Wir konnten aber auch feststellen, daß wir im Vergleich zum Eliminationsverfahren von Fourier und Motzkin eine gewaltige Verbesserung der Komplexität des Verfahrens vorfinden. Das Lösen eines Ungleichungssystems hat keine doppelt exponentielle Laufzeit mehr, sondern kann schon in einfach exponentieller Zeit berechnet werden. Angesichts der Vorteile, die uns aus dieser Verbesserung in der Praxis erwachsen, erscheint eine Anwendung des Verfahrens mit den im letzten Abschnitt besprochenen Erweiterungen sinnvoll.

4 Die Schleifengenerierung von F. Quilleré

4.1 Einleitung

Fabien Quilleré arbeitete zusammen mit Sanjay Rajopadhye und Doran Wilde ebenfalls an einer Verbesserung der Lösung linearer Ungleichungssysteme [QRW99]. Allerdings handelt es sich hierbei nicht um ein reines Eliminationsverfahren, wie das im vorangegangenen Kapitel beschriebene.

Das hier vorgestellte Verfahren zielt auf die Aufzählung des durch mehrere Polyeder dargestellten Raumes. Die Polyeder werden durch Ungleichungssysteme repräsentiert. Diese müssen für eine Aufzählung des Polyeders aufgelöst werden, so daß wir die Lösung für das Ungleichungssystem erhalten. Der Ansatz wurde erstmalig von Le Verge, Van Dongen und Wilde 1994 [VvDW94] vorgestellt. Im Gegensatz zur Vorgehensweise von Weispfenning (ein neues Konzept zu entwickeln) bediente man sich allerdings schon funktionierender Konzepte und verknüpfte diese auf eine innovative Art und Weise. Das Verfahren baut auf der Polylib auf, in der sich Elemente des Ansatzes von Fourier-Motzkin (siehe Abschnitt 2.5) sowie der Algorithmus von Chernikova (siehe Abschnitt 2.6) wiederfinden. Bei dem hier vorgestellten Algorithmus kommt die neuste Version dieser Bibliothek zum Einsatz.

4.2 Grundlegende Notationen und Methoden

Ein Statement S, das durch einen polyhedrisch beschränkten Indexraum D beschrieben wird, wird als ein durch Domänen beschriebenes (domain-qualified) Statement bezeichnet, und wir schreiben dafür $D :: S$.

Die folgenden Statements

$$\begin{aligned} \{i, j \mid 1 \leq i \leq N; 1 \leq j \leq M\} &\;::\; S1; \\ \{i, j \mid i = j; 3 \leq j \leq N\} &\;::\; S2; \end{aligned}$$

aus Abbildung 4.1(a) können unter der Annahme, daß $4 \leq M \leq N$ ist, auch graphisch (siehe Abbildung 4.1(b)) dargestellt werden.

In unserer Notation sind die Großbuchstaben, die als Indizes auftauchen, Programmparameter; sie sind in den Domänen nicht explizit deklariert. Im Polyedermodell können symbolische Programmparameter durch zusätzliche Dimensionen der Darstellung der ein-

(a) Zwei durch einen Indexraum beschränkte Statements

$$\begin{aligned} \{i,j \mid 1 \le i \le N; 1 \le j \le M\} &\;::\; S1; \\ \{i,j \mid i = j; 3 \le j \le N\} &\;::\; S2; \end{aligned}$$

(b) die graphische Darstellung der Domäne für den Fall, daß $4 \le M \le N$

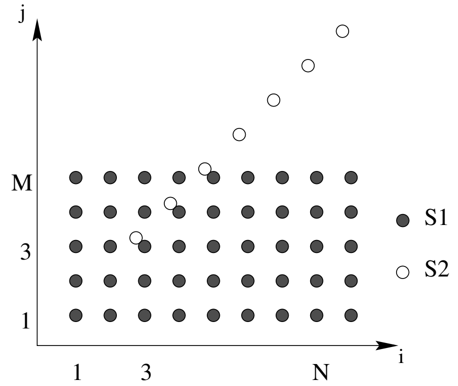


Abbildung 4.1: Darstellung und Beschreibung zweier durch Domänen beschriebenen Statements (Annahme: $4 \le M \le N$)

zelnen Statementdomänen hinzugefügt werden. Somit kann man die Domäne des Statements $S1$ aus Abbildung 4.1 entweder als parametrisierte zweidimensionale Darstellung, oder als vierdimensionales Polyeder darstellen.

Für die Darstellung von Polyedern gibt es, wie bereits in Kapitel 2.2.1 beschrieben, zwei äquivalente Möglichkeiten: erstens über eine Menge von Ungleichungen, die den Raum beschränken oder zweitens über die Generatoren, also die Scheitelpunkte und Strahlen, die das Polyeder beschreiben. Der hier vorgestellte Ansatz benutzt diese duale Darstellung, welche die Grundlage der Polylib ist, und den Algorithmus von Chernikova, um die beiden Darstellungen ineinander umzuwandeln.

Leider ist die Benutzung der dualen Darstellung nur für Polytope mit relativ kleiner Dimensionalität sinnvoll, da ein n-dimensionales Polytop bis zu $n^{\lfloor \frac{m}{2} \rfloor}$ Strahlen und Scheitelpunkte haben kann (m ist hierbei die Anzahl der Ungleichungen). Es wird jedoch angenommen, daß für den hier vorgestellten Zweck (nämlich die Berechnung von Polytopen, die verschachtelte Schleifensätze darstellen) Polytope im allgemeinen nicht von sehr großer Dimensionalität sind, da in der Praxis die Schleifen nicht zu tief verschachtelt werden.

4.2.1 Die Projektion von Polyedern

In der schon angesprochenen Polylib gibt es eine Funktion `DomainAddRays`, die eine Menge von Strahlen, Punkten oder Linien zu einer Domäne hinzufügt und die eventuell bestehenden Redundanzen entfernt.[1] Von Quilleré wird diese Funktion benutzt, um die

[1] Diese Funktion entspricht einem Teil des Algorithmus von Chernikova, der durch seinen Aufbau bei der Umwandlung der Ungleichungen die Redundanzen aus der dualen Darstellung entfernt. Allerdings wird in Chernikova diese Funktionalität für die Verfeinerung eines bestehenden Polyeders verwendet. Hier wird, wie wir sehen werden, das Polyeder vergröbert.

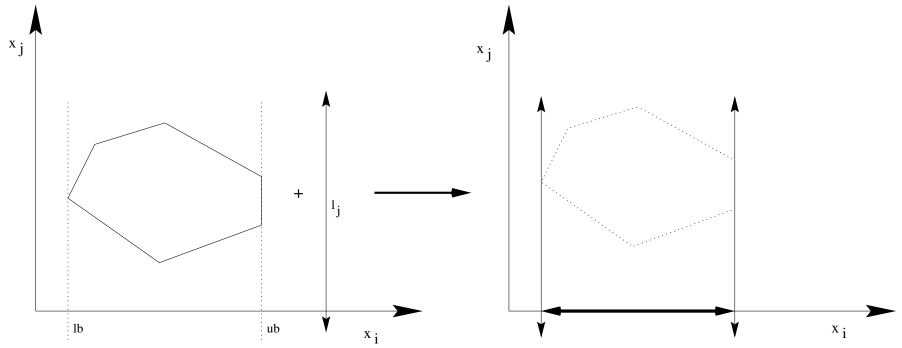


Abbildung 4.2: Projektion eines zweidimensionalen Polytops

inneren Schleifenindizes aus den Ungleichungen für die äußeren Schleifen zu entfernen. Man kann diesen Vorgang auch als ein Herausprojizieren bezeichnen, da bildlich gesehen das korrespondierende Polyeder um die herauszuprojizierende Dimension verringert wird. Genau genommen werden die Informationen für die zu eliminierende Dimension entfernt: durch die Addition einer Gerade (parallel zur herauszuprojizierenden Dimensionsachse) entsteht ein in dieser Dimensionsrichtung offener Zylinder.

In Abbildung 4.2 ist ein Polyeder mit zwei Schleifen, einem inneren Schleifenindex x_j und einem äußeren Schleifenindex x_i dargestellt. Um die Grenzen der äußeren Schleife zu berechnen, projizieren wir das Polytop entlang der x_j-Achse auf die x_i-Achse. Auf diese Weise erhalten wir die obere und untere Schranke für die äußere Variable und entfernen die Abhängigkeiten von den inneren Variablen. Diese Projektion kann mit Hilfe der schon erwähnten Funktion aus der Polylib sehr einfach berechnet werden, indem man zum Ausgangspolytop (das in einer Strahlen-Scheitelpunkt Darstellung und parallel dazu auch in der Halbraumschnittdarstellung vorliegt) die zur Koordinatenachse von x_j parallele Linie l_j addiert. Das neu entstandene Polyeder ist ein Zylinder und enthält keine Einschränkungen für die innere Variable mehr. Natürlich können auch mehrere Variablen hintereinander herausprojiziert werden, indem man die entsprechenden zur Koordinatenrichtung der dazugehörigen Variablen korrespondierenden Linien nacheinander addiert.

Allgemein kann man sagen, daß man die Schranken einer Schleife x_i berechnen kann, indem man alle inneren Variablen x_j, für die gilt: $j > i$, aus der Domäne durch eine Projektion in Richtung der x_i-Achse entfernt. Da wir hier die Polylib benutzen, sind diese Projektionen durch das Addieren der entsprechenden Linien $l_{i+1}, l_{i+2}, \ldots$ (korrespondierend zu den Richtungen der Variablen $x_{i+1}, x_{i+2}, \ldots$) und einem Aufruf des Chernikova-Algorithmus zu berechnen. Dadurch erreichen wir nach der Projektion eine minimale Beschreibung des Projektionsbildes.

4.2.2 Vereinfachung von Polyedern

Wir definieren die Vereinfachung einer Domäne D mit einer *Kontextdomäne* C wie folgt:

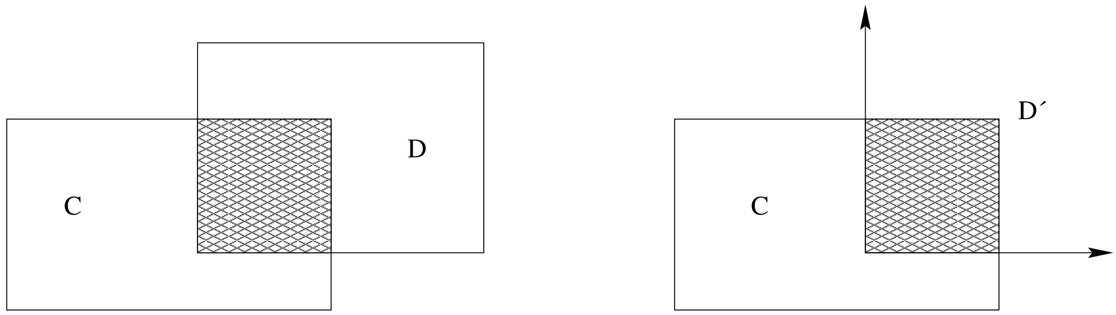


Abbildung 4.3: Vereinfachung von Domänen: $D' =$*Simplify* (D, C)

Gegeben seinen die Domänen C und D; dann ist *Simplify*$(D, C) = D'$, wenn gilt:

1. $D' \cap C = D \cap C$ mit $D' \supseteq D$ und
2. D' ist maximal in der Art, daß kein $D'' \supset D'$ existiert, so daß $D'' \cap C = D \cap C$.

Die Funktion *Simplify* liefert also die größte Domäne (oder anders ausgedrückt, die kleinste Menge von Ungleichungen), welche, wenn sie mit dem Kontext C geschnitten wird, dem Schnitt von C und D entspricht. Es werden alle Informationen aus D entfernt, die im Falle eines Schnittes mit dem Kontext C redundant werden. Die Funktion berechnet dies, indem sie den Schnitt von D mit C bildet und alle im Hinblick auf den Durchschnitt redundanten Ungleichungen von D während der Berechnung aufzeichnet. Als Resultat wird dann die Domäne D um diese „redundanten" Ungleichungen gekürzt und es entsteht D'. Dadurch werden sozusagen die Grenzen auf der von C „abgewandten" Seite entfernt.

Abbildung 4.3 zeigt dies an einem einfachen Beispiel. D' entstand aus D, indem die für den Schnitt $D \cap C$ redundanten Ungleichungen, hier die zwei, die D nach oben begrenzt haben, entfernt wurden. D' ist hier unbeschränkt.

4.3 Das Aufzählen eines einzelnen Polyeders

Nachdem wir nun einige grundlegende Voraussetzungen für dieses Kapitel geschaffen haben, können wir den Lösungsansatz Quillerés einführen.

Unser Ziel, ein Ungleichungssystem zu lösen, können wir auf verschiedene Wegen erreichen. Das im vorherigen Abschnitt vorgestellte Verfahren von V. Weispfenning war mathematisch begründet und auf viele verschiedene Arten (parametrisch, linear, nicht linear usw., siehe [Köb95] oder [Wei94]) von Ungleichungssystemen anwendbar gestaltet. Das nun folgende Verfahren zielt in seiner Projektierung schon vom Ansatz her auf die Probleme der Schleifenparallelisierung und die hier auftretenden Ungleichungssysteme ab. Es kann nur auf parametrisierte lineare Ungleichungssysteme angewandt werden.

Das Polyeder wird rekursiv (von außen nach innen) aufgezählt. Diese Methode wurde erstmals von H. Le Verge [VvDW94] vorgestellt. Standardalgorithmen für die Lösung von Ungleichungssystemen basieren auf der Elimination von FM (z.B. [AI91]). Das Verfahren von F. Quilleré verwendet dafür die duale Repräsentation von Polyedern, die in der Polylib [Wil93] implementiert ist.

4.3.1 Problemstellung: Aufzählen eines Polyeders

Zuerst wollen wir uns mit der Problemstellung beschäftigen, die einer Aufzählung zugrunde liegt. Die Aufgabe, die wir zu betrachten haben, läßt sich folgendermaßen umschreiben:

> Gegeben ist eine parametrische Domäne D. Sie hat entsprechend unserer Notation die folgende Form:
>
> $$D(p) = \{x \mid Ax \geq Bp + b\}$$
>
> und beschreibt eine Menge von Schleifengrenzen $lb_1, ub_1, \ldots, lb_n, ub_n$ für einen Schleifensatz:
>
> $lb_1 \leq x_1 \leq ub_1$
> $\quad lb_2 \leq x_2 \leq ub_2$
> $\qquad \ddots$
> $\qquad\quad lb_n \leq x_n \leq ub_n$
> $\qquad\qquad$ `Schleifenrumpf`
>
> Dieser Schleifensatz wird jeden Punkt x in der Domäne genau einmal bearbeiten und zwar in der lexikographischen Reihenfolge der Ausführungspunkte.
>
> Im generierten Code sind die durch den Algorithmus zu berechnenden Schleifengrenzen dann das Minimum oder Maximum der affinen Funktionen der äußeren Schleifenvariablen und Parameter (je nachdem, ob es sich um eine obere oder eine untere Schleife handelt).
>
> Das Berechnen der Schleifengrenzen ist ein Problem der ganzzahligen linearen Programmierung und hat deshalb deren Komplexität. Vorteilhaft für uns ist allerdings, daß unsere Problemgrößen eher klein sind. Wie schon mehrfach erwähnt, hängen sie von der in der Praxis vorkommenden Programmiermethodik ab, die Schleifen nicht zu tief ineinander schachtelt. Eine genaue Lösung für eine typische Berechnung von Schleifengrenzen läßt sich deshalb in einer in der Praxis verwendbaren Zeit lösen.

