UNIVERSITÄT PASSAU
Fakultät für Mathematik und Informatik

Datenabhängigkeitsanalyse zur Schleifenparallelisierung:

Vergleich und Erweiterung zweier Ansätze

Diplomarbeit

Verfasser: Harald Keimer

Aufgabensteller:
Prof. Christian Lengauer Ph. D.
Lehrstuhl für Informatik
Universität Passau

Betreuer:
Martin Griebl

Passau, den 26. Januar 1997

Vorab möchte ich mich bei Herrn Prof. Christian Lengauer Ph. D. für seine zahlreichen Hilfestellungen und Anregungen bedanken. Mein besonderer Dank gilt meinem Betreuer, Herrn Martin Griebl, der stets für Fragen offen war und mir dabei unermüdlich zur Seite stand. Weiterhin danke ich allen Mitgliedern des *LooPo*-Projektes für ihre Unterstützung: Andreas Dischinger, Nils Ellmenreich, Peter Faber, Robert Günz, Klaus Köbler, Radko Kubias, Frank Schuler, Martina Schumergruber, Sabine Wetzel, Christian Wieninger und Alexander Wüst. Schließlich möchte ich Herrn Jean-François Collard für viele ergiebige Diskussionen Dank sagen.

Zusammenfassung

Mit der Entwicklung von Mehrprozessorsystemen kommt der Parallelisierung sequentieller Programme eine immer größere Bedeutung zu. Weil Algorithmen jedoch auch weiterhin sequentiell entwickelt werden, ist ein Compiler erforderlich, der anschließend die Parallelisierung übernimmt. Die Datenabhängigkeitsanalyse stellt hierbei den ersten Schritt dar und bildet somit die Grundlage für die gesamte Arbeit des Compilers. Aus der Vielzahl verschiedener Ansätze zur Datenabhängigkeitsanalyse werden in dieser Arbeit diejenigen von Paul Feautrier [Fea91] und Utpal Banerjee [Ban93] gegenübergestellt, die sich mit Abhängigkeiten in Schleifenprogrammen befassen. Den Schwerpunkt bildet die Erweiterung der zugrunde liegenden Theorien mit dem Ziel, zwei möglichst gut vergleichbare Verfahren zur Datenabhängigkeitsanalyse zu gewinnen.

Inhaltsverzeichnis

Kapitel 1

Einführung

Mit der Entwicklung von Mehrprozessorsystemen kommt der Parallelisierung sequentieller Programme eine immer größere Bedeutung zu. Weil Algorithmen jedoch auch weiterhin sequentiell entwickelt werden, ist ein Compiler erforderlich, der anschließend die Parallelisierung übernimmt. Die Datenabhängigkeitsanalyse stellt hierbei den ersten Schritt dar und bildet somit die Grundlage für die gesamte Arbeit des Compilers. Von besonderer Bedeutung ist die Genauigkeit der Analyse, weil ein hohes Maß an Parallelität nur durch eine minimale Anzahl von Abhängigkeiten erreicht werden kann. Aus diesem Grund ist eine Vielzahl verschiedener Ansätze zur Datenabhängigkeitsanalyse entwickelt worden, von denen in dieser Arbeit diejenigen von Paul Feautrier [Fea91] und Utpal Banerjee [Ban93] gegenübergestellt werden, die sich mit Abhängigkeiten in Schleifenprogrammen befassen. Den Schwerpunkt bildet die Erweiterung der zugrunde liegenden Theorien mit dem Ziel, zwei möglichst gut vergleichbare Verfahren zur Datenabhängigkeitsanalyse zu gewinnen.

Hierzu werden in Kapitel 2 die Notation sowie die allgemeinen mathematischen Grundlagen und Verfahren vorgestellt. In Kapitel 3 erfolgt die Einführung aller verwendeten Begriffe aus der Parallelisierungstheorie. Die Spezifikation zulässiger Eingabeprogramme findet sich in Kapitel 4. Anschließend wird in den Kapiteln 5 und 6 je ein Ansatz zuerst vorgestellt und dann im erforderlichen Umfang erweitert. Zu diesem Zeitpunkt wird das Ziel einer weitestmöglichen Angleichung der Verfahren erreicht sein. Es folgt in Kapitel 7 die Beschreibung von Erweiterungsmöglichkeiten, die auf beide Ansätze gleichermaßen anwendbar sind, sowie eine abschließende Bewertung in Kapitel 8.

Die erweiterte Datenabhängigkeitsanalyse nach Banerjee wurde im Rahmen dieser Diplomarbeit im *LooPo*-Projekt[1] an der Universität Passau implementiert.

[1] http://www.uni-passau.de/loopo

Kapitel 2

Voraussetzungen, Notation

In den Kapiteln 5 und 6, insbesondere in den Teilen, die sich mit der Erweiterung der bestehenden Theorie befassen, werden Kenntnisse der linearen Algebra vorausgesetzt, die in diesem Abschnitt vorgestellt werden.

2.1 Notation

In dieser Arbeit wird die Notation von Dijkstra und Scholten [DS90] verwendet. Damit gestaltet sich die Schreibweise innerhalb von Beweisen wie folgt:

$$\begin{array}{l} \quad Term_1 \\ op\ \{Erklärung\} \\ \quad Term_2 \end{array}$$

Hierbei stellt *op* wahlweise die Implikation oder die Äquivalenz dar. Auf die Verwendung der Replikation wird an dieser Stelle verzichtet.

Innerhalb eines Termes auftretende Quantifizierungen einer Variablen haben die Form

$$(\ Q <variable> :\ D(<variable>)\ :\ P(<variable>)\ )$$

Dabei steht Q für den Existenz- oder Allquantor, während D den Wertebereich der Variablen angibt. P stellt eine Aussage in Abhängigkeit der Variablen dar.

2.2 Voraussetzungen

2.2.1 Satz (Umkehrrelation)
Seien U und V Vektorräume endlicher Dimension und $f : U \to V$ affin linear mit linearem Anteil f_{lin} und konstantem Anteil c. Sei ferner $v \in Bi(f)$.
Dann gilt für die Urbildmenge $f^{-1}(v)$:

$$f^{-1}(v) \;=\; u + \ker(f_{lin}) \quad \text{mit } f(u) \;=\; v$$

Beweis:

„$\subseteq$“: Sei $y \in f^{-1}(v)$
Dann gilt:

$$\begin{array}{ll} & f(y) \;=\; v \;=\; f(u) \\ \Rightarrow & \{ \text{ alg. Umformung } \} \\ & f(y) - f(u) \;=\; 0 \\ \Leftrightarrow & \{ \; f \text{ affin linear } \} \\ & f(y-u) \;=\; c \\ \Leftrightarrow & \{ \text{ Definition von } f \; \} \\ & f_{lin}(y-u) \;=\; 0 \\ \Rightarrow & \{ \text{ Definition Kern } \} \\ & y - u \in \ker(f_{lin}) \\ \Rightarrow & \{ \text{ alg. Umformung } \} \\ & y \in u + \ker(f_{lin}) \end{array}$$

„$\supseteq$“: Sei $x \in \ker(f_{lin})$.
Dann gilt:

$$\begin{array}{ll} & f(u+x) \;=\; f(u) + f(x) - c \\ \Leftrightarrow & \{ \text{ Definition von } f \; \} \\ & f(u+x) \;=\; f_{lin}(u) + f_{lin}(x) + c \\ \Leftrightarrow & \{ \; x \in \ker(f_{lin}) \; \} \\ & f(u+x) \;=\; f_{lin}(u) + c \\ \Leftrightarrow & \{ \text{ Definition von } f \; \} \\ & f(u+x) \;=\; f(u) \\ \Leftrightarrow & \{ \text{ Definition von } u \; \} \\ & f(u+x) \;=\; v \end{array}$$

□

2.2.2 Definition (allgemein lineare Gruppe)
Sei $\mathbb{K}$ ein beliebiger Körper und $n \in \mathbb{N}$.
Die allgemein lineare Gruppe (i. Z.: GL(n, $\mathbb{K}$)) *bezeichnet die Einheiten im Ring $\mathbb{K}^{n \times n}$.*

2.2.3 Bemerkung
Falls nicht anders angegeben, wird im folgenden $\mathbb{K} = \mathbb{Q}$ zugrunde gelegt.

2.2.4 Definition (Pseudoinverse)
Sei $M \in \mathbb{K}^{m \times n}$ eine Matrix mit maximalem Rang und $l = \min(m, n)$.
Dann bezeichnet

$$M^{\#} = \begin{cases} \left(M^\top M\right)^{-1} M^\top & \text{falls } l = n \\ M^\top \left(M M^\top\right)^{-1} & \text{falls } l = m \end{cases}$$

die Pseudoinverse *von M.*

Für die $l \times l$-Einheitsmatrix E gilt:

$$E = \begin{cases} M^{\#} M & \text{falls } l = n \\ M M^{\#} & \text{falls } l = m. \end{cases}$$

Für $l = n$ heißt $M^{\#}$ Linksinverse, *andernfalls* Rechtssinverse.

2.2.5 Bemerkung
Für $m = n = l$ ergibt sich $M^{\#} = M^{-1}$. Die Pseudoinverse verallgemeinert in diesem Sinne die Matrixinversion.

2.2.6 Definition (unimodular)
Sei $n \in \mathbb{N}$ und $M \in \mathbb{K}^{n \times n}$.
M ist unimodular *genau dann, wenn*

$$|\det(M)| \ = \ 1.$$

2.2.7 Bemerkung
Für M aus 2.2.6 gilt:
M unimodular $\Rightarrow$ $M \in GL(n, \mathbb{K})$.

2.2.8 Definition (echelon)
Seien n, $m \in \mathbb{N}$ und $M \in \mathbb{K}^{n \times m}$.
M ist echelon *genau dann, wenn*

$$\begin{aligned}\big(\,\exists\,\varphi\,:\,\{(1,....,n)\} \overset{\varphi}{\to} \mathbb{N}\,:\, & \\ & (1 \le \varphi(1) < \varphi(2) < \,....\, < \varphi(n)) \\ & \wedge\ (\forall\, i,j\,:\,1 \le i \le n, 1 \le j \le m\,:\, j < \varphi(i) \ \Rightarrow\ m_{i,j}\ =\ 0) \\ & \wedge\ (\forall\, i\,:\,1 \le i \le n\,:\, \varphi(i) \le m\ \Rightarrow\ m_{i,\varphi(i)}\ \neq\ 0)\big).\end{aligned}$$

Von besonderer Bedeutung werden in Kapitel 6 ganzzahlige Lösungen von (Un-)Gleichungssystemen sein. Die folgende Definition sowie das nachstehende Theorem hierzu sind [Ban93] entnommen.

2.2.9 Definition (Echelon-Reduktion)
Seien $m, n \in \mathbb{N}$. Ferner seien $A, S \in \mathbb{K}^{m\times n}$, $U \in \mathbb{K}^{m\times m}$ Matrizen mit Einträgen aus $\mathbb{Z}$.
Die Echelon-Reduktion *ist ein Verfahren, bei dem zu einer gegebenen Matrix A eine unimodulare Matrix U bestimmt wird, sodaß gilt*

$$UA\ =\ S,\ \ S \textit{ echelon}.$$

Ein Algorithmus für dieses Verfahren findet sich in [Ban93], Abschnitt I.2.3.

2.2.10 Theorem (Existenz ganzzahliger Lösungen)
Seien A, U, S die Matrizen aus 2.2.9 und ferner $c \in \mathbb{K}^{1\times n}$.
Zu einem gegebenen Gleichungssystem

$$(x_1, x_2, ..., x_m)\,A\ =\ c$$

existiert eine ganzzahlige Lösung genau dann, wenn

$$(\exists\, t\,:\, t \in \mathbb{K}^{1\times m} \cap \mathbb{Z}^{1\times m}\,:\, tS\ =\ c).$$

Gibt es ein solches t, so läßt sich die Lösungsmenge vollständig durch

$$(x_1, x_2, ..., x_m)\ =\ tU$$

beschreiben.

Beweis:
Sei $t \in \mathbb{Z}^{1\times m}$ mit

$$\begin{aligned} & tS\ =\ c \\ \Leftrightarrow\quad & \{\ \text{Definition von } S\ \} \\ & tUA\ =\ c \\ \Leftrightarrow\quad & \{\ (x_1, x_2, ..., x_m)\,A\ =\ c\ \} \\ & (x_1, x_2, ..., x_m)\ =\ tU \end{aligned}$$

□

2.2.11 Definition (konvex)
Sei V ein Vektorraum über $\mathbb{K}$, $\lambda \in \mathbb{K}$ und $A \subset V$.
A ist konvex *genau dann, wenn*

$$(\forall\, a, b \,:\, a, b \in A \,:\, \{\lambda a + (1-\lambda)\, b : \lambda \in [0,1]\} \subset A).$$

2.2.12 Satz (Schnitt konvexer Mengen)
Sei $I \neq \emptyset$ und $(A_i)_{i \in I}$ eine Familie konvexer Mengen in einem Vektorraum über $\mathbb{K}$.
Dann gilt:

$$\bigcap\, i \,:\, i \in I \,:\, A_i \quad \text{ist konvex.}$$

Beweis:
Seien $a, b \in \bigcap\, i \,:\, i \in I \,:\, A_i$.

$$\begin{array}{ll} & a, b \in \bigcap\, i \,:\, i \in I \,:\, A_i \\ \Leftrightarrow & \quad \{ \text{ Definition Durchschnitt } \} \\ & (\forall\, i \,:\, i \in I \,:\, a, b \in A_i) \\ \Rightarrow & \quad \{ \; A_i \text{ konvex } \} \\ & \{\lambda a + (1-\lambda)\, b : \lambda \in [0,1]\} \subset A \\ \Rightarrow & \quad \{ \; a, b \in \bigcap\, i \,:\, i \in I \,:\, A_i \text{ beliebig } \} \\ & \bigcap\, i \,:\, i \in I \,:\, A_i \text{ konvex.} \end{array}$$

□

2.2.13 Bemerkung
Seien A, V wie in 2.2.11 und bezeichne $\bar{A}$ den Abschluß von A.
Man überlegt sich leicht, daß gilt:

$$A \text{ konvex} \quad \Rightarrow \quad \bar{A} \text{ ist konvex.}$$

2.2.14 Satz
Seien U und V Vektorräume über $\mathbb{K}$, $A \subset U$ konvex und $f : U \to V$ affin linear.
Dann gilt

$$f(A) \quad \textit{ist konvex.}$$

Beweis:

$$
\begin{array}{ll}
& v_1, v_2 \in f(A) \\
\Rightarrow & \{ \text{ Definition } f \ \} \\
& (\exists\, u_1, u_2 \ : \ u_1, u_2 \in U \ : \ f(u_1) = v_1, f(u_2) = v_2) \\
\Rightarrow & \{ \ U \text{ konvex, } \lambda \in [0,1] \ \} \\
& (\lambda u_1 + (1-\lambda) u_2) \in A \\
\Rightarrow & \{ \text{ Definition } f \ \} \\
& f\,(\lambda u_1 + (1-\lambda) u_2) \in f(A) \\
\Rightarrow & \{ \text{ Additivität von } f,\ f = l_{lin} + c \ \} \\
& (f\,(\lambda u_1) + f\,((1-\lambda) u_2) - c) \in f(A) \\
\Rightarrow & \{ \text{ Homogenität von } f \ \} \\
& (\lambda f(u_1) + (1-\lambda)c + (1-\lambda) f(u_2) + \lambda c - c) \in f(A) \\
\Rightarrow & \{ \text{ alg. Umformung } \} \\
& (\lambda v_1 + (1-\lambda) v_2) \in f(A)
\end{array}
$$

□

2.2.15 Definition (konvexe Hülle)
Sei U Vektorraum über $\mathbb{K}$ und $A \subset U$.
Die konvexe Hülle (i. Z.: $co(A)$) *bezeichnet die kleinste konvexe Obermenge von A.*

2.2.16 Satz (Existenz konvexe Hülle)
Seien A und U wie in 2.2.15.
Zu jedem A existiert $co(A)$ in U.

Beweis:
Sei $\mathcal{R}$ das System aller konvexen Obermengen von A in U.

$$
\begin{array}{ll}
& \mathcal{R} \ := \ \{ X \in \wp(U) \mid X \text{ konvex } \wedge \ A \subset X \} \\
\Rightarrow & \{ \ U \in \mathcal{R} \ \} \\
& \mathcal{R} \neq \emptyset,\ A \subset \bigcap Y \ : \ Y \in \mathcal{R} \ : \ Y \\
\Rightarrow & \{ \ Z \ := \ \bigcap Y \ : \ Y \in \mathcal{R} \ : \ Y, \text{ 2.2.12 angewendet auf } Z \ \} \\
& Z \text{ ist konvex} \\
\Rightarrow & \{ \text{ Definition } \mathcal{R} \ \} \\
& Z \in \mathcal{R} \\
\Rightarrow & \{ \text{ Definition } Z \ \} \\
& Z \text{ ist die kleinste Menge in } \mathcal{R}
\end{array}
$$

□

2.2.17 Definition (abgeschlossener Halbraum)
Sei V ein Vektorraum über $\mathbb{K}$ und $H \subset V$.
H heißt abgeschlossener Halbraum *genau dann, wenn*

$$(\exists\, a \,:\, a \in V \,:\, (\exists\, \beta \,:\, \beta \in \mathbb{K} \,:\, H \;=\; \{x \mid ax+\beta \geq 0\})).$$

2.2.18 Bemerkung
Da später stets auf abgeschlossene Halbräume Bezug genommen wird, entfällt im folgenden ein Hinweis auf die Abgeschlossenheit.