4.3.2 Grundlagen des Algorithmus

Die vorgestellte Methode für das Aufzählen eines einzelnen Polyeders arbeitet ähnlich wie die FM-Elimination. Auch hier wird das Polyeder in der Richtung seiner inneren Schleifenvariablen projiziert, um diese aus den Ungleichungen zu entfernen, so daß die

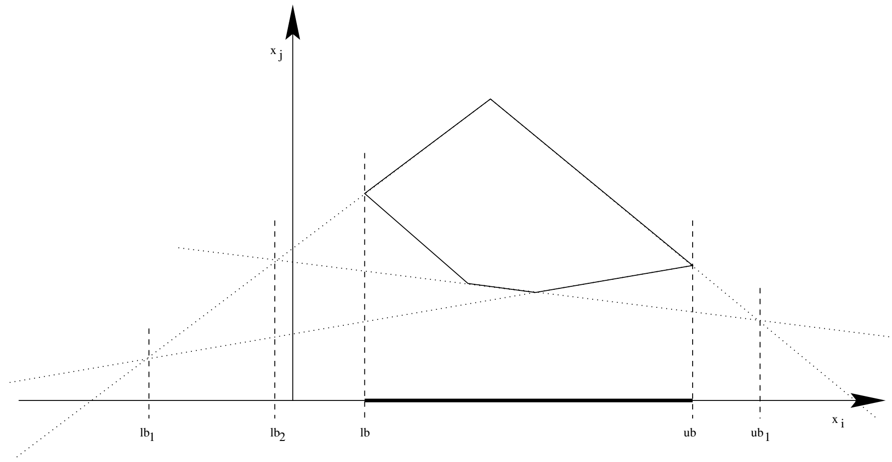


Abbildung 4.4: Beispielschranken, wie sie von FM berechnet werden

Grenzen einer Schleife unabhängig von den inneren Schleifenvariablen gefunden werden. Die Schranken lb_1, lb_2 und lb in Abbildung 4.4 sind die unteren, ub_1 und ub die oberen Schranken für das dargestellte Polyeder, so wie sie von FM generiert werden würden. Wir erinnern uns an die Einführung zu FM (siehe Abschnitt 2.5, speziell Abschnitt 2.5.3) und an die dort schon vorgestellte Komplexität; danach können wir generell sagen, daß bei k gegebenen Ungleichungen bis zu $\left(\frac{k}{2}\right)^2$ Schleifengrenzen bei der Elimination einer einzelnen Variable neu generiert werden. Diese Schleifengrenzen sind (insofern sie nicht die äußerste Variable betreffen) wieder Ungleichungen, die beim nächsten Berechnungsschritt erneut mit herangezogen werden müssen. Ein großer Teil dieser Schranken ist redundant. In unserem Beispiel in Abbildung 4.4 sind z.B. die Schranken lb_1, lb_2 und ub_1 überflüssig, da das Bild des Polyeder schon durch lb und ub eindeutig bestimmt ist. Sie müßten für eine wirklich effiziente Berechnung entfernt werden, was vom FM-Algorithmus allerdings nicht geleistet wird. Es entsteht eine von der Anzahl der Variablen abhängige exponentiell anwachsende Zahl von Ungleichungen, welche die Ursache für das doppelt exponentielle Laufzeitverhalten des FM-Algorithmus ist.

Im Gegensatz zu FM werden bei dem hier betrachteten Algorithmus in die Berechnung der Schranken nicht alle Ungleichungen, sondern nur Paare von adjazenten (also zusammenhängenden) Ungleichungen, einbezogen. Zusammenhängend sind dabei die Ungleichungen, die im korrespondierenden Polyeder benachbarte Begrenzungen beschreiben. Dadurch werden die redundanten Schranken gar nicht erst erzeugt.

Notwendig ist dadurch allerdings ein Adjazenztest auf den Ungleichungen. Dies wird uns durch die verwendete duale Darstellung der Polyeder erleichtert, da die Strahlen und Scheitelpunkte des Polyeders bekannt sind. Zwei zusammenhängende Beschränkungen[2]

[2]Ein Polyeder wird in der von uns bisher verwendeten Halbraumschnittdarstellung durch Ungleichungen beschränkt. Es wird deshalb bisher zur besseren Veranschaulichung oft einfach der Terminus „Ungleichung" benutzt, wenn eine Beschränkung des Polyeders gemeint war. Da wir nun aber mit der

des Polyeders sind damit einfach zu finden, sie haben einen gemeinsamen Schnittpunkt.

4.3.3 Die Synthese der Schleifensätze

Um die Schleifen eines durch ein Polyeder beschriebenen Schleifennests generieren zu können, benötigt man die Grenzen der einzelnen Schleifen in allen Dimensionen des Polyeders. Diese Schleifengrenzen sind somit auch eine Lösung des Ungleichungssystems.

Für die Synthese der einzelnen Schleifen aus der Ursprungsdomäne D benutzt Quilleré einen rekursiven Algorithmus, der die Domäne D in eine Anzahl von Schleifendomänen $D_1, D_2, \dots$ und eine initiale Kontextdomäne C_0 zerlegt.

Die Zerlegung der Domäne erfüllt dabei die folgenden drei Bedingungen:

- $D = C_0 \cap D_1 \cap \cdots \cap D_n$.
- Jede Schleifendomäne D_i enthält keinerlei Abhängigkeiten zu inneren Schleifenvariablen $(x_{i+1}, x_{i+2}, \dots)$; bildlich heißt das, daß der Körper in diesen Dimensionen[3] einen Zylinder beschreibt; damit ist er durch eine Projektion auf die äußeren i Variablen vollständig beschrieben.
- Die linearen Ungleichungen (Beschränkungen) einer jeden Schleifendomäne D_i sind nicht redundant zu Beschränkungen, die durch äußere Schleifen beschrieben werden (Vergleiche hierzu den Abschnitt 4.2.2 über die Vereinfachung von Polyedern am Anfang dieses Kapitels); der Kontext C_{i-1} einer Domäne D_i setzt sich aus den Informationen über die äußeren Schleifen zusammen:

$$C_{i-1} = C_0 \cap D_1 \cap D_2 \cap \cdots \cap D_{i-1}. \tag{4.1}$$

Der Algorithmus „zerlegt" also die Ungleichungen von D in n verschiedene Polyeder. Diese Polyeder sind zwar verschieden aber üblicherweise nicht disjunkt. Im Abschnitt 4.2.2 über die Vereinfachung von Polyedern haben wir gesehen, daß die entstehenden Polyeder durchaus auch unbeschränkt sein können. Eine Überschneidung der einzelnen Polyeder wird somit nicht vermieden, sondern durch die mögliche Unbeschränktheit der entstehenden Domänen vielmehr noch gefördert.

Die Ungleichungen, die ein solches Polyeder D_i beschreiben, sind die oberen und unteren Schranken der Schleife i. Wir haben somit nach einer Zerlegung von D in die Teildomänen $D_1, \dots, D_j$ die Schranken für jede Schleife in jeweils einer Domäne vorliegen, welche unabhängig von den weiter innen liegenden Schleifen sind.

Darstellung über die Generatoren des Polyeders arbeiten, ist diese Terminologie nicht mehr korrekt. Eine korrekte Unterscheidung wird hier notwendig um die beider Darstellungsformen im Text auch syntaktisch trennen zu können und so die Unterschiede in der Anwendung der einzelnen Hilfsmittel des Algorithmus deutlich zu machen.

[3] Gemeint sind hier die Dimensionen der inneren Schleifenvariablen.

4.3.3.1 Der Algorithmus

Wenden wir uns nun etwas genauer dem zugrundeliegenden Algorithmus zu. In jedem Rekursionsschritt i wird eine neue Domäne D_i unter Zuhilfenahme des Kontextes C_{i-1} generiert, indem die folgenden Schritte ausgeführt werden:

1. Die inneren Schleifenvariablen $x_j, j > i$ werden herausprojiziert (siehe 4.2.1).
2. Die erhaltene Domäne D_i wird in Abhängigkeit zum ihrem Kontext C (also C_{i-1}) vereinfacht (siehe Abschnitt 4.2.2).
3. Für den nächsten Rekursionschritt (die Generierung von D_{i+1}) wird der neue Kontext $C_i = C_{i-1} \cap D_i$ generiert bzw. die Rekursion abgebrochen, falls wir uns im letzten Rekursionsschritt ($i = n$) befinden.

Die Zerlegung der Ursprungsdomäne D liefert uns einen Schleifensatz, in dem jede Schleife der Tiefe i durch ein Polytop D_i beschrieben wird. Jedes dieser Polytope ist nur von äußeren Variablen abhängig, was bedeutet, daß die beschreibenden Ungleichungen nur Variablen x_j mit $j \leq i$ enthalten. D_i ist damit äquivalent zu einem Polyeder L_i der Dimensionalität i. L_i sei definiert als die Projektion von D_i auf seine äußeren i Dimensionen. Ein Schleifensatz wird im weiteren einfach mit $L_1 :: L_2 :: \cdots :: L_n :: S$ bezeichnet. In den folgenden Kapiteln werden die Bezeichnungen L_i und D_i manchmal auch synonym verwendet.

Die folgenden Schritte zeigen formal die Vorgehensweise des Algorithmus an einem beliebig dimensionalen Polyeder. Gegeben sei eine n-dimensionale Domäne D, mit x_n als innerste Schleifenvariable. Gesucht wird eine Zerlegung von D in seine Schleifendomänen $D_1, ..., D_n$ und seinen initialen Kontext C_0 wie auf Seite 48 beschrieben. Der initiale Kontext C_0 ergibt sich aus den Parametern der Domäne D. Zur seiner Berechnung projizieren wir im ersten Schritt des Algorithmus die Domäne D auf die Dimensionen der Programmparameter. Wir haben nun

$$D = C_0 \cap D'$$

berechnet und können mit der eigentlichen rekursiven Schleifenberechnung fortfahren.

Wir starten mit der Berechnung der äußersten Schleife x_1. Aus D werden in Schritt 1 die Dimensionen von $x_2, ..., x_n$ herausprojiziert. Das entstandene Polyeder P_1 ist von den inneren Variablen unabhängig. Wir erhalten

$$D = C_0 \cap P_1 \cap D'',$$

wobei P_1 nur aus Ungleichungen, die die Programmparameter und x_1 betreffen, besteht. P_1 wird nun in Schritt 2 mittels *Simplify* (siehe 4.2.2 Seite 44) und der initialen Kontextdomäne C_0 zu D_1 vereinfacht, so daß gilt:

$$P_1 \cap C_0 = C_0 \cap D_1.$$

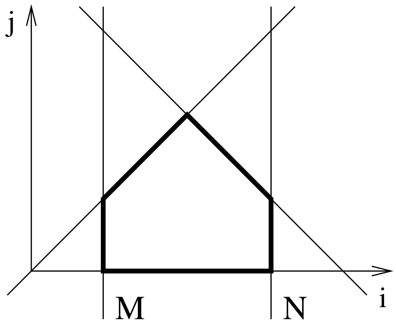


Abbildung 4.5: Darstellung der Domäne: $\{i, j \mid j \leq i \leq N;\, j \geq 0;\, i \geq M; i + j \leq N + M;\, M \geq 2;\, M < N\}$.

Damit ist

$$\begin{aligned} D &= C_0 \cap D_1 \cap D'' \text{ oder} \\ D &= C_1 \cap D'' \text{ mit } C_1 = C_0 \cap D_1 \end{aligned}$$

und wir können mit dem neuen Kontext C_1 die nächste Variable x_2 betrachten. Aus D werden nun (Schritt 1) alle Dimensionen der Variablen $x_3, \ldots, x_n$ herausprojiziert und das entstehende Polyeder mit seinem Kontext C_1 vereinfacht (Schritt 2). Wir haben nun folgende Situation:

$$\begin{aligned} D &= C_0 \cap D_1 \cap D_2 \cap D''' \text{ oder} \\ D &= C_2 \cap D''' \text{ mit } C_2 = C_1 \cap D_2 \text{ und } C_1 = C_0 \cap D_1. \end{aligned}$$

Für eine beliebige Variable x_i ergibt sich so im Rekursionsschritt i folgendes:

$$\begin{aligned} D &= C_0 \cap D_1 \cap \cdots \cap D_i \cap D'''' \text{ oder vereinfacht} \\ D &= C_{i-1} \cap D_i \cap D''''. \end{aligned}$$

Nach n Rekursionsschritten erhalten wir dann die gesuchte Zerlegung

$$D = C_0 \cap C_1 \cap \cdots \cap D_n$$

mit den geforderten Bedingungen von Seite 48.

4.3.3.2 Rechnerisches Beispiel für die Synthese

Die Vorgehensweise des vorherigen Abschnittes soll nun an einem kleinen Beispiel noch einmal praktisch vorgeführt werden. Gegeben sei die folgende Domäne D eines Programmes mit den Parametern N und M (siehe Abbildung 4.5):

$$\begin{aligned} \{i, j \mid\ & j \leq i \leq N;\, j \geq 0;\, i \geq M; \\ & i + j \leq N + M;\, M \geq 2;\, M < N\}. \end{aligned}$$

Gesucht ist die Zerlegung dieses Polytops in einen Schleifensatz wie oben beschrieben. Wir wollen also eine Zerlegung von D in D_1 und D_2 (da in D zwei Variablen vorkommen: $i \equiv x_1; j \equiv x_2$) berechnen, die folgendes erfüllt:

$$D = C_0 \cap D_1 \cap D_2.$$

Um die Zerlegung berechnen zu können, brauchen wir als Erstes den initialen Kontext C_0 des gesamten Polyeders D. Der initiale Kontext C_0 ist ein Polyeder, das durch die Parameter des Ursprungsprogrammes bestimmt wird, in unserem Falle durch N und M. Wir erhalten C_0 aus der Projektion von D in die Dimensionen M und N. C_0 hat damit folgende Form:

$$\{N, M \mid M \geq 2;\ M < N\}.$$

Das so entstandene Polyeder ist unabhängig von den Programmvariablen. Mit C_0 kann man nun die Rekursion über die Anzahl der Variablen starten. Zum Generieren der ersten Schleife müssen wir nur eine äußere Schleife festlegen, die als erste abgespalten wird und im nächsten Rekursionschritt den Kontext für die weiter innen liegenden Schleifen bildet. Wir wählen hier i. Wir berechnen nun also D_1, das uns die i-te Schleife beschreibt, indem wir aus D alle weiteren Dimensionen, in diesem Fall nur die j-te, herausprojizieren (Schritt 1 des Algorithmus). Wir erhalten ein Polyeder, das dem Schnitt von D_1 und C_0 entspricht[4]:

$$\{i, j \mid M \leq i \leq N;\ M \geq 2;\ M < N\}.$$

Nach Vereinfachung mittels des Kontextes C_0 (Schritt 2) erhalten wir für D_1 und damit als Schranken für die äußere Schleife:

$$L_1 = \{i \mid M \leq i \leq N\}.$$

Im folgenden Rekursionsschritt wird die nächste Schleife generiert. Dafür benutzen wir den Kontext $C_1 = C_0 \cap D_1$. Hier wird keine Projektion mehr benötigt, da keine inneren Schleifenvariablen mehr existieren. Die Vereinfachung von D mittels des Kontextes C_1 ist ausreichend.