2.2.19 Definition (Polyeder)
Ein Polyeder *bezeichnet den Durchschnitt endlich vieler Halbräume.*

2.2.20 Bemerkung
Ein Halbraum veranschaulicht geometrisch die Lösungsmenge einer Ungleichung, ein Polyeder die eines Ungleichungssystems.

2.2.21 Definition (Polytop)
Sei P Polyeder.
P ist ein Polytop *genau dann, wenn gilt*

$$P \textit{ ist beschränkt.}$$

2.2.22 Bemerkung
Jeder Halbraum ist konvex und damit nach 2.2.12, 2.2.19 auch jeder Polyeder, insbesondere also auch jedes Polytop.

2.2.23 Satz
Sei wiederum V ein Vektorraum über $\mathbb{K}$, $P \subset V$ Polyeder und $i,\, j \in \mathbb{N}$. Ferner sei $H = \{h_1, h_2, ...\}$ eine endliche Menge weiterer Halbräume in V, welche die folgende Eigenschaft besitzen:

$$\begin{array}{ll} & (\forall\, i \,:\, 1 \leq i \leq |H| \,:\, P^+ \neq P \cap \bigcap\, j \,:\, 1 \leq j \leq |H|\,,\, i \neq j \,:\, h_i) \\ \text{mit} & P^+ = P \cap \bigcap\, h \,:\, h \in H \,:\, h. \end{array}$$

$P \backslash P^+$ ist genau dann konvex, wenn gilt

$$H \text{ ist höchstens einelementig.}$$

Beweis:

$\Leftarrow:$ Klar.

$\Rightarrow$: Zeige: $|H| \geq 2 \quad \Rightarrow \quad P \backslash P^+$ nicht konvex.

Die einen Halbraum h begrenzende Hyperebene wird im folgenden mit δh bezeichnet.

Es sind die folgenden zwei Fälle zu unterscheiden:

1.: $P \backslash P^+$ zerfällt in zwei oder mehr Zusammenhangskomponenten und ist somit offensichtlich nicht konvex.

2.: $P \backslash P^+$ ist zusammenhängend.

$$
\begin{array}{ll}
 & \text{wähle } h_i \in H \text{ beliebig} \\
\Rightarrow & \{ \ |H| \geq 2 \ \} \\
 & (\exists\, h_j \ : \ h_j \in H \ : \ h_i \neq h_j) \\
\Rightarrow & \{ \ (2.2.23.1) \ \} \\
 & (\exists\, p, q \ : \ p \in h_i, q \in h_j \ : \ p, q \notin \delta h_i \cap \delta h_j) \\
\Rightarrow & \{ \ P^+ \text{ konvex } \} \\
 & \overline{pq} \in P^+ \\
\Rightarrow & \{ \ \text{Differenzmengenbildung } \} \\
 & p, q \in \overline{P \backslash P^+}, \ \text{ jedoch } \ \overline{pq} \notin \overline{P \backslash P^+} \\
\Rightarrow & \{ \ \text{Bemerkung 2.2.13 } \} \\
 & P \backslash P^+ \text{ nicht konvex.}
\end{array}
$$

□

2.3 Verfahren zur Lösung von Ungleichungssystemen

Aus einer Vielzahl von Methoden zur Behandlung von Ungleichungssystemen werden hier stellvertretend PIP [Fea89] und Fourier-Motzkin [Ban93] genannt. Beide Methoden sind Bestandteil des *LooPo*-Projektes.

2.3.1 PIP

Mit PIP stellte Feautrier die Implementation eines Algorithmus zur Lösung parametrischer Integerprobleme vor. Er dient dazu, das lexikografische Minimum einer Menge von Integervektoren zu finden. (Eine Definition der lexikografischen Ordnung erfolgt Abschnitt 3.2.)

Da sich die Maximierung bezüglich einer gegebenen Ordnung als ein zur Minimierung duales Problem darstellen läßt [Wie95], kann dieser Algorithmus ebenso zur Bestimmung des Maximums verwendet werden.

2.3.2 Fourier-Motzkin

Der französische Mathematiker Fourier entwickelte im Jahre 1827 ein Eliminationsverfahren zur Lösung linearer Ungleichungen. Diese Methode wurde später durch Dines und Motzkin wiederentdeckt und ist heute unter dem Namen Fourier-Motzkin-Methode bekannt.

Näheres zu diesem Algorithmus findet sich in [Ban93].

2.4 Minimierung und Maximierung von Zielfunktionen

Um nun auch das Minimum bzw. Maximum des Wertebereiches einer affin linearen Funktion bestimmen zu können, wird hier ein Verfahren vorgestellt, durch das die Verwendung eines der in 2.3 genannten Algorithmen ermöglicht wird. Dieser Ansatz geht auf eine Erweiterung des Algorithmus von Fourier-Motzkin durch G. B. Dantzig [Dan66] zurück.

Seien n, m, $q \in \mathbb{N}$, $n \geq m$, $A \in \mathbb{K}^{q \times n}$, $C \in \mathbb{K}^{q \times 1}$,
$x := (x_1, x_2, ..., x_m)$, $p := (p_{m+1}, p_{m+2}, ..., p_n)$,
$\alpha := (\alpha_1, \alpha_2, ..., \alpha_m) \in \mathbb{K}^m$ und $\beta := (\beta_{m+1}, \beta_{m+2}, ..., \beta_n) \in \mathbb{K}^{n-m}$.
Dann wird durch

$$A\,(x|p)^\top + C \geq 0$$

ein aus q Ungleichungen in m Variablen $(x_1, ..., x_m)$ und $(n-m)$ Parametern $(p_{m+1}, ..., p_n)$ bestehendes Ungleichungssystem beschrieben. Die Lösungsmenge dieses Systems sei L.

Ziel ist es, die extremalen Werte eines affin linearen Funktionals über $\mathbb{K}$

$$f : L \to \mathbb{K} : (x_1, x_2, ..., x_m) \to \underbrace{\sum_{i=1}^{m} \alpha_i\, x_i + \sum_{j=m+1}^{n} \beta_j\, p_j + \gamma}_{=:\ w}$$

zu bestimmen.

Hierzu wird für den Funktionsausdruck w von f eine neue Variable z eingeführt.

Durch Aufspaltung der Gleichung $z - w = 0$ in zwei Ungleichungen ist es nun möglich, letztere dem obigen System hinzuzufügen:

$$\left(\begin{array}{c|cccccc} 0 & & & & & & \\ \vdots & & & A & & & \\ 0 & & & & & & \\ 1 & -\alpha_1 & \dots & -\alpha_m & -\beta_{m+1} & \dots & -\beta_n \\ -1 & \alpha_1 & \dots & \alpha_m & \beta_{m+1} & \dots & \beta_n \end{array}\right)(z|x|p)^\top + \begin{pmatrix} c_1 \\ \vdots \\ c_q \\ -\gamma \\ \gamma \end{pmatrix} \geq 0$$

Das resultierende System in $(m+1)$ Variablen läßt sich nun mit einem der zuvor genannten Verfahren lösen. Hierbei stellen die für z gefundenen Schranken den minimalen bzw. maximalen Funktionswert von f dar.

2.4.1 Beispiel

Es gelten weiterhin die in diesem Abschnitt eingeführten Bezeichnungen:

Gegeben: $*\quad n = m = 2,\ q = 4,$

$$* \quad A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ -1 & -1 \\ 1 & -1 \end{pmatrix}, \quad C = \begin{pmatrix} -4 \\ 2 \\ 8 \\ 2 \end{pmatrix},$$

$$* \quad h = (h_1, h_2) = (-x_1 + 11, x_1 + x_2 + 3)$$

Gesucht: Extremalwerte von h_1, h_2

Geometrisch läßt sich das gegebene Ungleichungssystem als das Polytop P in Abbildung 2.1 veranschaulichen (vgl. 2.2.20), welches durch h auf P' abgebildet wird.

Die durch die Einführung der Variablen z_1 und z_2 für h_1 und h_2 entstandenen Gleichungen

$$\begin{array}{rcrcrcrcl} z_1 & + & & & x_1 & - & & & 11 = 0, \\ & & z_2 & - & x_1 & - & x_2 & - & 3 = 0 \end{array}$$

werden in je zwei Ungleichungen aufgespalten, welche dann dem gegebenen System hinzugefügt werden.

Man erhält:

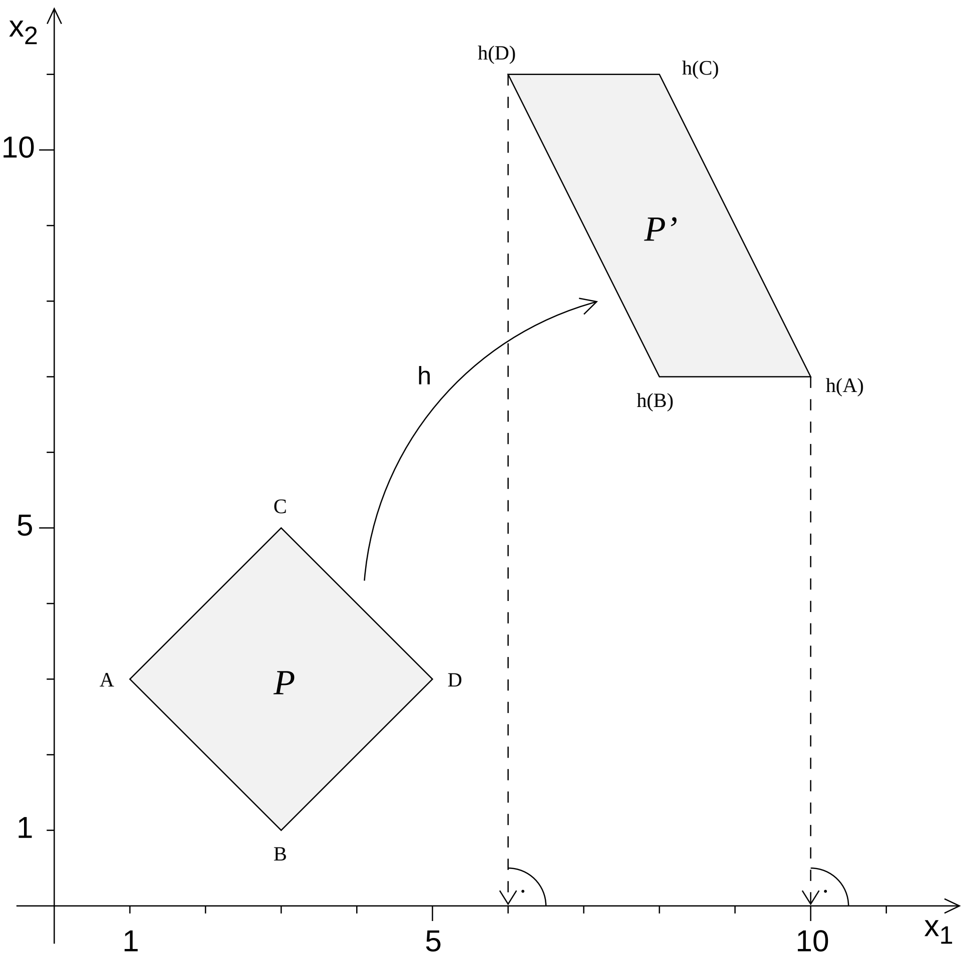


Abbildung 2.1: Minimierung und Maximierung von Zielfunktionen

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 \\ 0 & -1 & 1 & 1 \end{pmatrix} (z_1\, z_2\, x_1\, x_2)^\top + \begin{pmatrix} 4 \\ 2 \\ 8 \\ 2 \\ -11 \\ 11 \\ -3 \\ 3 \end{pmatrix} \geq 0$$

Mit Hilfe eines der unter 2.3 angeführten Verfahren ergibt sich für z_1:

$$6 \leq z_1 \leq 10$$

Diese Lösung entspricht in der geometrischen Darstellung der Orthogonalprojektion von P' auf die x_1-Achse.

Für z_2 ergeben sich in Abhängigkeit von z_1 als Grenzen :

$$\max(-2z_1+23,\, 7) \leq z_2 \leq \min(-2z_1+27,\, 11)$$

Kapitel 3

Definitionen, Grundlagen

Es werden nun die im folgenden benötigten Begriffe für die Datenabhängigkeitsanalyse eingeführt. Hierbei bleibt die in [Ban93] sowie [Fea91] eingeführte Notation so weit wie möglich erhalten.

3.1 Räume

3.1.1 Definition (Indexraum)
Sei S ein Statement, welches von n Schleifen mit Indizes $i_1, ..., i_n$ umgeben wird. Für jeden einzelnen Index i_j gibt es durch die Verwendung des Minimum-/Maximumoperators (vgl. Kapitel 4) bis zu $k_{(j)}$ untere und $k^{(j)}$ obere Grenzen (i. Z.: l_j bzw. u_j).
Der Indexraum *D_S des Statements S wird definiert durch:*

$$D_S = \left\{ (i_1, ..., i_n) \;\middle|\; \begin{array}{l} (i_1, ..., i_n) \in \mathbb{Z}^n \wedge \\ \big(\forall\, j \;:\; 1 \le j \le n \;:\; \big(\forall\, p \;:\; 1 \le p \le k_{(j)} \;: \\ (\forall\, q \;:\; 1 \le q \le k^{(j)} \;:\; l_{j_p} \le i_j \le u_{j_q}))\big)\big) \end{array} \right\}$$

3.1.2 Bemerkung
In der obigen Definition beschreibt der zweite Teil der Konjunktion ein Ungleichungssystem, das sich nach 2.2.20 als Polyeder veranschaulichen läßt. Geometrisch stellt D_S also den Schnitt dieses Polyeders mit $\mathbb{Z}^n$ dar.

3.1.3 Definition (Dimension Indexraum)
Die Anzahl der ein Statement S umgebenden Schleifen gibt die Dimension *des Indexraumes D_S an.*

3.1.4 Bemerkung
Man beachte, daß dieser Dimensionsbegriff im allgemeinen *nicht* mit der Dimension des assoziierten Polyeders in seiner geometrischen Darstellung übereinstimmt! So ist die Dimension des Indexraumes im nebenstehenden Programm 2 (Anzahl der Schleifen), die geometrische Dimension des beschriebenen Polyeders beträgt jedoch 1, weil es in i_2-Richtung keine Ausdehnung besitzt.

DO $i_1 = 1,\ 5$
  DO $i_2 = 5,\ 5$
    $A[i_2,\ i_1,\ 1] = i{*}j$
  END DO
END DO

3.1.5 Definition (Operation)
Sei S ein Statement und $\vec{i} \in D_S$.
Dann wird durch das Paar $<S, \vec{i}>$ eine Operation *des Statements S zum Zeitpunkt $\vec{i}$ bezeichnet.*

3.1.6 Definition (Datenraum)
Durch ein Array $A[a_1,\ \ldots.,\ a_n]$ wird eine Teilmenge eines n-dimensionalen Raumes beschrieben. Dieser Raum wird als Datenraum *bezeichnet.*

3.1.7 Definition (Dimension Datenraum)
Mit den Bezeichnungen aus Definition 3.1.6 gibt die Anzahl n der Komponenten eines Arrays die Dimension *seines Datenraumes an.*

3.1.8 Bemerkung
In Analogie zu 3.1.4 entspricht dieser Dimensionsbegriff ebenfalls *nicht* dem der geometrischen Veranschaulichung. Auch hiefür gibt das Programm aus Bemerkung 3.1.4 ein Beispiel. Die Dimension des Datenraumes beträgt 3 (Anzahl der Komponenten). Weil aber das beschriebene Polyeder nur in der Richtung der zweiten Komponente (i_1) eine Ausdehnung aufweist, ist seine geometrische Dimension 1.

3.1.9 Definition (Indexfunktion)
Sei S ein Statement mit Indexraum D_S, das von n Schleifen mit Indizes $i_1, \ldots., i_n$ umgeben wird, und X ein in S verwendetes Array mit m-dimensionalem Datenraum, $n, m \in \mathbb{N},\ n, m \geq 1$.
Dann existiert für Progamme, die der Eingabespezifikation (Kapitel 4) genügen stets eine affin lineare Funktion $g : \mathbb{K}^n \to \mathbb{K}^m$, so daß gilt

$$g\left((i_1, \ldots., i_n)\right) \quad \textit{stellt den Indexausdruck von } X \textit{ dar.}$$

Die Indexfunktion g von X *(i. Z.: g_X) ist das Bindeglied zwischen dem Indexraum D_S und dem Datenraum von X. Sie bildet jeden Punkt von*

D_S auf denjenigen Punkt des Datenraumes ab, der während der Operation $<S,\vec{i}>$ durch X referenziert wird.

3.1.10 Bemerkung
Für den Fall, daß es sich bei X aus 3.1.9 um einen Skalar handelt, wird g_X als die leere Funkion angenommen.

3.2 Ordnungen

3.2.1 Definition (lexikografische Ordnung)
Seien $\vec{i} = (i_1, ..., i_n)$ und $\vec{j} = (j_1, ..., j_n)$ zwei beliebige Vektoren und weiter $l \in \mathbb{N}$, $1 \leq l \leq n$.