D_2 ist somit:

$$L_2 = \{i, j \mid 0 \leq j \leq i;\ j \leq M + N - i\}.$$

Unter der Annahme, daß $M \geq 2$ und $M < N$ ist, ergibt sich der folgende Schleifensatz:

$M \leq i \leq N$
    $0 \leq j \leq \min(i, (M + N - i))$
        `Schleifenrumpf`

[4]Dies entsteht nach der Definition der Zerlegung: $D = C_0 \cap D_1 \cap D_2$ und des Kontextes: $C_{j-1} = C_0 \cap D_1 \cap \cdots \cap D_{j-1}$ einer Domäne D_j.

4.3.4 Komplexitätsbetrachtungen

Bei näherer Betrachtung des Verfahrens zeigt sich, daß man hierbei von einem angepaßten, verbessertem FM sprechen kann. Durch Verwendung der dualen Darstellung und der Polylib konnten die größten Probleme des FM beseitigt werden. Bei der Projektion verhindert der Adjazenztest, daß sich die Anzahl der zu betrachtenden Ungleichungen exponentiell entwickelt und damit der ganze Algorithmus doppelt exponentiell wird. Erkauft wird dies zwar mit der zusätzlichen Berechnung der dualen Darstellung (exponentiell in der Anzahl der Ungleichnungen), was jedoch im Vergleich zur bisherigen Lösung mittels FM einen relativ geringen Mehraufwand darstellt. Wir werden im folgenden noch sehen, daß sich dieser Mehraufwand für die Gesamtlaufzeit des Algorithmus rechnet.

Bei der Berechnung eines n-dimensionalen Polyeders wird für jede Variable einmal die Berechnung der Zerlegung ausgeführt. Der Algorithmus wird somit n mal durchlaufen.

Den größten Aufwand in Schritt 1, der Projektion, erzeugt im wesentlichen der Aufruf des Chernikova-Algorithmus. Wie wir in Abschnitt 2.6 gesehen haben, ist der Algorithmus in der Anzahl der Strahlen polynomial. Allerdings ist die Anzahl der Strahlen in der Anzahl der Ungleichungen exponentiell. Damit ist auch der Chernikova-Algorithmus in der Anzahl der Ungleichungen exponentiell. Die gleiche Komplexität gilt deshalb auch für die Projektion einer Variable.

Die Vereinfachung (siehe Abschnitt 4.2.2) in Schritt 2 hat ebenfalls eine exponentielle Laufzeit, da auch hier bei der Berechnung des Schnittes der Chernikova-Algorithmus ausgeführt wird. *Simplify* kombiniert erst die Ungleichungen der beiden Polyeder und berechnet dann die duale Darstellung. Folglich ist die Routine exponentiell in der Summe der Ungleichungen der zu vereinfachenden Polyeder.

Da die bestimmenden Faktoren der Laufzeit des gesamten Algorithmus von den Strahlen der Polyeder und diese wiederum von der Anzahl der Ungleichungen abhängig sind, ist die Berechnung der Unbekannten mit diesem Algorithmus in der Anzahl der Ungleichungen des Ausgangssystems exponentiell. Der Algorithmus wird für jede Variable nur einmal durchlaufen. Die Gesamtlaufzeit des Algorithmus ist allerdings, durch die Exponentialität von Chernikova in jeder Berechnungsebene, ebenfalls exponentiell.

4.3.5 Zusammenfassung

Mit den bisher vorgestellten Methoden ist es möglich geworden, ein einzelnes Polyeder effektiv aufzuzählen. Das Lösen des korrespondierenden Ungleichungssystems mittels des FM ist nicht mehr nötig. In der Methode ist ein Algorithmus integriert, der von FM abgeleitet ist und das Auflösen des Ungleichungssystems übernimmt. Dadurch eignet sich die Methode auch als Lösungsverfahren für einfache schwachparametrische lineare Ungleichungssysteme. Sie ist auch unter dem Gesichtspunkt der Effizienzsteigerung bei der Parallelisierung interessant, da die Komplexität nur einfach exponentiell in der Anzahl der Ungleichungen ist.

4.4 Das Aufzählen mehrerer Polyeder

In den vorangegangenen Abschnitten dieser Arbeit haben wir uns mit der Problematik des Aufzählens einzelner durch Ungleichungssysteme dargestellter Polyeder beschäftigt. Im weiteren soll Quillerés Erweiterung des gerade vorgestellten Algorithmus zur Synthese eines einzelnen Polyeders betrachtet werden. Wir zählen nun nicht mehr nur ein Polyeder auf, sondern eine Menge von Ungleichungssystemen, die sich auch überschneiden können. Als erstes beschäftigen wir uns deshalb mit der damit verbundenen Problematik und den bisherigen Lösungsansätzen. Im weiteren wird dann der von bisherigen Verfahren abweichende Ansatz von Quilleré näher erläutert.

4.4.1 Eine allgemeine Einführung

Ein grundlegendes Problem für das Überführen imperativer Schleifenprogramme in ihre parallelen Pendants ist die schon in der Einleitung angesprochene Berechnung einer Reihenfolge der auszuführenden Berechnungsschritte. Damit eine korrekte Aufzählung möglich ist, ist es unbedingt notwendig, miteinander verknüpfte (abhängige) Schleifeninstanzen zu ordnen. Dadurch ergibt sich eine eindeutige Reihenfolge der Berechnungsschritte.

Solange man ein einzelnes Polyeder betrachtet, ist die Berechnung einer effizienten Aufzählung der einzelnen Schleifeninstanzen unkompliziert. Ein solcher Schleifensatz wird über ein einzelnes Ungleichungssystem dargestellt. Eine Ordnung der Instanzen ist in diesem Falle schon durch die Schachtelung der Schleifen gegeben. Lediglich die Abhängigkeiten der inneren Variablen des Ungleichungssystems von den äußeren müssen hier berechnet werden. Dies kann zum Beispiel durch Lösen des Systems, wie im Kapitel 4.3 gezeigt, geschehen.

Komplizierter wird die Bearbeitung von Schleifensätzen, falls diese nicht perfekt verschachtelt sind. Solch eine Konstellation wird durch mehrere (sich möglicherweise überlappende) Polyeder dargestellt. Jedes dieser Polyeder beschreibt genau den Problemraum eines Schleifensatzes. Bei den bisherigen Methoden versucht man, die Lösung eines solchen Problems[5] über eine Vereinfachung zu erreichen. Es wird ein Schleifensatz generiert, der die bisherigen umhüllt (siehe Abbildung 4.6a). Die neue äußere Schleife liefert nun die Abhängigkeiten für die inneren Schleifen und damit eine Ordnungsrelation. Der Nachteil dieser Lösung ist offensichtlich: es werden viel mehr Instanzen aufgezählt als eigentlich ausgeführt werden dürfen, da auch Bereiche, die eigentlich nicht in unserem Berechnungsraum liegen, durch die umschließende Schleife mit erfaßt werden. Genau an dieser Stelle setzt Quillerés Verbesserungsvorschlag an. Um einen solchen Mehraufwand zu vermeiden, wird der Raum in kleinere Teilbereiche zerlegt. Die Zerlegung geschieht immer entlang der Schnittkanten der einzelnen Polyeder. Es enstehen also – betrachten wir wieder das Beispiel in Abbildung 4.6b – neue Polyeder, die jeweils die Probleme aus den Polyedern A, $A \cap B$, $A \cap C$, B, $B \cap C$, C und $A \cap B \cap C$ enthalten. Jetzt müssen noch die einzelnen von einander unabhängigen neu entstandenen Polyeder in eine Reihenfolge

[5] Die derzeit in LooPo verwendeten Methoden werden im Abschnitt 4.4.2 genauer untersucht.

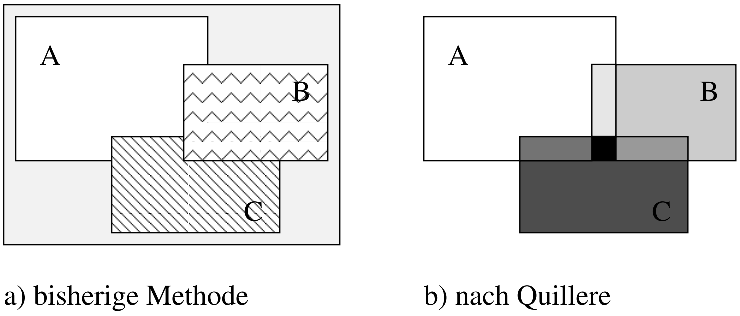


Abbildung 4.6: Verschiedene Ansätze für Verschachtelung

gebracht werden. Die nächsten Abschnitte werden die Zusammenhänge und die nötigen Berechnungen näher erläutern.

4.4.2 Problem des Aufzählens mehrerer Polyeder

Bisher haben wir uns mit der Synthese einzelner Statements beschäftigt, die jeweils in einem einzelnen Polyeder beschrieben waren. Nun erweitern wir das Problem auf mehrere Statements, also mehrere Polyeder. Wie bisher angenommen entspricht ein Polyeder der Darstellung der Instanzen eines Statements. Allerdings bestehen im Falle mehrerer Statements im allgemeinen Abhängigkeiten zwischen den einzelnen Instanzen verschiedener Statements und damit zwischen den einzelnen Schleifen verschiedener Polyeder. Dadurch ergeben sich Beschränkungen für die Reihenfolge der Abarbeitung. Um eine Vereinigung mehrerer solcher Domänen korrekt lösen zu können, ist die Reihenfolge der Aufzählung entscheidend.

Betrachten wir das Problem, das es zu lösen gilt. Bei der Parallelisierung zeichnet sich für uns folgendes Bild ab:

> Ein **Schedule** hängt an jeden Indexvektor jeder Domäne ein logisches Datum an. Wir nehmen an, daß dieses Datum ein k-dimensionaler Vektor[6] und eine affine Funktion Λ des Indexvektors ist.
>
> Die **Ordnung**, in der wir aufzählen müssen, entspricht einer lexikographischen Ordnung der logischen Ausführungspunkte der einzelnen Abarbeitungsschritte. Eine solche strikte Ordnung $\prec_l$, die für unsere Zwecke notwendig ist, haben wir bereits in Abschnitt 2.1.1 definiert.
>
> Aus diesen Betrachtungen folgt, daß die einzelnen Domänen nicht voneinander unabhängig aufgezählt werden können. Dies sei an einem kleinen Beispiel verdeutlicht:

[6] k ist für alle Domänen gleich.

> Gegeben seien zwei Schleifendomänen D_1 und D_2 mit der Ausprägung $D_1 = \{i \mid 1 \leq i \leq 10\}$ und $D_2 = \{i \mid 4 \leq i \leq 12\}$ sowie den dazugehörigen Schedules Λ_1 und Λ_2 mit $\Lambda_1(i) = i + 1$ und $\Lambda_2(i) = 2i$. Wenn wir nun die Statements betrachten, die zu $i = 1$ bzw. $i = 10$ von D_1 gehören, so sehen wir, daß sie wegen Λ_1 zur Zeit 2 bzw. 11 ausgeführt werden. Im Gegensatz dazu wird z.B. das Statement von D_2 mit $i = 4$ zur Zeit 8 und mit $i = 8$ zur Zeit 16 ausgeführt. Da $2 < 8 < 11 < 16$ ist, ergibt sich eine Verzahnung der Ausführungsreihenfolgen von D_1 und D_2 und wir sehen, daß wir unter den gegebenen Umständen keine korrekte Aufzählung finden können, in der entweder D_1 vor D_2 oder D_2 vor D_1, vollständig aufgezählt wird. Keine der Reihenfolgen $D_1; D_2$ oder $D_2; D_1$ erhält eine korrekte zeitliche Ordnung der betrachteten Daten! Die beim Aufzählen ermittelten Schleifen müssen vermischt werden, um die korrekte Ausführungsreihenfolge der Statements, die durch die zeitliche Ordnung gegeben ist, sicherzustellen.

In unserem Beispiel haben wir gesehen, daß wir mehrere Domänen nicht voneinander unabhängig betrachten können, sondern auch deren Zusammenspiel beachten müssen.

Eine allgemeine Lösung, die zur Übersetzungszeit die gültige Aufzählung der Schleifendomänen finden soll, müßte die Vereinigung aller Domänen in einer lexikographischen Reihenfolge aufzählen und dann die jeweiligen zugehörigen Statements ausführen. Die einfachste Lösung für die Aufzählung einer solchen Vereinigung ist es, eine Schleife um alle Domänen zu legen. Eine solche Schleife bildet eine konvexe Obermenge der Vereinigung (bildlich eine umschließende Box) und kann dann einfach aufgezählt werden. Dies führt allerdings dazu, daß diese herumgelegte Hülle auch eine Obermenge für jedes einzelne Statement der vereinigten Domänen bildet. Deshalb muß jedes Statement mit zusätzlichen Bedingungen versehen werden, welche testen sollen, ob zu diesem Zeitpunkt ein Statement tatsächlich auszuführen ist oder nicht. Eine solche Bedingung wird als *Guard* bezeichnet. Guards sind, wie wir später noch sehen werden, einer der Gründe für das schlechte Laufzeitverhalten des auf diesem Wege generierten Codes.

In LooPo wurde dieses Verfahren bisher angewandt (siehe [Wet95, Kap. 4.4.2.1]). Es wird dort als *Runtime*-Lösung bezeichnet, da die Entscheidung, wann ein Statement auszuführen ist, zur Laufzeit des Programmes getroffen wird. Eine ähnliche Lösung verfolgt der Omega Code Generator vom W. Kelly ([KPR95]). Allerdings wird hier eine möglichst kleine konvexe Hülle der Domänen generiert. In Abbildung 4.7 kann man den jeweils erzeugten Code der beiden Verfahren für das Beispiel aus Abbildung 4.1 vergleichen.