Der Vektor $\vec{i}$ ist bezüglich der Komponente l lexikografisch kleiner *als der Vektor $\vec{j}$ (i. Z.: $\vec{i} \ll^l \vec{j}$) genau dann, wenn*

$$(\forall\, m \,:\, 1 \leq m < l,\, m \in \mathbb{N} \,:\, i_m = j_m) \wedge (i_l < j_l)\,. \tag{3.2.1.1}$$

Folglich ist die lexikografische Ordnung *(i. Z.: $\ll$) eine Disjunktion n linearer Prädikate:*

$$\vec{i} \ll \vec{j} \iff (\exists\, l \,:\, 1 \leq l \leq n \,:\, \vec{i} \ll^l \vec{j}) \tag{3.2.1.2}$$

Hierauf aufbauend werden $\underline{\ll}^l$ und $\underline{\ll}$ definiert durch

$$\vec{i} \,\underline{\ll}^l\, \vec{j} \iff \vec{i} \ll^l \vec{j} \,\vee\, \vec{i} = \vec{j} \tag{3.2.1.3}$$

und

$$\vec{i} \,\underline{\ll}\, \vec{j} \iff \vec{i} \ll \vec{j} \,\vee\, \vec{i} = \vec{j}. \tag{3.2.1.4}$$

Die lexikografische $\ll$-Ordnung gibt also an, in welcher Komponente sich die Vektoren $\vec{i}$ und $\vec{j}$ *erstmalig* unterscheiden.

3.2.2 Bemerkung
Die von [Ban93] und [Fea91] für die lexikografische Ordnung verwendeten Bezeichnungen unterscheiden sich in ihrer Indizierung. Für Feautrier besteht im Falle $\vec{i} \ll^l \vec{j}$ Gleichheit bis *einschließlich* zur Komponente l. Entsprechend gilt dann $0 \leq l < n$.

Um einen ständigen Wechsel in der Notation zu vermeiden, wird die lexikographische Ordnung durchgängig im Sinne der Definition 3.2.1 verwendet.

3.2.3 Definition (textuelle Ordung)
Seien S und T Statements.
Die textuelle Ordnung *(i. Z.: $<_{txt}, \leq_{txt}$) wird ausschließlich durch die Notation des Eingabeprogrammes bestimmt:*

$$S <_{txt} T \iff S \text{ steht im Programm vor } T \tag{3.2.3.1}$$

beziehungsweise

$$S \leq_{txt} T \iff S \text{ steht im Programm vor } T \ \vee \ S = T \tag{3.2.3.2}$$

Diese beiden Ordnungen stellen die Grundlage für alle weiteren Überlegungen dar. Wünschenswert ist an dieser Stelle jedoch die Möglichkeit, nicht nur (Iterations-)Vektoren oder Statements zueinander in Beziehung setzen zu können, sondern auch Operationen. Benötigt wird also eine weitere Ordnung, welche den Zusammenhang zwischen diesen beiden zuvor genannten Ordnungen herstellt. Es handelt sich hierbei um das

3.2.4 Definition (Sequenzierungsprädikat)
Seien S und T wiederum Statements, N_{ST} die Anzahl der S und T umgebenden Schleifen und $\vec{i} \in D_S$, $\vec{j} \in D_T$.
Das Sequenzierungsprädikat *sagt aus, ob die Operation $<S, \vec{i}>$ vor der Operation $<T, \vec{j}>$ ausgeführt wird (i. Z.: $<S, \vec{i}> \prec <T, \vec{j}>$).*
Es gilt

$$<S, \vec{i}> \prec <T, \vec{j}> \iff \vec{i} \ll \vec{j} \ \vee \ \left(\vec{i} = \vec{j} \ \wedge \ S <_{txt} T\right) \tag{3.2.4.1}$$

In Erweiterung von (3.2.1.2) *zerfällt das* Sequenzierungsprädikat *in $N_{ST}+1$ lineare Prädikate*

$$\begin{aligned} &\vec{i} \prec^{l} \vec{j}, \quad \forall\, l : 1 \leq l \leq N_{ST}, \\ &\vec{i} = \vec{j} \ \wedge \ S <_{txt} T \end{aligned} \tag{3.2.4.2}$$

Für den Fall der Gültigkeit von $<S, \vec{i}> \prec <T, \vec{j}>$ bei lexikografischer Gleichheit von $\vec{i}$ und $\vec{j}$ wird die bestimmende textuelle Ordnung in Fortführung von Bemerkung 3.2.2 durch $<S, \vec{i}> \prec^{1+N_{ST}} <T, \vec{j}>$ ausgedrückt.

3.2.5 Bemerkung

Durch Gleichung (3.2.1.4) wird eine Totalordnung definiert, und die durch (3.2.1.2) bzw. (3.2.3.1) gegebenen Striktordnungen (d. h.: die definierte Relation ist irreflexiv und transitiv) sind linear (d. h.: es existieren keine paarweise unvergleichbaren Elemente). Folglich erklärt das Sequenzierungsprädikat als Komposition der lexikografischen sowie der textuellen Ordnung auf der Menge aller ausgeführten Operationen ebenfalls eine lineare Striktordnung.

Gegenstand vieler Überlegungen werden Extrema einer gegebenen Menge bezüglich einer bestimmten Ordnung sein. Hierzu wird die folgende Schreibweise vereinbart.

Notation: *Den Funktionen* min *und* max *wird die jeweilige Ordnung beigestellt (etwa:* $\min_{\ll}$, $\max_{\prec}$*). Ist keine Ordnung angegeben, so gilt die übliche* $\leq$*-Ordnung auf den Elementen von* $\mathbb{K}$.

3.2.6 Bemerkung

Aus Definition 3.2.4 geht hervor, daß die lexikografische Ordnung der Iterationsvektoren *eines* Statements äquivalent zu der durch das Sequenzierungsprädikat gegebenen Ordnung auf den Operationen dieses Statements ist.

3.3 Abhängigkeiten

3.3.1 Definition (Abhängigkeit)

Seien S und T zwei nicht notwendigerweise verschiedene Statements, die beide das Array X verwenden. Die zugehörigen Indexfunktionen seien g_S und g_T.
Das Statement T ist abhängig *vom Statement S (i. Z.: $S\delta T$) genau dann, wenn es Operationen $<S,\vec{i}>$, $<T,\vec{j}>$ gibt, für die gilt:*

$$\vec{i} \in D_S,\ \vec{j} \in D_T, \tag{3.3.1.1}$$

$$g_S\left(\vec{i}\right) = g_T\left(\vec{j}\right), \tag{3.3.1.2}$$

$$\text{und} \quad <S,\vec{i}> \ \prec \ <T,\vec{j}>. \tag{3.3.1.3}$$

Für die Art der Abhängigkeit ergeben sich vier Kombinationsmöglichkeiten:

Zugriff in S	*Zugriff in T*	*Art*
schreibend	*schreibend*	*output*
schreibend	*lesend*	*true*
lesend	*schreibend*	*anti*
lesend	*lesend*	*input*

Durch (3.3.1.1) wird garantiert, daß es sich bei $<S,\vec{i}>$ und $<T,\vec{j}>$ um tatsächlich ausgeführte Operationen handelt. Gleichung (3.3.1.2) beschreibt den Speicherkonflikt, der durch den Zugriff auf X erzeugt wird. Durch (3.3.1.3) wird zum Ausdruck gebraucht, daß der Zugriff im Statement S vor dem Zugriff im Statement T erfolgt. Jedoch muß es sich bei der Operation $<S,\vec{i}>$ nicht um die letzte aller vor $<T,\vec{j}>$ ausgeführten Operationen handeln, die zu einem Speicherkonflikt führen. Eine Abhängigkeit im Sinne von Definition 3.3.1 kennzeichnet folglich *nicht* den Datenfluß.

3.3.2 Bemerkung

Die in 3.3.1 eingeführte Notation für eine Abhängigkeit erfährt gelegentlich verschiedene Erweiterungen:

- Die Art einer Abhängigkeit kann dem δ-Symbol beigestellt werden. So bezeichnet etwa $S\delta^a T$ eine anti-Abhängigkeit von S nach T.

- Oftmals ist es erforderlich, nicht nur die gesamte Abhängigkeit zweier Statements, sondern auch den Zusammenhang zwischen einzelnen Operationen darstellen zu können. Zu diesem Zweck wird die Notation sinngemäß auf Operationen erweitert. Es bezeichnet $<S,\vec{i}>\delta<T,\vec{j}>$ die Abhängigkeit einer Operation des Statements T von einer Operation von S.