Lösungen, die auf einer solchen Hülle basieren, erzeugen zwar einen kompakten jedoch ineffizienten Code. Die Gründe hierfür sind die folgenden Einschränkungen:

- Leere Iterationen: eine perfekt verschachtelte Schleife zählt ein konvexes Polyeder auf; da aber die Vereinigung aller Domänen im allgemeinen nicht konvex ist, ist es wahrscheinlich, daß einige Iterationen der Schleife kein Statement ausführen. Betrachten wir z.B. die Schleife aus Abbildung 4.7, die von LooPo erzeugt wurde. Für

(a) nach dem Prinzip der bounding box (LooPo)

```
for (i=1; i<=N; ++i) {
  for (j=1; j<=N; ++j) {
    if (j<=M) {
       S1;
    }
    if (i==j && i>=3) {
       S2;
    }
  }
}
```

(b) nach dem Prinzip der konvexen Hülle (Omega)

```
for (i=1; i<=N; ++i) {
  for (j=1;
        j<=min(i+3N+M-4/4,i+M-1,N);
        ++j) {
    if (j<=M) {
      S1;
    if (i==j && i>=3) {
      S2;
    }
  }
}
```

Abbildung 4.7: Generierter Code für das Beispiel aus Abbildung 4.1

jeden Durchlauf der äußeren i-Schleife werden von den N Durchläufen der inneren j-Schleife nur etwa M Punkte auch wirklich ausgeführt. Wenn die Parameter entsprechende Verhältnisse zueinander aufweisen, z.B. wenn $M \ll N$ ist, kann dies zu einem signifikantem Overhead bei der Abarbeitung führen.

- Kontrolloverhead: jeder Schleifendurchlauf testet die Guards für alle Statements. Das erhöht den Aufwand von Kontrollberechnungen in jedem Durchlauf. Unter Umständen wird mehr Rechenaufwand für die Kontrollberechnungen als für die eigentliche Ausführung des Statements benötigt.

Das Ziel ist nun, diesen Overhead zu vermeiden. Der vorgestellte Lösungsansatz zerlegt die Vereinigung der Domänen in mehrere Regionen, welche dann mit nicht perfekt verschachtelten Schleifensätzen aufgezählt werden. Der dabei enstehende Code ist effizienter (allerdings dafür meist auch weniger kompakt als solcher mit perfekter Verschachtelung). Die Gründe hierfür sind:

- nicht perfekt verschachtelte Schleifensätze können auch nicht konvexe Regionen aufzählen, damit die leeren Iterationen vermieden werden;

- die Schleifen können so gewählt werden, daß einige Guards von Statements immer *true* oder immer *false* ergeben. Wenn ein Guard aber immer *true* liefert, so kann er entfernt werden; liefert er immer *false*, so kann man den Guard und die von ihm „überwachten" Instanzen des Statements aus dem Code eliminieren. Damit läßt sich die Anzahl der zu prüfenden Bedingungen senken und der Anzahl der Statementausführungen annähern. Dies reduziert den Kontrolloverhead. Im optimalen Falle hätten wir einen Schleifensatz ohne Guards.

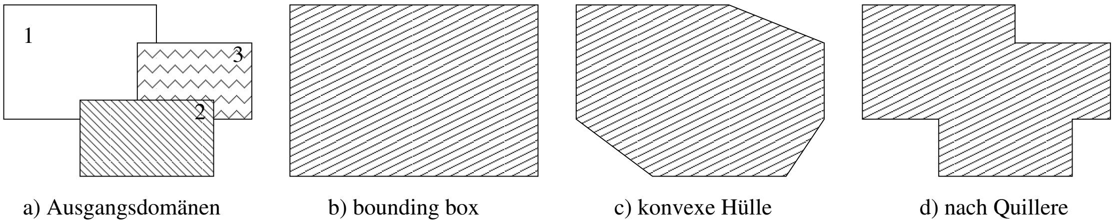


Abbildung 4.8: Vergleich der zu betrachtenden Punkte der verschiedenen Ansätze

Der Unterschied zwischen dem bisherigen und dem neuen Ansatz ist in Abbildung 4.8 illustriert. Abbildung 4.8 a) zeigt die Ausgangsdomänen. Die Abbildungen 4.8 b) bis 4.8 d) zeigen die aufgezählten Bereiche, wie sie nach den verschiedenen Ansätzen entstehen. Sehr deutlich ist hierbei der Vorteil der neuen Methode zu erkennen. Die Anzahl der betrachteten Punkte ist deutlich kleiner als z.B. im bisherigen Ansatz von LooPo. Eine Beschleunigung des entstehenden Codes ist deshalb zu erwarten.

Die Elimination dieses Overheads geht allerdings aus zwei Gründen zu Lasten der Codegröße:

1. Die perfekt verschachtelten Schleifen werden durch einen nicht perfekt verschachtelten Schleifensatz ersetzt; es werden nun disjunkte Polyeder aufgezählt; die Anzahl der Schleifen wächst dadurch rapide.

2. Die Domäne eines Statements wird in eine Menge von Polyedern zerlegt. Jedes dieser Polyeder wird von einem eigenen Schleifensatz aufgezählt. Der erzeugte Code wird dadurch für jede Schleife dupliziert. In Abbildung 4.9 wird dies am Beispiel aus Abbildung 4.1 verdeutlicht. Zum Vergleich der Codelänge der bisher angewandten Verfahren sei noch einmal auf Abbildung 4.7 verwiesen.

Trotz dieser Vergrößerung ist die Ausführung des Codes in der Praxis jedoch deutlich effizienter. Zurückzuführen ist dies auf die signifikante Verringerung des Overheads und eine deutliche Verbesserung des *Worstcase*-Verhaltens des Zielcodes. In Abbildung 4.10 ist ein Beispiel zu sehen, in dem die bisherigen Methoden einen sehr ineffizienten Code produzieren würden. Als Beispiel sei hier die Vorgehensweise von LooPo genannt. Teilbild a) zeigt die Grenzen der perfekten Verschachtelung, die generiert wird, wenn nach dem bisherigen Verfahren der *Bounding Box* vorgegangen wird ([Wet95]). Es werden zwei Schleifen erzeugt und die Punkte des Schleifenraumes, an denen Statements ausgeführt werden, mit einer `if`-Anweisung ausgewählt. Im Beispiel läuft die erste Schleife von x_1 bis x_2 und die zweite entsprechend von y_1 bis y_2. Schon in diesem sehr einfachen Beispiel müssen in jedem Schleifendurchlauf zwei Prüfberechnungen ausgeführt werden. Außerdem werden viel mehr Schleifendurchläufe gemacht, als eigentlich nötig wären. Der rechnerische Aufwand ist dadurch bedeutend größer geworden. Im Teilbild b) dagegen werden durch die neue Methode von Quilleré nur jeweils die Bereiche aufgezählt, in denen

```
for (i=1; i<=2; ++i) {
  for (j=1; j<=M; ++j) {
   S1;
  }
}
for (i=3; i<=N; ++i) {
  for (j=1; j<=min((-1+i),M); ++j) {
    S1;
  }
  if (i<=M) {
    j=i;
    S1;
    S2;
  }
  if (i>=M+1) {
    j=i;
    S2;
  }
  for (j=(1+i); j<=M; ++j) {
    S1;
  }
}
```

Abbildung 4.9: Aufzählen der Vereinigung der Domänen aus Abbildung 4.1

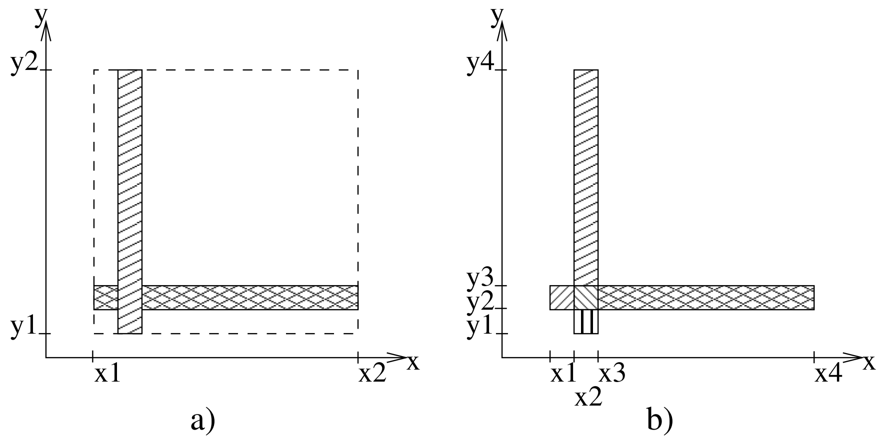


Abbildung 4.10: Vergleich der unterschiedlichen Ansätze

tatsächlich auch eine Anweisung ausgeführt wird. Wir erhalten also mehrere Schleifensätze, so z.B. einen, der die Grenzen x_1 und x_2 und y_2 und y_3 hat. Außerdem entfallen die Vergleichsoperationen der if-Anweisungen, die in a) die Ausführung kontrollieren. Der Berechnungsaufwand sinkt; allerdings wird dies mit einem deutlich vergrößerten Zielcode erkauft.

In der Praxis wurde dieses Verhalten von Sven Anders getestet, der in seiner Diplomarbeit [And00] bei Anwendung der hier vorgestellten Methode einen Geschwindigkeitsvorteil bei der Ausführung des Zielcodes feststellen konnte.

Wir stellen fest, daß das Verfahren von Quilleré durch seinen Aufbau eine starke Ähnlichkeit zu einer weiteren Lösungsmöglichkeit der Aufzählung mehrerer Polyeder aufweist: die sogenannte *Compiletime*-Lösung von LooPo [Wet95, Kap. 4.4.2.2]. Auf den ersten Blick erscheinen beide Verfahren in ihrer Idee gleich zu sein. Die Unterschiede sind jedoch sehr groß. Auch bei [Wet95] werden die Polyeder in kleine Teilpolyeder zerlegt und für jedes Polyeder ein Schleifensatz generiert. Sind die Grenzen der Schleifen jedoch so beschaffen, daß zur Übersetzungszeit mehrere Reihenfolgen bzw. Lagen der Polyeder zu einander denkbar wären, so wird kein eindeutiges Schleifenprogramm generiert, sondern eine Menge von Zielprogrammen (je eines für jede mögliche Permutation). Im ungünstigsten Fall entstehen so in der Anzahl der Parameter exponentiell viele Schleifensätze, von denen aber bei der Ausführung des Programmes nur ein Teil betrachtet wird. Auch hier entsteht eine nicht perfekt verschachtelte Schleifenstruktur, jedoch mit mehr Codezeilen. Die Guards, die in der *Runtime*-Lösung die Ausführung gesteuert haben, sind hier zwar aus den Schleifenrümpfen verschwunden, sie wurden jedoch durch entsprechende Guards ersetzt, die die Auswahl der richtigen Schleifenstruktur steuern. Durch diese exponentielle Entwicklung ergeben sich sowohl beim Speicherverbrauch als auch bei der Laufzeit des Zielprogrammes nur in günstigen Fällen Vorteile gegenüber der *Runtime*-Lösung. Auch die Laufzeit des Verfahrens selbst steigt durch den sehr hohen Speicherverbrauch bei der Generierung des Zielcodes stark an. Gegenüber dem Ansatz von Quilleré ist dies natürlich ein Nachteil, da er eine exponentielle Entwicklung im Code durch eine Sortierung vermeidet. Dabei wird in jedem Eliminationsschritt die aktuelle Ebene in Abhängigkeit des umliegenden Kontextes sortiert, so daß die Generierung exponentiell vieler möglicher Reihenfolgen der Abarbeitung vermieden wird. Eine genauere Erklärung des verwendeten Sortierungsalgorithmus findet sich in Kapitel 4.4.4.2.

4.4.3 Informelle Einführung in das Verfahren

Um die Methode zur Generierung solcher nicht perfekt verschachtelten Schleifensätze weiter beschreiben zu können, setzen wir voraus, daß alle Schleifen sequentiell sind und das folgende Problem ansprechen:

1. sie beschreiben eine Vereinigung disjunkter Polyeder,
2. sie zählen diese Vereinigung in einer lexikographischen Reihenfolge auf.

Versuchen wir nun, den Ansatz von Quillerés Lösung etwas genauer darzustellen. Es existiert also eine Vereinigung von Polyedern, die rekursiv in verschachtelte Schleifen zerlegt wird und zwar beginnend mit der äußersten hin zur innersten Schleife. In jedem Rekursionsschritt wird nun versucht, folgendes Problem zu lösen: für einen gegebenen Kontext[7] und eine Vereinigung von Polyedern in $\mathbb{Z}^n$ mit $n \geq d$ ist eine Schleife zu generieren, die die Vereinigung in lexikographischer Reihenfolge aufzählt.

Eine neue Ebene von Schleifen wird durch die folgenden Schritte generiert:

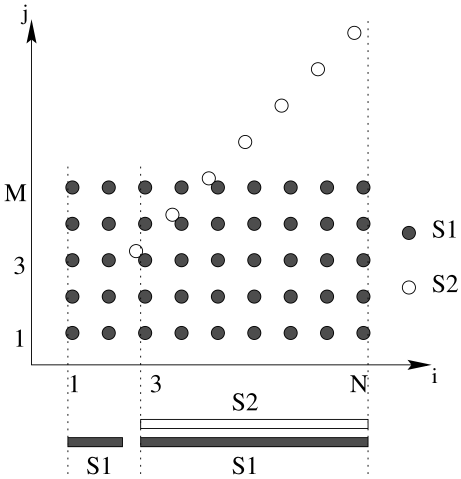
1. Die Polyeder werden in die äußerste Dimension projiziert.
2. Die Projektion wird gemäß der Überlappung der berechneten Projektionen in disjunkte Polyeder zerlegt.
3. Die Schleifensätze für jedes dieser neuen Polyeder werden rekursiv generiert.
4. Die Schleifen werden sortiert, so daß ihre textuelle Reihenfolge der lexikographischen Reihenfolge entspricht.

Die vier Schritte sollen an einem Beispiel näher erläutert werden. Wir betrachten das Programm, das schon in Abbildung 4.1 vorgestellt wurde. Den dazugehörigen Programmcode haben wir bereits in Abbildung 4.9 gesehen.