Eine Veranschaulichung der Abhängigkeiten gibt der

3.3.3 Definition (Abhängigkeitsgraph)

Ein Abhängigkeitsgraph G ist ein Tupel $G = (V, E)$. Dabei gilt:

- *V ist die Knotenmenge des Graphen und stellt die Zuweisungen des Programms dar. Zwischen diesen Statements und den Knoten existiert eine Bijektion.*

- *die Kantenmenge E enthält für jede Abhängigkeit $S\delta T$ eine Kante zwischen den Knoten, die S und T repräsentieren. Auch diese Zuordnung ist bijektiv.*

```
for ( i = 1; i ≤ 10;  i++ ) {
    X[i,0] = Z[0];                  {S1}
    for ( j = 1; j ≤ 10;  j++ ) {
        X[i,j] = X[i−1, j−1];       {S2}
    }
}
```

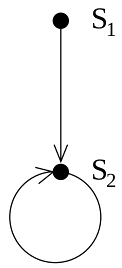


Abbildung 3.1: Beispielprogramm 1 mit Abhängigkeitsgraphen

3.3.4 Beispiel

Sei das Programm in Abbildung 3.1 gegeben. Es enthält die true-Abhängigkeiten $S_1 \delta S_2$ und $S_2 \delta S_2$. Man beachte, daß keine output-Abhängigkeit zwischen beiden Statements besteht, weil es aufgrund der zweiten Komponente der Indexfunktionen zu keinem Speicherkonflikt bei den schreibenden Zugriffen kommen kann.

3.3.5 Definition (Quellraum, Zielraum)

Seien S und T Statements und bezeichne e die Abhängigkeit $S\delta T$.

Der Quellraum der Abhängigkeit e (i. Z.: $QR(e)$) besteht aus genau den Vektoren des Indexraumes D_S für die gilt:

$$(\exists\, \vec{j} \,:\, \vec{j} \in D_T \,:\, <S, \vec{i}> \delta_e <T, \vec{j}>) \tag{3.3.5.1}$$

Umgekehrt wird der Zielraum von e (i. Z.: $ZR(e)$) definiert durch:

$$ZR(e) = \left\{ \vec{j} \mid \vec{j} \in D_T \wedge (\exists\, \vec{i} \,:\, \vec{i} \in D_S \,:\, <S, \vec{i}> \delta_e <T, \vec{j}>) \right\} \tag{3.3.5.2}$$

3.3.6 Definition (direkte Abhängigkeiten)

Mit den Bezeichnungen aus 3.3.1 gelte $S\delta T$. Sei U ein beliebiges Statement, in dem X auftritt, g_U die zugehörige Indexfunktion, $\vec{k} \in D_U$ und $<U, \vec{k}>$ eine von $<S, \vec{i}>$ und verschiedene Operation.
Die Abhängigkeit ist direkt *genau dann, wenn*

$$g_U\left(\vec{k}\right) = g_T\left(\vec{j}\right), <U, \vec{k}> \prec <T, \vec{j}> \quad \Rightarrow \quad <U, \vec{k}> \prec <S, \vec{i}>. \tag{3.3.6.1}$$

Durch (3.3.6.1) wird ausgedrückt, daß die Speicherzelle $X\left[g_S\left(\vec{i}\right)\right]$ durch die Operation $<S,\vec{i}>$ letztmalig vor dem Zugriff in $<T,\vec{j}>$ referenziert wird.

3.3.7 Definition (flow-Abhängigkeit)
Eine direkte true-Abhängigkeit heißt flow- *bzw.* Fluß-Abhängigkeit.

3.3.8 Bemerkung
Es gibt für input-, output- und anti-Abhängigkeiten keine zur true-Abhängigkeit äquivalente Möglichkeit, die Direktheit durch die Namensgebung auszudrücken.
Aus diesem Grund wird im folgenden für diese Arten von Abhängigkeiten ausdrücklich auf die Direktheit hingewiesen, falls diese gegeben ist. Fehlt ein Hinweis, so ist davon auszugehen, daß die jeweilige Abhängigkeit nicht notwendigerweise den Datenfluß beschreibt.

3.3.9 Korollar
Bezeichne e die Abhängigkeit $S\delta T$.

Dann gilt:

$$S\delta T \text{ direkt} \quad \Longrightarrow \quad (\exists\, \varphi : \varphi \text{ Bijektion} : QR(e) \stackrel{\varphi}{\to} ZR(e))$$

Beweis:
Surjektivität folgt unmittelbar aus Definition 3.3.5.

Zeige für $\varphi : QR(e) \to ZR(e)$

$$\varphi \text{ nicht injektiv} \quad \Longrightarrow \quad S\delta T \text{ nicht direkt}$$

Seien $\vec{s_1}, \vec{s_2} \in D_S$ und $\vec{t} \in D_T$ mit $\vec{s_1} \ll \vec{s_2}$. Gelte zusätzlich $\vec{s_1} \neq \vec{t} \neq \vec{s_1}$, falls $S = T$. Die zugehörigen Indexfunktionen werden mit g_S und g_T bezeichnet.

$$\begin{array}{ll} & <S,\vec{s_1}>\delta<T,\vec{t}> \;\wedge <S,\vec{s_2}>\delta<T,\vec{t}> \\ \Rightarrow & \quad \{ \text{ Definition 3.3.1 } \} \\ & g_S\left(\vec{s_1}\right) = g_S\left(\vec{s_2}\right) = g_T\left(\vec{t}\right) \\ \Rightarrow & \quad \{ \; \vec{s_1} \ll \vec{s_2}, \text{ Definition 3.3.1 } \} \\ & <S,\vec{s_1}>\delta<S,\vec{s_2}> \\ \Rightarrow & \quad \{ \text{ Definition 3.3.6 } \} \\ & <S,\vec{s_1}>\delta<T,\vec{t}> \text{ indirekt} \end{array}$$

$\square$

3.3.10 Bemerkung
Man beachte, daß der Umkehrschluß von 3.3.9 falsch ist.

3.3.11 Bemerkung
Eine äquivalente Charakterisierung direkter Abhängigkeiten ergibt sich aus der Definition des Abhängigkeitsgraphen: eine Kante $e = (S,T) \in E$ repräsentiert genau dann eine direkte Abhängigkeit, wenn es in dem Graphen $G' = (V, E\backslash\{e\})$ keinen Pfad von S nach T gibt, sich e also nicht transitiv durch andere Kanten darstellen läßt.

Bei der Schleifenparallelisierung spielen input-Abhängigkeiten eine untergeordnete Rolle. Zwar beschreiben sie die zeitliche Abfolge zweier lesender Zugriffe auf eine Speicherzelle, jedoch ist diese nicht von Bedeutung, weil der Inhalt dieser Zelle nicht verändert wird. Aus diesem Grund werden input-Abhängigkeiten in dieser Arbeit nicht betrachtet. Der Wegfall dieser Abhängigkeiten hat jedoch eine wichtige Konsequenz: die beiden zuvor genannten Charakterisierungen direkter Abhängigkeiten sind nicht länger äquivalent! Das verdeutlicht folgendes Beispiel.

3.3.12 Beispiel
Gegeben sei das Programm in Abbildung 3.2. Neben den direkten Abhängigkeiten $<S_1,()> \delta^f <S_2,()>$ und $<S_2,()> \delta^i <S_3,()>$ enthält es die indirekte true-Abhängigkeit $<S_1,()> \delta^f <S_3,()>$. Wird nun die Inputabhängigkeit außer acht gelassen, so bleibt die true-Abhängigkeit $<S_1,()> \delta^f <S_3,()>$ gemäß Definition 3.3.6 indirekt. Jedoch führt die Anwendung von Bemerkung 3.3.11 zu dem Ergebnis, daß diese Abhängigkeit nun direkt ist.

Konsequenz hieraus ist, daß bei der Entscheidung hinsichtlich der Direktheit einer Abhängigkeit stets die Definition aus Bemerkung 3.3.11 zugrunde gelegt wird.

3.3.13 Definition (Distanzvektor)
Mit den Bezeichnungen aus 3.3.1 gelte $S\delta T$, wobei S und T von denselben Schleifen umgeben werden.
Der Distanzvektor $\vec{d}$ *wird definiert durch:*

$$\vec{d} := \vec{j} - \vec{i} \qquad (3.3.13.1)$$

3.3.14 Korollar
Für einen Distanzvektor $\vec{d}$ gilt stets

$$\vec{d} \not< 0.$$

$a[0] = b[0];$	$\{S1\}$
$b[0] = 3 * a[0];$	$\{S2\}$
$c[0] = 5 * a[0];$	$\{S3\}$

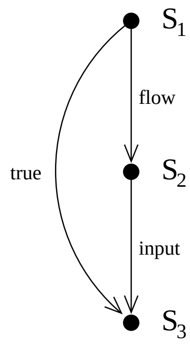


Abbildung 3.2: Beispielprogramm 2 mit Abhängigkeitsgraphen

Beweis:
Die Behauptung folgt sofort aus (3.2.1.2), (3.3.1.3) und 3.3.13.