1. Als erstes wird jede Domäne in die erste Dimension projiziert. In unserem Beispiel ist sie durch die Variable i dargestellt. Wir erhalten eine Projektion, die von den Programmparametern `N` und `M` abhängig ist.
2. Das Ergebnis dieser Projektion wird nun in disjunkte Regionen zerlegt. Bei zwei Statements werden drei Regionen gebildet, nämlich jene, welche nur `S1` enthält, jene, welche `S1` und `S2` enthält und jene, welche nur `S2` enthält. Wie man aus Abbildung 4.11a leicht ersehen kann, ist in unserem Beispiel die letzte Region leer. Wir erhalten beim Separieren also nur zwei Polyeder, die durch zwei Schleifen in der i-ten Dimension beschrieben und von den Programmparametern `N` und `M` beschränkt werden. In Abbildung 4.11b sind die entstandenen Polyeder textuell dargestellt. Wie man sieht, ließe sich an diesem Punkt das erste Polyeder (Region) schon als Schleifensatz interpretieren, da es nur ein Statement (`S1`) enthält. Das zweite läßt sich noch nicht als Schleifesatz schreiben, da es zwei Statements mit unterschiedlichen Bedingungen beinhaltet. Diese Bedingungen sind zwar zum Teil redundant, die Elimination erfolgt allerdings erst später. Eine ausführliche Begründung dafür findet sich im nächsten Kapitel in den Bemerkungen zum Algorithmus.
   Zu bemerken sei hier noch, daß eine Region, die durch die Projektion gefunden wurde, natürlich auch eine Vereinigung von Polyedern darstellen kann. In unserem Beispiel werden wir gleich sehen, daß dies auf die zweite Region, die `S1` und `S2` enthält, zutrifft.
3. und 4. Jetzt wird zuerst rekursiv die Region 1 (die nur `S1` enthält) und danach Region 2 (mit `S1` und `S2`) aufgezählt und der Code von Region 1 vor dem der Region

[7] Der Kontext kann z.B. aus einem Polyeder in $\mathbb{Z}^{(d-1)}$ bestehen, das die äußeren Schleifen und die Systemparameter enthält.

(a) Projektion auf die Variable i und das Zerlegen in disjunkte Polyeder



(b) Die sortierten Schleifen der Variable i

```
{ i | 1<=i<=2 } ::   #Schleife L1#
    { i,j | 1<=i<=2; 1<=j<=M} ::
         S1;

{ i | 3<=i<=N } ::   #Schleife L2#
    { i,j | 3<=i<=N; 1<=j<=M } ::
        S1;
    { i,j | 3<=i<=N; j=i } ::
        S2;
```

Abbildung 4.11: Projektion und Separation am Beispiel

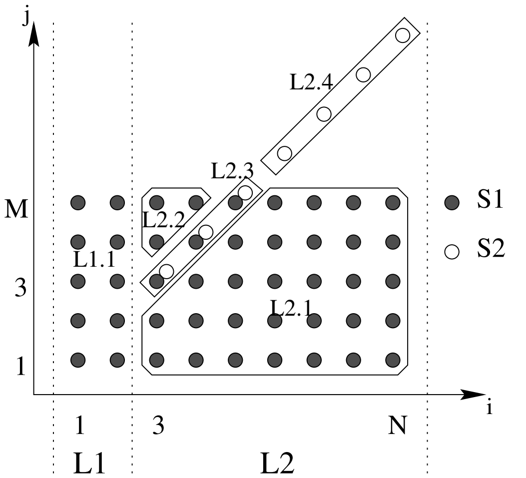
2 plaziert. Das verlangt die lexikographische Ordnung. Das Finden einer solchen Ordnung ist in der äußersten Dimension trivial, wird aber in den folgenden Ebenen komplizierter. Hier wollen wir im folgenden für unser Beispiel die Reihenfolge ohne eine genauere Erklärung angeben, da es an dieser Stelle den Rahmen sprengen würde. Die genaue Erklärung der Sortierung findet sich in Abschnitt 4.4.4.2.

An dieser Stelle soll nun noch auf die Berechnung der Schleifen in der nächsten Ebene eingegangen werden. Die Vorgehensweise soll weiter präzisiert werden, als dies bisher beim Betrachten der äußersten Dimension möglich war. Wir haben im vorherigen Schritt in der äußersten Dimension zwei neue Polytope erhalten (siehe Abbildung 4.11). Diese zwei Polytope stellen schon die Schleifen in der ersten Dimension dar. Ihre Grenzen werden als die jeweils äußerste Schleife des Schleifensatzes interpretiert. Die Rekursion in Schritt 3 läßt uns nun diese neu gewonnenen Polytope wieder in Hinblick auf die aufzuzählenden Schleifen betrachten. Die erste i-Schleife[8] enthält nur ein Statement (siehe dazu Abbildung 4.11), weshalb das Generieren des Zielschleifensatzes hier einfach ist. Die innere Dimension enthält nur eine Domäne, wodurch nur eine innere Schleife generiert werden muß. Es entsteht somit ein perfekt verschachtelter Schleifensatz. Eine weitere Zerlegung dieses Polyeders in der nächsten Dimension ist nicht nötig.

Schwieriger wird es, wenn wir die zweite i-Schleife[9] betrachten. Nach der vorherigen Berechnungsebene enthält `L2` zwei durch Domänen eingeschränkte Statements (siehe Abbildung 4.11b). Jetzt werden diese Domänen in vier disjunkte Polyeder zerlegt: `L2.1, L2.2, L2.3, L2.4` (siehe Abbildung 4.12). `L2.1` und `L2.2` enthalten nur das Statement

[8]Im Beispiel in Abbildung 4.11b) ist sie mit L1 markiert.
[9]Im Beispiel in Abbildung 4.11b) ist sie mit L2 markiert.

(a) Disjunkte Polyeder in den Dimensionen i und j



(b) Geordete Schleifen für die Polyeder

```
L1     { i | 1<=i<=2 } ::
L1.1     { i,j | 1<=j<=M} ::
             S1;

L2     { i | 3<=i<=N } ::
L2.1     { i,j | 1<=j<=i-1; j<=M } ::
             S1;
L2.3     { i,j | i=j; j<=M } ::
             S1;
             S2;
L2.4     { i,j | i=j; M+1<=j } ::
             S2;
L2.2     { i,j | i+1<=j<=M } ::
             S1;
```

Abbildung 4.12: Zweite Projektions- und Separationsebene

`S1`. In `L2.3` ist sowohl `S1` als auch `S2` enthalten. In `L2.4` findet sich nur `S2` wieder.

Auf diese Weise wird das Polyeder rekursiv zerteilt, so daß Teilpolyeder entstehen, die entweder das eine oder das andere oder beide Statements enthalten. Dadurch entstehen Schleifensätze, welche zum Ausführen eines Statements keinerlei Fallunterscheidung mehr benötigen. Die Polytope müssen so sortiert werden, daß die Schleifen, die sie beschreiben, textuell in der richtigen Reihenfolge angeordnet sind. Die Sortierung für unser Beispiel kann wie folgt angegeben werden[10]:

- um die lexikographische Ordnung zu erhalten, muß die Schleife `L2.1` vor `L2.2`, `L2.3` und `L2.4` ausgeführt werden;
- die Schleife `L2.3` muß vor `L2.2` stehen;
- es sei festgestellt, daß hier keine Beschreibung für die relativen Positionen der Schleifen `L2.2` und `L2.3` zu `L2.4` gemacht wurde, da die Kontexte (die i-ten Dimension: `{ i | i<=M-1 }` und `{ i | M+1<=i }`) voneinander unabhängig sind. Für jeden beliebigen Wert M gilt also: die Schleifen `L2.2/L2.3` werden nur ausgeführt, wenn $i \leq M$ bzw. $i < M$ ist; `L2.4` kommt dagegen nur zur Ausführung, wenn $i > M$ ist. Damit sind alle Reihenfolgen, die die beiden Schleifen zueinander einnehmen können, korrekt.

[10] Für eine genaue Diskusion und Beschreibung der Sortierung für dieses Beispiel verweisen wir auf Abschnitt 4.4.4.2; das Ergebnis ist hier schon vorweg genommen.

Daraus folgt, daß es unterschiedliche Möglichkeiten für die Ordnung der 4 Schleifen gibt. Eine davon ist in Abbildung 4.12(b) dargestellt worden.

Aus dieser Darstellung wird jetzt noch der Code generiert. Den erzeugten Code dieses Beispiels haben wir bereits6 in Abbildung 4.9 gesehen.

4.4.4 Formale Beschreibung des Algorithmus aus 4.4.3

Hier soll auf die formale Begründung des in Abschnitt 4.4.3 vorgestellten Algorithmus eingegangen werden. Im zweiten Teil wird darüberhinaus die schon mehrfach erwähnte Sortierung der Schleifen erläutert.

4.4.4.1 Der Algorithmus

Wie in 4.4.3 beschrieben, wird die Vereinigung der Polyeder von der äußersten zur innersten Schleifenvariable rekursiv zerlegt. Jeder Rekursionsschritt generiert dabei wieder eine neue Menge von Schleifen. Dabei ist ein Kontext gegeben, der beispielsweise ein Polyeder enthält, das die äußeren Schleifengrenzen und die Parameter beschreibt. Die vier Schritte, die für die Berechnung nötig sind, wurden in Abschnitt 4.4.3 auf Seite 60ff informell beschrieben. Der folgende Algorithmus generiert die Schleifen der Tiefe d aus einer gegebenen Vereinigung von den durch Domänen beschriebenen Statements $P_i :: S_i$ und dem Kontext C:

1. Ersetze jedes Polyeder P_i durch seinen Schnitt mit dem Kontext C; das gewährleistet, daß das neue Polyeder P_i genau die Domäne von S_i im Kontext der aktuellen umschließenden Schleifen definiert[11].

2. Berechne die Projektion L_i jedes Polyeders P_i auf die äußersten d Schleifenvariablen und die Parameter (Eliminieren der inneren Variablen). Jedes Statement S_i ist nun durch die Gültigkeit seiner Domäne P_i und seiner Projektion L_i auf diese Domäne bestimmt. Wir schreiben dafür $L_i :: P_i :: S_i$. Hier sei angemerkt, daß P_i nicht im Kontext von L_i vereinfacht wird, da diese Notation redundant wäre. P_i ist die Domäne von S_i und deren Beschränkungen werden in Abhängigkeit vom Kontext im letzten Schritt der Rekursion vereinfacht (eine nähere Erläuterung dazu findet sich in den nachfolgenden Bemerkungen zum Algorithmus). Alle übriggebliebenen Beschränkungen – z.B. solche, die keine Schleifengrenzen darstellen – bleiben als Bedingungen für S_i übrig.

3. Separiere alle Projektionen L_i in disjunkte Polyeder. Aus zwei gegebenen Schleifen $L_i :: P_i :: S_i$ und $L_j :: P_j :: S_j$, bei denen sich L_i und L_j überschneiden, werden drei Bereiche isoliert. Einer enthält nur S_i, der zweite enthält nur S_j und der dritte enthält den Bereich, in dem sich die beiden Ausgangsprojektionen überschneiden. Es entstehen die drei folgenden Schleifen:

[11] In Abschnitt 4.3.3 haben wir bereits gezeigt, daß der Kontext jeweils aus den Schleifeninformationen der umschließenden Schleifen und den Informationen über die Programmparameter besteht.

- $(L_i - L_j) :: P_i :: S_i$
- $(L_j - L_i) :: P_j :: S_j$
- $(L_i \cap L_j) :: (P_i :: S_i, P_j :: S_j)$

Die beiden ersten Regionen können durchaus aus mehreren Polyedern bestehen. In Abbildung 4.12 auf Seite 62 ist ein solches Beispiel zu sehen. Innerhalb der zweiten Schleife `L2` wird die Region, die nur das Statement `S1` enthält, aus der Vereinigung von `L2.1` und `L2.2` gebildet. In einem solchen Fall betrachtet der Algorithmus jedes an einer Vereinigung beteiligte Polyeder als eine Region und dupliziert die Statements entsprechend (für jedes Polyeder ensteht also eine eigene Schleife).
Die Separation wird so lange wiederholt, bis alle L disjunkt sind und damit alle Überschneidungen aufgelöst und in einzelne Polyeder zerlegt sind. Einer Schleife können auch mehrere Statements zugeordnet sein.
Der Übersichtlichkeit wegen erklären wir die Separierung von mehreren Schleifen in einer Bemerkung am Ende des Abschnitts (siehe Seite 65).

4. Rekursionschritt: Wende den Algorithmus für jede Schleife $L :: (P_r :: S_r, P_s :: S_s)$ mit einem neuen Kontext $C \cap L$ an.

5. Sortiere die Schleifen, um ihre Reihenfolge festzulegen. Wie der Algorithmus für die Sortierung funktioniert, wird ausführlich in Abschnitt 4.4.4.2 beschrieben.

6. Vereinfache alle Schleifen L_i mit dem Kontext C; dadurch werden redundante Schleifengrenzen eliminiert.

Bemerkung zur *Vereinfachung:*

Der Algorithmus vereinfacht die Schleifen erst in Schritt 6 ganz am Ende des Algorithmus und nicht schon in Schritt 2. Der Code kann dadurch optimiert werden, da nicht schon im ersten Durchlauf[12] des Algorithmus Informationen entfernt werden, die im nächsten Rekursionsschritt zu einer Verringerung des Aufwands genutzt werden können. Wir wollen dies am folgenden Beispiel zeigen:

Jeder der beiden folgenden Statements ist in eine zweidimensionale Domäne gefaßt:

```
{i,j,N | 1<=i<=10; 1<=j<=i}::S1
{i,j,N | 1<=i<=10; 11<=j<=N}::S2.
```

Der erste Durchlauf des Algorithmus erzeugt in Schritt 2 folgendes Programm:

```
{i,N | 1<=i<=10}::
```

[12] Da die Rekursion in Schritt 4 passiert, wird in einer Rekursionsebene der Schritt 6 erst ausgeführt, nachdem alle in der Rekursion tiefer liegenden Berechnungen ausgeführt wurden. Dadurch wird für eine Schleife die Vereinfachung erst nach Beendigung aller rekursiven Berechnungen ausgeführt.

```
{i,j,N | 1<=i<=10; 1<=j<=i}::S1
{i,j,N | 1<=i<=10; 11<=j<=N}::S2.
```

Wenn i an dieser Stelle in jeder j-ten Schleife vereinfacht wird, erhalten wir folgendes:

```
{i,N | 1<=i<=10}::
    {i,j,N | 1<=j<=i}::S1
    {i,j,N | 11<=j<=N}::S2.
```

Dadurch haben wir bei der Abarbeitung des Algorithmus im nächsten Rekursionsschritt (Generierung der j-ten Schleife) die Information $i \leq 10$ verloren. Deshalb muß an dieser Stelle die Annahme getroffen werden, daß $i \geq 11$ sein kann. Somit liegt eine Überschneidung der beiden Schleifen vor. Außer der Schleife für `S1` (mit $1 \leq j \leq 10$) und der Schleife für `S2` (mit $i \leq j \leq N$) wird noch eine zusätzliche Schleife für `S1` und `S2` generiert, welche für $11 \leq j \leq i$ gilt. Natürlich wissen wir aus dem ursprünglichen Kontext, daß die ersten beiden Schleifen disjunkt sind und die dritte Schleife immer leer ist. Dem Algorithmus liegt diese Information jedoch nicht mehr vor.

Es zeigt sich also, daß es sinnvoll ist, die Vereinfachung erst nach Generierung aller weiter innen liegenden Schleifen und somit erst im Schritt 6 vorzunehmen.

Bemerkung zur *Separierung* (Schritt 3):

Als Erweiterung der Erklärungen aus Schritt 3 geben wir hier den Algorithmus an, der für die Separierung der Polyeder verantwortlich ist. Er berechnet aus einer Menge vom Polyedern $\mathcal{P} = \{P_i\}$ eine Menge disjunkter Polyeder $\mathcal{Q} = \{Q_i\}$, so daß $\cup_i P_i = \cup_j Q_j$ ist. In jeder Iteration des Algorithmus wird ein Polyeder $P_i \in \mathcal{P}$ betrachtet und zur Berechnung einer neuen Menge $\mathcal{Q}'$ herangezogen. Die neue, anfangs leere Menge $\mathcal{Q}'$ wird mit folgendem Algorithmus errechnet:

1. Für alle $Q_j \in \mathcal{Q}$ wird $Q_j - P_i$ zum neu berechneten $\mathcal{Q}'$ hinzuaddiert. Damit werden die Bereiche, die nur den Statements von Q_j zugeordnet sind, der neuen Menge hinzugefügt.
2. Für alle $Q_j \in \mathcal{Q}$ wird $Q_j \cap P_i$ zum neu berechneten $\mathcal{Q}'$ hinzuaddiert. Damit werden die Bereiche, die den Statements von Q_j **und** P_i zugeordnet sind, der neuen Menge hinzugefügt.
3. Außerdem muß noch $P_i - (\cup_k Q_k)$ zu $\mathcal{Q}'$ hinzugenommen werden, um die Bereiche, die nur Statements von P_i enthalten, mit der neuen Vereinigung abzudecken.

4. Die Schritte 1–3 werden mit neuem $P_i \in \mathcal{P}$ und neuem $\mathcal{Q} = \mathcal{Q}'$ ausgeführt.

$\mathcal{Q}'$ enthält damit, nach Abarbeitung aller $P_i \in \mathcal{P}$, genau die Vereinigung disjunkter Polyeder, die $\mathcal{P}$ entspricht. Die Schritte 1-3 generieren dabei genau die Bereiche, wie sie auch im Separationsschritt des Algorithmus beschrieben sind, nur mit dem Unterschied, daß hier die Erweiterung auf eine beliebige Menge von Polyedern gemacht wurde.

Der Algorithmus zerteilt anfangs den Ausgangsraum nur in eine Menge von disjunkten Polyedern. Es werden keinerlei Berechnungen vorgenommen, um eine eventuelle Sortierung der Polyeder vorzunehmen. Der Algorithmus nimmt eine Menge ungeordneter Polyeder und liefert als Ergebnis auch eine Menge ungeordneter, jetzt allerdings disjunkter Polyeder.

Im nächsten Abschnitt wollen wir zeigen, daß wir aus einer Menge ungeordneter, disjunkter Polyeder eine geordnete Menge von Schleifen generieren können, die einer lexikographischen Ordnung genügt. Wir werden sehen, daß die Disjunktheit der Polyeder für die Berechnung einer solchen Ordnung notwendig ist.

4.4.4.2 Das Sortieren

Hier soll nun näher auf das schon mehrmals angesprochene Problem des Sortierens der voneinander unabhängigen neu generierten Domänen (Schleifen) eingegangen werden. Dabei wollen wir im ersten Teil zunächst die Idee erläutern, um dann im zweiten Teil einen allgemeinen Algorithmus vorzustellen.

Die Idee der Sortierung

Die bisher besprochenen Schritte für die Generierung der Schleifen liefert uns zwar separierte Schleifenstücke, allerdings erhalten wir keinerlei Information über die Reihenfolge, in der sie auszuführen sind. Ohne diese Ausführungsreihenfolge können jedoch keine Schleifen für das Zielprogramm generiert werden, da nicht sichergestellt ist, daß die lexikographische Ordnung der einzelnen Abarbeitungsschritte des Urspungsschleifensatzes erhalten bzw. berücksichtigt wird. Dieser Abschnitt soll nun zeigen, daß die Reihenfolge implizit in den errechneten Schleifen vorhanden ist und daraus eine entsprechende Ordnung berechnet werden kann. Als erstes wird eine Ordnung zwischen den Polyedern hergestellt. Daraus kann dann die textuelle Ordnung der Schleifen abgeleitet werden.

Um eine solche Sortierung oder Ordnung zweier Schleifen (oder Polyeder) feststellen zu können, muß als erstes gezeigt werden, daß es immer möglich ist, den Programmcode, der die eine Schleife aufzählt, textuell vor den Code der zweiten Schleife zu setzen, ohne dabei die lexikographische Ordnung[13] zu verletzen. Natürlich ist eine solche Ordnung nur dann sinnvoll, wenn die zu ordnenden Polyeder einen gemeinsamen Kontext haben.

[13] Gemeint ist hierbei die lexikographische Ordnung der Daten der Ausführungspunkte, die durch den Schedule gegeben ist.

Andernfalls gibt es keine Grundlage für eine wie auch immer geartete Ausführungsreihenfolge; sie wäre egal. Die Ordnung der Polyeder ist eine partielle Ordnung. Grundlegend ist dabei, daß alle betrachteten Polyeder disjunkt sind, da es nicht möglich ist, für zwei sich überschneidende Polyeder eine eindeutige Ordnung der Ausführungspunkte zu finden.

Eine Schleife, die von l-Schleifen umgeben ist und mit p-Parametern bestimmt wird, wird mit einem 1-dimensionalen Polyeder beschrieben, das durch einen $(l + p)$-Vektor parametrisiert ist. Dieser $(l + p)$-Vektor ist der Kontext der Schleife. Eine Ordnung der Schleifen (Reihenfolge der Abarbeitung) muß natürlich für alle möglichen Werte des Kontextes (also jeden Wert der äußeren Schleifenvariablen und der Programmvariablen) gelten, damit sie eine gültige Reihenfolge darstellt.

Als erstes wird eine partielle Ordnung auf den parametrisierten Polyedern definiert. Seien D_1 und D_2 zwei 1-dimensionale Polyeder mit den $(l + p)$-Vektoren als Parameter und die Kontexte C_1 und C_2 die jeweiligen $(l + p)$-dimensionalen Domänen aus den Parametervektoren der beiden Polyeder. Anders ausgedrückt bedeutet dies, daß C_1 und C_2 die jeweilige Projektionen von D_1 und D_2 auf die Parameter sind.

Eine mit $\lhd$ bezeichnete Ordnung sei wie folgt definiert:

$D_1 \lhd D_2 \Leftrightarrow$

1. $D_1 \cap D_2 = \emptyset$
2. $C_1 \cap C_2 \neq \emptyset$
3. $\forall q \in C_1 \cap C_2,\ \forall x \in D_1(q),\ y \in D_2(q),\ x < y$,
   d.h. für einen beliebigen Wert im Kontext $C_1 \cap C_2$, also in dem Bereich, in dem sich die umgebenden Schleifen von D_1 und D_2 überlappen, gilt: jeder Punkt aus D_1 muß in der Richtung der betrachteten Dimension vor einem beliebigen Punkt aus D_2 stehen. Betrachtet man die Kombination aus Kontext und der gerade betrachteten Dimension als den „Zeitpunkt" der Ausführung von x bzw. y, so kann man sagen, daß x in der lexikographischen Ordnung $\prec_t$ der „Zeitpunkte" vor y steht.

Mit diesen Voraussetzungen ist $\lhd$ eine strikte Ordnung.

Die Bedingung 1, daß wir den Algorithmus nur auf disjunkte Polyeder anwenden, ist schon implizit durch den Separationsalgorithmus des vorherigen Absatzes sichergestellt.

Falls Punkt 2 nicht zutrifft, ist die Lösung bereits gefunden. In diesem Falle sind die beiden Polyeder nicht vergleichbar, also im Sinne einer Ordnung voneinander unabhängig. Die korrespondierenden Schleifen sind somit in jeder beliebige Reihenfolge ausführbar.

Falls allerdings beide Bedingungen erfüllt sind, sind die beiden Polyeder miteinander vergleichbar und eine der beiden folgenden Relationen muß zutreffen:

- $D_1 \lhd D_2$ oder
- $D_2 \lhd D_1$.

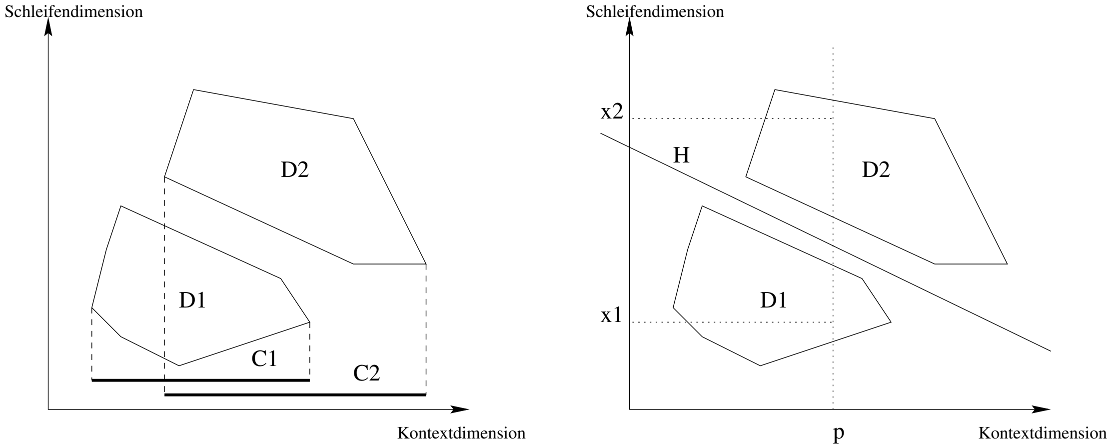


Abbildung 4.13: Relation zweier Schleifen zueinander ($D_1 \lhd D_2$)

Das bedeutet, daß eine Ordnung der beiden Polyeder existieren muß. Um dies zu verdeutlichen, betrachten wir die beiden Polyeder D_1 und D_2 in Abbildung 4.13. Sie sind disjunkt und es existiert deshalb eine Hyperebene H, die den kombinierten Kontext-Schleifenraum in zwei Teilräume R_1 und R_2 teilt. Die beiden entstandenen Räume enthalten einmal D_1 und einmal D_2. H ist nicht parallel zur Schleifendimension (siehe Abbildung 4.13), da H nicht rechtwinklig zum Kontext verlaufen kann. Sonst wären die beiden Kontexte C_1 und C_2 disjunkt und somit die Bedingung 2 verletzt.

Allgemein kann man folgendes sagen: H ist genau so gelegen, daß eine der beiden folgenden Eigenschaften erfüllt ist:

$$\forall q \in \mathbb{Z}^{(l+p)}, \forall x, y \in \mathbb{Z} : \begin{pmatrix} q \\ x \end{pmatrix} \in R_1 \wedge \begin{pmatrix} q \\ y \end{pmatrix} \in R_2 \Rightarrow x < y \tag{4.2}$$

oder

$$\forall q \in \mathbb{Z}^{(l+p)}, \forall x, y \in \mathbb{Z} : \begin{pmatrix} q \\ x \end{pmatrix} \in R_1 \wedge \begin{pmatrix} q \\ y \end{pmatrix} \in R_2 \Rightarrow y < x \tag{4.3}$$

Falls die beiden Bedingungen $D_1 \cap D_2 = \emptyset$ und $C_1 \cap C_2 \neq \emptyset$ erfüllt sind, gilt:

wenn (4.2) erfüllt ist, dann gilt $D_1 \lhd D_2$ oder
wenn (4.3) erfüllt ist, dann gilt $D_2 \lhd D_1$.

Da die beiden Polyeder sicher disjunkt sind, ist eine der beiden Bedingungen auf jeden Fall erfüllt.

Das Beispiel aus Abbildung 4.13 erfüllt die Bedingung 4.2 und es gilt somit:

$$D_1 \lhd D_2.$$

Der Algorithmus

Wir wollen nun einen Algorithmus vorstellen, der für zwei beliebige disjunkte parametrisierte Polyeder D_1 und D_2 deren Ordnungsrelation zueinander berechnet.

Das Resultat eines solchen Algorithmus ist entweder die „Unvergleichbarkeit"[14] von D_1 und D_2 oder eine Ordnung der Form $D_1 \lhd D_2$ bzw. $D_2 \lhd D_1$.

Im folgenden sei der Algorithmus schrittweise dargestellt:

1. Berechne die Projektionen von D_1 und D_2 auf den Kontextraum C_1 und C_2.

2. Bilde den Schnitt der Projektionen: $\mathcal{I} = C_1 \cap C_2$.

3. Falls der Schnitt $\mathcal{I} = \emptyset$ gilt, dann sind die zwei Polyeder nicht miteinander vergleichbar, sondern voneinander unabhängig $\rightarrow$ `exit`.

4. Andernfalls sind die Polytope miteinander vergleichbar. Es gilt also entweder $D_1 \lhd D_2$ oder $D_2 \lhd D_1$.

5. Um die richtige Relation der beiden Polyeder zu berechnen, nehmen wir an, daß $D_1 \lhd D_2$ gilt. Daraus folgt:

$$\forall p \in \mathcal{I}, \forall x_1 \in D_1(p), \forall x_2 \in D_2(p), \exists x : x_1 < x < x_2.$$

   Dies bedeutet nichts anderes, als daß wir eine Hyperebene H zwischen die beiden Polyeder legen können. Um diese Annahme nun zu bestätigen, berechnen wir die Menge

$$P = \{x \mid x_1 < x < x_2, x_1 \in D_1(p), x_2 \in D_2(p)\}.$$

   Diese Menge ist ein Polyeder, das wie folgt konstruiert wird:

   a) Wähle aus D_1 alle Beschränkungen "$x < \ldots$".
   b) Wähle aus D_2 alle Beschränkungen "$x > \ldots$".
   c) Negiere alle diese Beschränkungen und berechnen dann das daraus resultierende Polyeder P.

   Ist P nun eine leere Menge, so war unsere Annahme $D_1 \lhd D_2$ falsch und es muß gelten: $D_2 \lhd D_1$. Ist P nicht leer, so haben wir unsere Annahme bewiesen und es gilt $D_1 \lhd D_2$. Zur Veranschaulichung ist in Abbildung 4.14 der Fall dargestellt, daß die Annahme richtig war.

Der eben vorgestellte Algorithmus ist in der schon mehrfach erwähnten Polylib implementiert.

Wir bekommen also eine partielle Ordnung der Polyeder. Allerdings brauchen wir **eine** textuelle Ordnung (von mehreren möglichen) auf allen Polyedern.