□

3.3.15 Definition (uniform)
Sei $\vec{d}$ Distanzvektor einer Abhängigkeit.
Die Abhängigkeit heißt uniform, *genau dann wenn gilt*

$$\vec{d} \textit{ ist konstant.}$$

3.3.16 Definition (schleifengetragen, statementgetragen)
Sei wiederum $\vec{d}$ Distanzvektor einer Abhängigkeit.

Dann wird die Abhängigkeit von Schleife l getragen, *falls*

$$0 \ll^l \vec{d}$$

gilt. Existiert kein solches l, ist also $\vec{d} = 0$, so heißt die Abhängigkeit statementgetragen.

3.3.17 Bemerkung
Bei der Definition der Begriffe *schleifengetragen* und *statementgetragen* wird üblicherweise die Uniformität der Abhängigkeit vorausgesetzt. Definition 3.3.16 enthält diese Annahme nicht.

Zu beachten ist jedoch, daß für nicht-uniforme Abhängigkeiten keine allgemeine Aussage darüber gemacht werden kann, ob sie schleifen- oder statementgetragen sind.

Beispielsweise bestehen für einen Distanzvektor $\vec{d} = (0, i, j)$, $0 \leq i, j \leq 2$, drei Möglichkeiten. Für $i = j = 0$ ist die Abhängigkeit statementgetragen, andernfalls schleifengetragen. In letzterem Fall wird sie für $i \geq 1$ von der zweiten und für $i = 0 \wedge j \geq 1$ von der dritten Schleife getragen.

Eine Aussage läßt sich bei nicht-uniformen Abhängigkeiten also nur für bestimmte Mengen von Operationen machen.

3.3.18 Definition (Richtungsvektor)
Sei $\vec{d}$ Distanzvektor einer Abhängigkeit.

Dann ist der Richtungsvektor $\vec{r}$ *definiert durch*

$$\vec{r} = \operatorname{sign}\left(\vec{d}\right),$$

wobei die Signumsfunktion sign *komponentenweise und für $\vec{d} = ()$ der Richtungsvektor $\vec{r} = ()$ erklärt wird.*

3.3.19 Bemerkung
Der Distanz- und damit auch der Richtungsvektor sind für perfekt verschachtelte Schleifen definiert worden. Die für den Fall einer nicht perfekten Verschachtelung erforderliche Erweiterung sieht vor, daß nur jeweils soviele Komponenten der Vektoren $\vec{i}$ und $\vec{j}$ einbezogen werden, wie die Zahl N_{ST} der Schleifen angibt, die sowohl S, als auch T umgeben.

Mit der Projektion π läßt sich (3.3.13.1) für den allgemeinen Fall schreibem als:

$$\vec{d} := \pi_{<1,\ldots,N_{ST}>}\,\vec{j} \;-\; \pi_{<1,\ldots,N_{ST}>}\,\vec{i}$$

Um Abhängigkeiten vollständig zu beschreiben, benötigt man neben dem Quell- und dem Zielraum dieser Abhängigkeit noch eine Funktion, die den Zusammenhang zwischen beiden Räumen herstellt. Bei dieser Funktion handelt es sich um die von Feautrier eingeführte h-Transformation.

3.3.20 Definition (h-Transformation)
Mit den Bezeichnungen aus Korollar 3.3.9 läßt sich die h-Transformation h *definieren durch*

$$h : ZR(e) \to QR(e) : \vec{b} \to \varphi^{-1}\left(\vec{b}\right).$$

Die Funktion h ordnet jedem Punkt des Zielraumes einer Abhängigkeit den eindeutig bestimmten Punkt des Quellraumes zu.

Kapitel 4

Spezifikation des Eingabeprogammes

In diesem Abschnitt wird eine genaue Spezifikation zulässiger Eingabeprogramme gegeben. Diese Restriktionen werden später in wesentlichen Teilen aufgehoben werden (vgl. Kapitel 7).

Ein Programm wird als zulässig betrachtet, wenn es die folgenden Kriterien erfüllt:

(1) Das Programm besteht ausschließlich aus *for*-Schleifen, Zuweisungen oder *skip*-Statements. Letztere dienen dazu, dem Benutzer im Rahmen des LooPo-Projektes eine einfache Möglichkeit zur freien Definition von Abhängigkeiten zu geben.

(2) Das Programm unterliegt einem statischen Kontrollfluß. Das bedeutet, daß die Schleifengrenzen nur aus affin linearen Funktionen der Indizes umgebender Schleifen sowie der Strukturparameter (Integer-Konstanten, die nur einmal definiert werden und deren jeweiliger Wert nicht durch das Eingabeprogramm bestimmt wird) bestehen. Insbesondere sind Arrays als Grenzen nicht zulässig. Die untere Grenze einer Schleife wird durch das Maximum solcher Funktionen dargestellt, die obere durch das Minimum.

(3) Der Gültigkeitsbereich eines Schleifenindexes ist auf die Schleife beschränkt. Das hat zur Folge, daß kein Index außerhalb seiner Schleife wiederverwendet werden darf.

$$
\begin{array}{ll}
\textbf{for}\ (\ i=1;\ i\leq 10;\quad i{+}{+}\ )\ \{ & \\
\quad \textbf{for}\ (\ j=i;\ j\leq 10;\quad j{+}{+}\ )\ \{ & \\
\quad\quad a[i,j] = a[i,j-1]; & \{\text{S1}\} \\
\quad \} & \\
\quad b[i] = a[i+1,10]; & \{\text{S2}\} \\
\} &
\end{array}
$$

Abbildung 4.1: Beispiel eines zulässigen Eingabeprogrammes

(4) Die Schrittweite jeder Schleife beträgt +1.

(5) Die Indexausdrücke bestehen aus Funktionen, für welche die unter (2) genannten Bedingungen gelten.

(6) Alle Strukturparameter werden als nicht negativ angenommen.[2]

Für die Notation der Eingabeprogamme in dieser Arbeit werden die Programmiersprachen Fortran und C gewählt.

Ein Beispiel für eine zulässige Eingabe gibt Abbildung 4.1. Hingegen widerspricht das Programm in Abbildung 4.2 in den Schleifen den Bedingungen 2 bzw. 4 und in den Statements den Bedingungen 3 bzw. 5.

[2]Diese Annahme ist strenggenommen für keinen der hier behandelten Ansätze erforderlich. Sie ist ein Zugeständnis an Feautriers PIP-Algorithmus und wird beibehalten, um die Äquivalenz der beiden Theorien in diesem Punkt zu sichern.

DO $i = 1,\ 10\ ,\ 2$
    DO $j = i^2,\ 10$
        $a[i,j] = a[i, i*j]$ {S1}
    END DO
    $b[i,j] = a[i+1, 10]$ {S2}
END DO

Abbildung 4.2: Beispiel eines unzulässigen Eingabeprogrammes

Kapitel 5

Datenabhängigkeitsanalyse nach Feautrier

5.1 Der ursprüngliche Ansatz

5.1.1 Die Idee

In [Fea91] stellt Feautrier ein Verfahren zur Berechnung aller Flußabhängigkeiten in einem den Bedingungen aus Kapitel 4 genügenden Eingabeprogramm vor.

Die gesamte Analyse läßt sich in zwei Phasen unterteilen. Während der ersten Phase werden zu einer lesenden Operation diejenigen Statements bestimmt, in welchen auf dasselbe Array schreibend zugegriffen wird. Für jedes dieser Statements wird eine Menge bestehend aus allen Iterationsvektoren von Operationen berechnet, die vor der gegebenen lesenden Operation ausgeführt werden und zusammen mit derselben zu einem Speicherkonflikt führen. Schließlich erfolgt für jede der zuvor bestimmten Mengen die Ermittlung desjenigen Vektors, der zu der zuletzt ausgeführten Operation des jeweiligen Statements gehört. In der zweiten Phase wird dann unter den auf diese Weise erhaltenen letzten Operationen eines jeden Statements diejenige berechnet, die unmittelbar vor der lesenden Operation ausgeführt wird. Genau diese Operation erzeugt den Datenfluß zur lesenden Operation und ist somit die Quelle der gesuchten Flußabhängigkeit.