[14] D_1 und D_2 sind unvergleichbar, wenn sich die Kontexte der beiden Polyeder nicht schneiden. Dies ist der Fall, wenn H orthogonal zur Achse der Kontextdimension liegt.

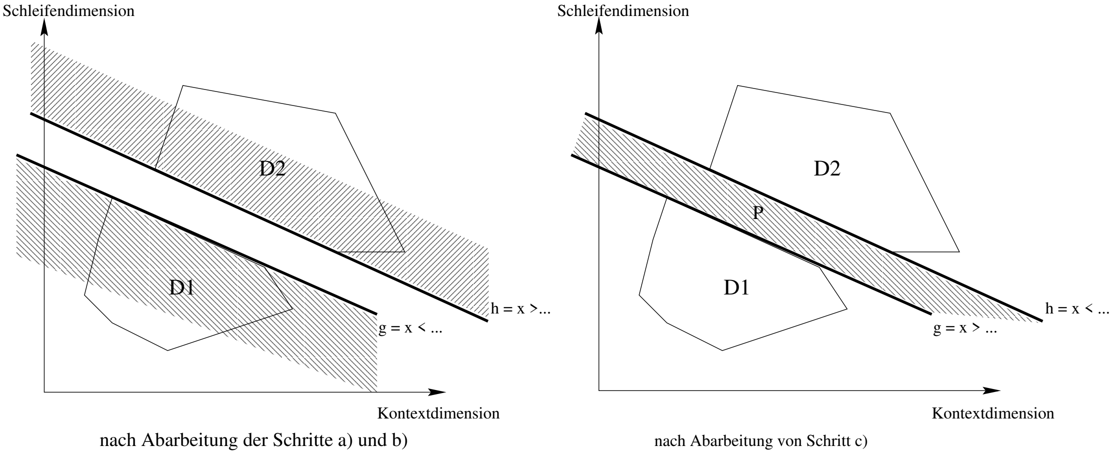


Abbildung 4.14: Konstruktion des Polyeders in Punkt 5 des Algorithmus

Das Verfahren, das eine solche Ordnung findet, besteht aus zwei Teilen. Im ersten Schritt wird ein Vorgänger/Nachfolger-Graph (Abhängigkeitsgraph) der Menge aller Polyeder konstruiert. Im zweiten Schritt wird durch einen Durchlauf durch den Graphen aus der Menge der möglichen Ordnungen **eine** ausgewählt.

Zur Konstruktion des Graphen werden die Schleifen als Knoten im Graphen dargestellt. Die Kanten werden durch das Finden der vergleichbaren Polyeder generiert. Immer wenn zwei Polyeder miteinander vergleichbar sind (d.h. es überschneiden sich ihre Kontexte), wird zwischen beiden eine Kante in den Graphen eingefügt.

Für x Schleifen ergeben sich also $\frac{x(x-1)}{2}$ zu berechnende Vergleiche und demnach eine Komplexität von $O\left(x^2\right)$. Dies ist allerdings die obere Schranke, die sich durch Ausnutzung der Transitivität verkleinern läßt; die bereits gefundenen Kanten müssen nicht noch einmal berechnet werden. Diese obere Schranke wird außerdem nur dann erreicht, wenn keines der Polyeder mit einem anderen vergleichbar ist und somit die Anzahl der Kanten im Graphen 0 beträgt. Es gibt dann keine Abhängigkeiten zwischen den Polytopen und jede beliebige Ordnung auf den Schleifen wäre möglich. Theoretisch wäre dies der günstigste Fall, da man keine Zeit auf die Berechnung einer Ordnung verschwenden müßte. In der Praxis muß leider gerade in diesem Fall der größte rechnerische Aufwand betrieben werden, da für jedes Paar von Polytopen erst festgestellt werden muß, ob sie miteinander vergleichbar sind oder nicht. Typische Algorithmen für Sortierungen bzw. Vergleiche (da bei einer Sortierung natürlich alle Elemente miteinander verglichen werden müssen) erreichen eine Komplexität $O\left(x \log x\right)$, wie man zum Beispiel bei [Her92] nachlesen kann.

Im zweiten Schritt wird aus dem Graphen, der die Ordnungsrelationen zwischen den Polyedern abbildet, eine gültige Ordnung für die Polyeder extrahiert. Der Algorithmus durchläuft dabei den Graphen wie folgt:

1. Wir beginnen mit $n = 0$.

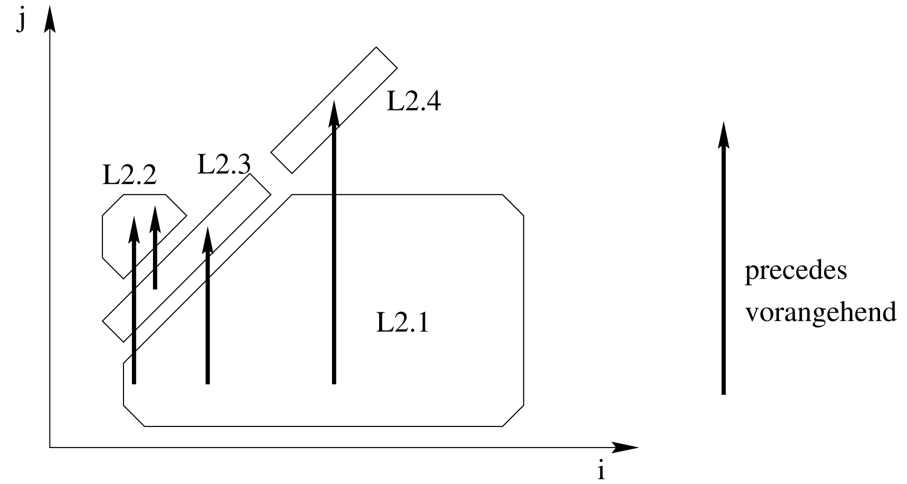


Abbildung 4.15: Partielle Ordnung zwischen parametrisierten Schleifen aus Abbildung 4.12

2. Nun suchen wir im Graphen einen Knoten (Polyeder), der keinen Vorgänger hat, und markieren ihn mit n. Danach wird dieser Knoten aus dem Graphen gelöscht.
3. Jetzt wird n um 1 erhöht und (solange der Graph noch nicht leer ist) Schritt 2 wiederholt.
4. Jedes Polyeder hat nun eine eindeutige Nummer zwischen 0 und der Anzahl der Polyeder weniger 1. Die generierten Schleifen können dann anhand ihrer Positionsnummern aufsteigend angeordnet werden.

Nach Abarbeitung des Algorithmus sind alle Knoten eindeutig mit einer Nummer versehen und es wurde eine der möglichen Reihenfolgen für die weitere Bearbeitung ausgewählt.

Beispiel für eine Sortierung

Wir wollen den Algorithmus am Beispiel aus Kapitel 4.4.3 etwas genauer illustrieren. Wir erinnern uns an die Abbildung 4.12 a) auf Seite 62. Die Sortierung der dort gezeigten, im zweiten Schritt der Rekursion generierten Polyeder wurde auf Seite 62 schon vorweggenommen. Abbildung 4.15 zeigt uns die Schleife `L2` mit den in ihr enthaltenen vier disjunkten j-Schleifen, nämlich `L2.1`, `L2.2`, `L2.3` und `L2.4` und deren Relation zueinander. Wir wollen nun an diesem Beispiel noch einmal die Berechnung der Auswahl (und damit auch die Bestimmung der Ordnung) demonstrieren:

- für jedes beliebige i in den Parametern liegt `L2.1` vor allen anderen Schleifen. Im korrespondierenden Graphen hat `L2.1` keinen direkten Vorgänger. `L2.1` ist die Wurzel des Graphen und wird deshalb mit 0 markiert.

- `L2.1` wird ignoriert und aus dem korrespondierenden Graphen entfernt. Wir finden nun `L2.3` und `L2.4` ohne direkten Vorgänger und beide könnten unmittelbar hinter `L2.1` eingeordnet werden. Damit ergeben sich zwei mögliche Reihenfolgen:

  - `L2.1`; `L2.3` oder
  - `L2.1`; `L2.4`.

  Wir wählen eine der beiden, in unserem Fall die erste, aus und markieren `L2.3` mit 1.

- Wir können den gefundenen Knoten wiederum aus dem Graphen entfernen und den Rest betrachten. Die zwei verbliebenen Knoten `L2.4` und `L2.2` sind nicht vergleichbar (voneinander unabhängig) und haben deshalb keine Kante im Graphen. Die Reihenfolge ist somit irrelevant. Es ergeben sich also wieder zwei mögliche „richtige" Reihenfolgen:

  - `L2.1`; `L2.3`; `L2.2`; `L2.4` oder
  - `L2.1`; `L2.3`; `L2.4`; `L2.2`.

- Wenn wir im zweiten Schritt statt `L2.3` nun `L2.4` gewählt hätten, so würde, wegen der Abhängigkeit in der `L2.3` und `L2.2` zueinander stehen, die korrekte Ordnung die folgende sein:

  - `L2.1`; `L2.4`; `L2.3`; `L2.2`.

Da die Sortierung in jedem Rekursionsschritt der Schleifengenerierung auf den jeweiligen Polyedern berechnet wird, ist die Einordnung in den richtigen Kontext immer gegeben. Die inneren Schleifen werden vor den umschließenden Schleifen sortiert. Einen Konflikt mit einer anderen, in einer weiter außen liegenden Dimension befindlichen Schleife ist damit ausgeschlossen. Die richtige Verschachtelung der Schleifen ist gewährleistet.

4.4.5 Komplexitätsbetrachtungen

Wie wir gesehen haben, stellt die in den vorangegangenen Kapiteln vorgestellte Methode zur Aufzählung einer sich überlappenden Menge von Polyedern einen erhebliche Verbesserung der bisherigen Lösungsansätze dar. Im Gegensatz zu den bisher verwendeten Verfahren sinkt der Berechnungsaufwand, der bisher einen zusätzlichen Overhead im Zielcode darstellte, deutlich. Gerade bei größeren Projekten ist dies ein enormer Geschwindigkeitsvorteil. Erreicht wird dies durch die Verwendung nicht perfekt verschachtelter Schleifensätze statt der bisher eingesetzten perfekten Verschachtelung. Nachteil dieses Ansatzes ist die deutlich höhere Anzahl an Zeilen im Zielcode. Dies vergrößert den Speicherbedarf und macht das Ergebnis sehr unübersichtlich. Schon der relativ einfache Beispielschleifensatz, der hier zur Illustration verwendet wurde, produziert bei Verwendung der Methode doppelt so viele Codezeilen wie die Anwendung eines der bisherigen Verfahren (vergleiche Abbildung 4.7a mit 4.9).

Wenden wir uns nun der Komplexität des Verfahrens zu. Entscheidend für die Laufzeit sind die Schritte des Algorithmus, wie in 4.4.4.1 aufgeführt.

In Schritt 1 wird der Schnitt eines Polyeders mit dem Kontext berechnet. Die Berechnung ist von der Anzahl der Ungleichungen der beiden Schnittkandidaten abhängig. Innerhalb der Berechnung werden erst die Ungleichungen beider Systeme miteinander kombiniert und dann einmal der Chernikova-Algorithmus ausgeführt. Die Laufzeit ist damit exponentiell in der Anzahl der Summe der Ungleichungen.

Schritt 2 besteht aus der Projektion des Polyeders wie sie in 4.2.1 beschrieben ist. Auch hier ist ein Aufruf des Chernikova nötig und somit ist auch hier die Laufzeit in der Anzahl der Ungleichungen des Polyeders exponentiell.

Die Separation in Schritt 3 wurde genauer in den Bemerkungen zum Algorithmus beschrieben. Sie ist abhängig von der Anzahl der sich überlappenden Polyeder, die berechnet werden. In jedem Durchlauf werden für jeden schon berechneten separierten Polyeder maximal 3 neue Polyeder hinzugefügt. Wir sind damit exponentiell in der Anzahl der Ausgangspolyeder.

Schritt 4 ist der Rekursionsschritt. Die Schritte 1-3 werden für alle neu entstandenen Polyeder ausgeführt. Die Tiefe der Rekursion wird dabei von der Anzahl der zu berechnenden Unbekannten bestimmt. Die Rekursionsbreite ist von der Menge der Überschneidungen abhängig, da dies die Anzahl der neu entstehenden Polyeder bestimmt.

Die Sortierung in Schritt 5 wird für jede Rekursionsebene einmal ausgeführt. Der Algorithmus (siehe 4.4.4.2) erstellt erst einen Ordnungsgraphen der Polyeder einer Rekursion und wählt danach aus den möglichen Reihenfolgen, die dieser Ordnung entsprechen, eine aus. Die Berechnung der Ordnung zwischen zwei Polyedern bedient sich dabei wieder des Algorithmus von Chernikova und ist dadurch exponentiell in der Summe der Anzahl der Ungleichungen der beiden Polyeder. Für n Polyeder muß diese Berechnung maximal $\frac{n(n-1)}{2}$ mal ausgeführt werden, womit die Berechnung einer Ordnung aller Polyeder im schlimmsten Fall exponentiell in der Anzahl der Polyeder ist. Das Finden der Reihenfolge im Graphen (nach dem Algorithmus aus 4.4.4.2) ist polynomial in der Anzahl der Knoten im Graphen.

Der sechste und letzte Schritt des Algorithmus ist die Vereinfachung aller Polyeder wie in Abschnitt 4.2.2 beschrieben. Auch hier wird wieder der Algorithmus von Chernikova benötigt, so daß die Komplexität ebenfalls exponentiell in der Anzahl der Ungleichungen ist.

Der gesamte Algorithmus ist somit (wegen des Rekursionsschritts 4) exponentiell in der Anzahl der zu berechnenden Variablen; die einzelnen Instanzen der Berechnung sind allerdings in der Anzahl der betrachteten Ungleichungen exponentiell.

4.4.6 Zusammenfassung

Das Berechnen einer Aufzählung für mehrere sich überlappende Polyeder ist für das Parallelisieren komplexerer Programme essentiell, da sonst komplexere Schleifenstrukturen in einem sequentiellen Programm nicht parallelisiert werden könnten. Die bisher verwendeten Methoden lieferten allerdings nur wenig effiziente und somit oft auch unbefriedigende Lösungen. Die bisher in LooPo verwendeten Verfahren ([Wet95], [KPR95]) liefern einen Programmcode, der unter ungünstigen Umständen die Ausführung eines Programms auf einer parallelen Architektur wegen der vielen zusätzlichen Berechnungen von Bedingungen bei der Ausführung ausbremst.

An dieser Stelle setzt das Verfahren von F. Quilleré an. Es vermeidet die Generierung solcher zusätzlichen Bedingungen, die bei den bisherigen Methoden die Ausführung der Statements steuerten. Durch den Einsatz einer imperfekten Verschachtelung im Zielcode ist es möglich, ein effizienter und schneller ausführbares Zielprogramm zu erzeugen. Die Methode ist somit für den Einsatz in LooPo sehr gut geeignet und kann als Ergänzung zu bisherigen Verfahren dienen. Allerdings muß auch festgehalten werden, daß das Verfahren für den Einsatz im großen Rahmen bisher nur bedingt empfehlenswert ist, da die Beschleunigung in der Ausführung mit deutlichen Nachteilen bei der Größe des Zielcodes erkauft wird. Hier sind noch Verbesserungen notwendig (siehe Abschnitt 4.4.7).

4.4.7 Erweiterungen und Probleme der Methode

Wir haben gesehen, daß die vorgestellte Methode gegenüber den bisher verwendeten Verfahren einen Effizienzvorteil liefern kann. Es verbleibt allerdings auch hier noch Raum für Verbesserungen.

Ein Beispiel dafür ist, daß auch mit den bisher vorgestellten Funktionalitäten die Entstehung von Codefragmenten, die unter Umständen für eine Kontextsituation eine leere Schleife beschreiben, nicht vermieden werden kann. Abhängig vom Kontext (z.B. den Programmparametern) werden so bei der Abarbeitung Schleifen berechnet, die keine Anweisungen ausführen. Dies führt zu einem unnötigen Berechnungsmehraufwand. Die Abhängigkeiten vom Kontext sind allerdings schon zur Übersetzungszeit bekannt und können deshalb auch schon zum Zeitpunkt der Übersetzung optimiert werden. Dafür ist eine zusätzliche Separation im Algorithmus (zwischen Schritt 5 und 6) notwendig. Die Schleifen werden dabei auf ihren Kontextraum projiziert und dann entsprechend separiert und sortiert. Dadurch werden die äußeren Schleifen in Abhängigkeit vom Kontext noch einmal zerteilt. Auf diesem Wege erhalten wir eine Verfeinerung der Schleifensätze, die allerdings auch zu einer weiteren Vergrößerung des Codes durch die Entstehung einer stärkeren Verschachtelung führt.

Damit haben wir bereits das nächste Problem angesprochen: Wie kann man ein gutes Verhältnis zwischen der Effizienz des Codes und seiner Größe herstellen? Eine allgemeingültige Antwort auf diese Frage läßt sich nicht finden, da die Effizienzsteigerung eines Programmes durch die Elimination der Guards (wie sie von Quillerés Ansatz erreicht wird) von der Anzahl der Berechnung der einzelnen Guards während der Abarbeitung des Programms abhängt. Verläßliche Algorithmen gibt es für eine Lösung dieses Problems nicht. Man ist hier auf die Anwendung von Heuristiken angewiesen. Eine mögliche Größe für die Abschätzung ist die Tiefe des Guards in der Verschachtelung der Schleife. Ein weiter innen liegender Guard wird im allgemeinen öfter als ein außenliegender berechnet und ist somit „teurer". Eine solche Heuristik ist im Algorithmus einfach umzusetzen, da durch die Rekursion in jedem Schritt entschieden werden kann, ob das gerade zu betrachtende Polyeder zerlegt werden soll oder nicht. Eine weitere Möglichkeit wäre das Abschätzen des Volumens des Polyeders. Großvolumige Polyeder haben viele Berechnungspunkte und somit meist auch viele zu testende Guards. Sie sollten also besser aufgespalten werden. Kleinvolumige Polyeder dagegen können mit geringeren Verlusten ohne eine Zerlegung berechnet werden. Problematisch ist wiederum die Grenzziehung zwischen „klein" und „groß" als Entscheidungskriterium, da auch hier keine verläßlichen Regeln gelten.

5 Zusammenfassung

In den vorangegangenen Kapiteln wurden verschiedene Möglichkeiten vorgestellt, wie man bei Anwendung einiger neuer Methoden Steigerungen in der Effizienz der automatischen Parallelisierung erreichen kann. Solche Verbesserungen sind notwendig, wenn parallele Verfahren in der Praxis angewendet werden sollen. Der theoretische Vorteil, den eine parallele Abarbeitung verspricht wird heute oftmals durch Probleme bei der Umsetzung blockiert. Nur wenn es gelingt bestehende Strukturen und Methoden effektiv auf die parallele Welt abzubilden, können die möglichen Steigerungen in Laufzeit und Aufwand auch genutzt werden.

Die Art der Verbesserung, die mit den vorgestellen Algorithmen zu erreichen ist, ist allerdings sehr unterschiedlich. Das erste Verfahren, die Eliminationsmethode von Weispfenning, bringt vor allem Einsparungen in der Laufzeit des Parallelisierungsalgorithmus. Es beschleunigt das Lösen der Ungleichungssysteme, die mit den bisherigen Verfahren einen hohen Rechenaufwand erzeugt haben. Durch seine *Divide & Conquer* Architektur kann viel Zeit bei der Berechnung der Schranken von Ungleichungssystemen gewonnen werden. Gegenüber bisherigen Verfahren sinkt der Berechnungsaufwand von doppelter auf einfache Exponentialität.

Auch der erste Teil des zweiten Verfahrens kann für die Parallelisierung eines Algorithmus einen Vorteil bringen. Hier wird der Geschwindigkeitsvorteil aber auf einem anderen Wege erreicht. Es wird an der eigentlichen Idee des Verfahrens zum Lösen eines Ungleichungssystems, wie es dem Algorithmus von Fourier und Motzkin zugrunde liegt, nicht viel verändert. Die Einsparungen werden auf dem Wege einer effizienteren Berechnung erzielt. Durch die Verwendung der dualen Darstellung kann bei der Elimination der einzelnen Variablen durch einen Adjazenztest die doppelt exponentielle Vergrößerung der zu betrachtenden Ungleichungen verhindert werden und es wird somit gegenüber FM eine Verbesserung der Geschwindigkeit erreicht.

Der Vorteil der aus diesen zwei Verfahren erwächst liegt im Bereich der Generierung einer parallel ausführbaren Programmstruktur, also bei den Verfahren, welche die sequentielle Realität in die parallele Welt abbilden.

Aus dem zweiten Verfahren ist eine Erweiterung hervorgegangen, die an einer anderen Stelle von LooPo Verbesserungen hervorbringt. Das Verfahren aus Abschnitt 4.4 zählt mehrere Polyeder auf und erreicht bei der Erzeugung von Codesequenzen im Vergleich zu bisherigen Verfahren ([Wet95], [KPR95]) Verbesserungen. So wird durch eine Zerlegung der betrachteten Polyeder und eine imperfekte Schachtelung des Zielcodes die Menge der Codezeilen zwar deutlich vergrößert, im Gegenzug jedoch die Anzahl der in jedem Durchlauf zu berechnenden Vergleiche (Guards) deutlich vermindert. Desweiteren wird im Gegensatz zu den bisherigen Verfahren, die mit *Bounding Box* oder *konvexer Hülle* arbeiten, die Anzahl der Instanzen des Zielcodeschleifensatzes auf die Menge der tatsächlich im Quellcode vorkommenden reduziert. Das Ergebnis ist die Verbesserung des Codes in zweierlei Hinsicht: erstens werden weniger Schleifeninstanzen im Zielcode aufgezählt und somit weniger Berechnungen ausgeführt und zweitens werden die nötigen Vergleiche für jeden Durchlauf reduziert. Das verringert wiederum die Anzahl der nötigen Berechnungen. Der Nachteil des Verfahrens ist die signifikant steigende Anzahl an Codezeilen und die auch mit diesem Verfahren unvollständige Elimination von unerreichbarem Code im Zielprogramm. Hier bleibt noch viel Raum für Verbesserungen.

Der Vorteil dieser Erweiterung liegt also nicht unbedingt im Bereich der Generierung eines parallelen Algorithmus, sondern in der Effizienz des nach der Parallelisierung entstandenen Codes. Dies ist ein Schritt in Richtung einer effizienten Umsetzung sequentiellen Codes in sein paralleles Pendant.

Literaturverzeichnis

[AI91] C. Ancourt and F. Irigoin. Scanning polyhedra with do loops. In *Third Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 39–50. ACM Press, 1991.

[And96] Sven Anders. Das Modul FM - eine Umsetzung des Fourier-Motzkin Algorithmus, 1996. Implementation im Rahmen von LooPo, Universität Passau, http://www.fmi.uni-passau.de/loopo/.

[And00] Sven Anders. Analyse und Optimierung des Kommunikationsaufwandes in automatisch parallelisierten Programmen. Master's thesis, Universität Passau, 2000.

[Ban93] Uptal Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Kluwer Academic Publishers, Boston, 1993.

[BS91] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik*, volume 1. B.G. Teubner Verlagsgesellschaft Stuttgart Leipzig und Verlag Nauka Moskau, 25 edition, 1991.

[Che65] N. Chernikova. Algorithm for finding a general formula for the non-negative solution of a system of linear inequalities. *U.S.S.R Computational Mathematics and Mathematical Physics*, 5(2):228–233, 1965.

[DE73] G. B. Danzig and B. C. Eavens. Fourier-Motzkin elimination and its dual. *Journal of Combinational Theory*, A(14):288 – 297, 1973.

[FQ88] Felipe Fernández and Patrice Quinton. Extension of chernikova's algorithm for solving general mixed linear programming problems. Technical Report 437, IRISA, Rennes, France, October 1988.

[GL96] Martin Griebl and Christian Lengauer. The loop parallelizer LooPo. In Michael Gerndt, editor, *Proc. Sixth Workshop on Compilers for Parallel Computers*, volume 21 of *Konferenzen des Forschungszentrums Jülich*, pages 311–320. Forschungszentrum Jülich, 1996.

[Gri96] Martin Griebl. *The Mechanical Parallelization of Loop Nests Containing* while *Loops*. PhD thesis, University of Passau, 1996. Also available as technical report MIP-9701.

[Her92] Dietmar Herrmann. *Algorithmen Arbeitsbuch*. Addison-Wesley, Bonn, München, 1992.

[Köb95] Klaus Köbler. Theoretischer und praktischer Vergleich von Algorithmen zur Lösung von Ungleichungssystemen. Master's thesis, Universität Passau, 1995.

[KPR95] W. Kelly, W. Pugh, and E. Rosser. Code generation for multiple mappings. In *Frontiers '95: The 5th Symposium on the Frontiers of Massively Parallel Computation*. McLean, VA, 1995.

[Krö91] Fred Kröger. *Einführung in die Informatik*. Springer-Verlag, Berlin-Heidelberg, 1991.

[Le 92] Hervé Le Verge. A note on Chernikova's Algorithm. Technical Report 635, IRISA, Rennes, France, Februrary 1992.

[Len93] Christian Lengauer. Loop parallelization in the polytope model. In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 398–416. Springer-Verlag, 1993.

[LfMA] Universität Passau Lehrstuhl für Mathemathik (Algebra). Redlog: The REDUCE Logic System. Verison 2.0 is included in REDUCE Version 3.7. http://www.uni-passau.de/~redlog/.

[LfP] Universität Passau Lehrstuhl für Programmierung. The Polyhedral Loop Parallizer: LooPo. http://www.uni-passau.de/loopo/.

[Loe99] Vincent Loechner. A library for manipulating parameterized polyhedra. Technical report, ICPS, Université de Strasbourg, France, 1999.

[LW93] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36:450–462, 1993.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, II. Princeton University Press, New Jersey, 1953.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer an Combinatorial Optimization*. III. John Wiley & Sons, 1988.

[QRW99] Fabien Quilleré, Sanjay Radjopadhye, and Doran K. Wilde. Generation of efficient nested loops from polyhedra. Unpublished note, IRISA, Rennes, France, 1999.

[Sch94] Alexander Schrijver. *Theory of linear an integer programming*. II. John Wiley & Sons, 1986/94.

[VvDW94] Hervé Le Verge, Vincent van Dongen, and Doran K. Wilde. Loop nest sythesis using the polyhedral library. Technical Report PI 830, IRISA, Rennes, France, 1994.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Comp.*, 5:3–27, 1988.

[Wei94] Volker Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, Universität Passau, 1994.

[Wet95] Sabine Wetzel. Automatic code generation in the polytope model. Master's thesis, Universität Passau, 1995.

[Wil93] Doran K. Wilde. A library for doing polyheral operations. Technical Report PI 758, IRISA, Rennes, France, 1993.

Abbildungsverzeichnis

2.1 Mögliche Halbraumschnitte für die Polyederdarstellung 11
2.2 Darstellung eines Polyeders über Scheitelpunkte und Hyperebenen 12
2.3 Einteilung der Strahlen nach der Beschränkung 20
2.4 Algorithmus zu Berechnung von `Dual`(A) 23
2.5 Prozedur zur Berechnung von `Combine`(r_1, r_2, a) 23
2.6 Prozedur zur Berechnung von `ConstrainL`(L, a) 24
2.7 Prozedur zur Berechnung von `AugmentR`(R, a, r_{new}) 25
2.8 Prozedur zur Berechnung von `ConstrainR`(R, a) 26

3.1 Rekursionsbaum 29
3.2 Polyeder für das Ungleichungssystem 3.1, dargestellt in den Dimensionsachsen x_1 und x_2 bzw. x_2 und x_3. 31
3.3 Lösungsbaum des Beispiels 36
3.4 Projektion eines Polyeders 39
3.5 Redundante Informationen, die bei der Projektion entstehen 40

4.1 Darstellung und Beschreibung zweier durch Domänen beschriebenen Statements (Annahme: $4 \leq M \leq N$) 43
4.2 Projektion eines zweidimensionalen Polytops 44
4.3 Vereinfachung von Domänen: D' =*Simplify* (D, C) 45
4.4 Beispielschranken, wie sie von FM berechnet werden 47
4.5 Darstellung der Domäne: $\{i, j \mid j \leq i \leq N; j \geq 0; i \geq M; i + j \leq N + M; M \geq 2; M < N\}$. 50
4.6 Verschiedene Ansätze für Verschachtelung 54
4.7 Generierter Code für das Beispiel aus Abbildung 4.1 56
4.8 Vergleich der zu betrachtenden Punkte der verschiedenen Ansätze 57
4.9 Aufzählen der Vereinigung der Domänen aus Abbildung 4.1 58
4.10 Vergleich der unterschiedlichen Ansätze 58
4.11 Projektion und Separation am Beispiel 61
4.12 Zweite Projektions- und Separationsebene 62
4.13 Relation zweier Schleifen zueinander ($D_1 \triangleleft D_2$) 68
4.14 Konstruktion des Polyeders in Punkt 5 des Algorithmus 70
4.15 Partielle Ordnung zwischen parametrisierten Schleifen aus Abbildung 4.12 71

Tabellenverzeichnis

2.1 Abhängigkeiten von Instanzen 15

Eidesstattliche Erklärung

Hiermit erkläre ich, daß ich diese Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Alle Ausführungen die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Diese Diplomarbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Passau, den 20.9.2000

Oliver Nyderle