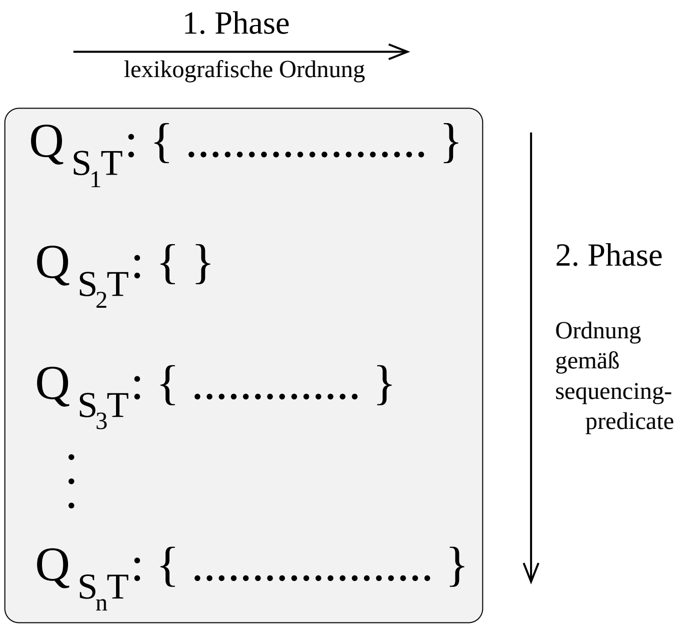


Abbildung 5.1: zweiphasige Berechnung

5.1.2 Formale Darstellung

Sei T Statement eines zulässigen Eingabeprogrammes mit lesendem Zugriff auf Array X, $\vec{b} \in D_T$ und $n \in \mathbb{N}$. Ferner seien $S_1, ..., S_n$ Statements mit schreibendem Zugriff auf X. Die zugehörigen Indexfunktionen werden mit g_T bzw. g_{S_i}, $1 \leq i \leq n$, bezeichnet. Schließlich gibt N_{S_iT} die Anzahl der S_i und T gemeinsam umgebenden Schleifen an.

1. Phase:

Zu jedem Statement S_i wird die Menge

$$Q_{S_iT}\left(\vec{b}\right) = \left\{\vec{v_i} \mid \underbrace{g_{S_i}(\vec{v_i}) = g_T\left(\vec{b}\right)}_{I}, \underbrace{<S_i, \vec{v_i}> \prec <T, \vec{b}>}_{II}, \underbrace{\vec{v_i} \in D_{S_i}}_{III}\right\} \tag{5.1.0.1}$$

berechnet.

Diese Mengen enthalten alle Iterationsvektoren (III) derjenigen Operationen des Statements S_i, welche vor $<T, \vec{b}>$ schreibend auf Array X zugreifen (II) und hierdurch einen Speicherkonflikt (I) verursachen.

Einerseits induziert nun die lexikographische $\lll$-Ordnung gemäß Bemerkung 3.2.5 eine Totalordnung auf Q_{S_iT}, andererseits ist Q_{S_iT} – Termination des

Eingabeprogramms voraussetzend – endlich. Folglich existiert genau dann ein Maximum bezüglich der lexikographischen $\ll$-Ordnung in der Menge Q_{S_iT}, wenn diese nicht leer ist.

Wegen der Eindeutigkeit des Maximums läßt sich eine Funktion K_{S_iT} definieren, die dem Iterationsvektor $\vec{b}$ der gegebenen Leseoperation das Maximum von Q_{S_iT} zuordnet. Zu beachten ist jedoch, daß Q_{S_iT} im allgemeinen auch leer sein kann. Für diesen Fall wird das Maximum dieser Menge als undefiniert (i. Z.: $\perp$) angenommen.

Es gilt:

$$K_{S_iT} \,:\, D_T \,\rightarrow\, Q_{S_iT} \cup \{\perp\} : \vec{b} \rightarrow \begin{cases} \max_{\ll} Q_{S_iT} & \text{falls } Q_{S_iT} \neq \emptyset \\ \perp & \text{falls } Q_{S_iT} = \emptyset \end{cases} \tag{5.1.0.2}$$

Phase 2:

Der zuletzt ausgeführte Schreibzugriff wird durch das Maximum aller Operationen $< S_i, K_{S_iT}(\vec{b}) >$ bezüglich der durch das Sequenzierungsprädikat gegebenen Ordnung definiert. Als Konsequenz des in der vorherigen Phase eingeführten $\perp$-Symbols wird für die undefinierte Operation, welche in gleicher Weise durch das Zeichen $\perp$ dargestellt wird, vereinbart, daß sie vor jeder definierten Operation ausgeführt wird. Durch diese Konvention bleibt die Linearität der durch das Sequenzierungsprädikat gegebenen Striktordnung (Bemerkung 3.2.5) erhalten. Für die Quelle S der Flußabhängigkeit gilt dann:

$$S\left(\vec{b}\right) = \max_{\prec}\{< S_i, K_{S_iT}(\vec{b}) > \mid 1 \leq i \leq n\} \tag{5.1.0.3}$$

S ist genau die in 3.3.20 definierte h-Transformation, durch welche die Flußabhängigkeit vollständig beschrieben wird.

5.1.3 Auswertungsstrategie

Das Vorgehen in der Auswertung von (5.1.0.3) basiert auf der Idee, die erste Phase in zwei Teile zu untergliedern. Der erste Teil stellt im weitesten Sinne eine Vorverarbeitung für die sich anschließenden Berechnungen dar. Der zweite Teil der ersten Phase kann dann gemeinsam mit Phase 2 in demselben Algorithmus berechnet werden.

Teil 1 sieht die Zerlegung der Mengen Q_{S_iT} in $N_{S_iT}+1$ disjunkte Teilmengen $Q^p_{S_iT}$ $(1 \leq p \leq N_{S_iT}+1)$ gemäß Definition 3.2.4 vor.

Es gilt:

$$Q^p_{S_iT}\left(\vec{b}\right) \;=\; \left\{\vec{v_i} \mid g_{S_i}\left(\vec{v_i}\right) = g_T\left(\vec{b}\right),\ <S_i,\vec{v_i}> \prec^p <T,\vec{b}>,\ \vec{v_i} \in D_{S_i}\right\} \tag{5.1.0.4}$$

Analog ergibt sich aus (5.1.0.2) durch Restriktion des Wertebereiches:

$$K^p_{S_iT} \;:\; D_T \;\rightarrow\; Q^p_{S_iT} \cup \{\perp\} : \vec{b} \rightarrow \max_{\ll} Q^p_{S_iT} \tag{5.1.0.5}$$

Teil 2:

Nachdem die Maxima aller Einzelmengen $Q^p_{S_iT}$ bestimmt worden sind, steht in der ersten Phase noch die Ermittlung der letzten Iteration eines Statements S_i, also die Berechnung des lexikografischen Maximums von $Q^p_{S_iT}$, $1 \leq i \leq n$, aus. Dieses läßt sich berechnen durch

$$K_{S_iT}\left(\vec{b}\right) = \max_{\ll} \{K^p_{S_iT} \mid 1 \leq p \leq N_{S_iT}\}. \tag{5.1.0.6}$$

Hiermit ergibt sich die folgende Ableitung für S:

$$\begin{aligned}
& S = \max_{\prec}\left\{<S_i, K_{S_iT}(\vec{b})> \;\middle|\; 1 \leq i \leq n\right\} \\
\Leftrightarrow \quad & \{\ \text{Definition von } K^p_{S_iT} \text{ (5.1.0.6)}\ \} \\
& S = \max_{\prec}\left\{<S_i, \max_{\ll}\{K^p_{S_iT} \mid 1 \leq p \leq N_{S_iT}\}> \;\middle|\; 1 \leq i \leq n\right\} \\
\Leftrightarrow \quad & \{\ \text{Wechsel der Ordnung gemäß Bemerkung 3.2.6}\ \} \\
& S = \max_{\prec}\left\{\ \max_{\prec}\{<S_i, K^p_{S_iT}> \mid 1 \leq p \leq N_{S_iT}\} \;\middle|\; 1 \leq i \leq n\right\} \\
\Leftrightarrow \quad & \{\ \text{Elimination des inneren } \max_{\prec}\text{-Operators}\ \} \\
& S = \max_{\prec}\left\{<S_i, K^p_{S_iT}(\vec{b})> \;\middle|\; 1 \leq p \leq N_{S_iT}+1,\ 1 \leq i \leq n\right\}
\end{aligned}$$

Als Ergebnis dieser Ableitung erhält man eine Formel für S, die nicht mehr von K_{S_iT}, sondern allein von $K^p_{S_iT}$ abhängt. Das bedeutet nun, daß auf die explizite Berechnung des zweiten Teils von Phase eins nach (5.1.0.6) verzichtet werden kann.

Um mehrfache Indizierung zu vermeiden, erfolgt hier eine bei 1 beginnende und bis L fortlaufende Numerierung der Elemente $<S_i, K^p_{S_iT}(\vec{b})>$, die jetzt mit $e_1, e_2, \ldots., e_L$ bezeichnet werden.

$$\begin{array}{rcl}
\text{quast} & = & \perp \\
 & | & \text{vector} \\
 & | & \texttt{if}\ \text{form} \geq 0\ \texttt{then}\ \text{quast}\ \texttt{else}\ \text{quast} \\
\text{form} & = & \text{integer} \\
 & | & \text{structure-parameter} \\
 & | & \text{integer} * \text{form} \\
 & | & \text{integer} \div \text{form} \\
 & | & \text{form} + \text{form} \\
\text{vector} & = & (\text{form}[,\ \text{form}])
\end{array}$$

Abbildung 5.2: BNF des Konstruktes *quast*

Sind alle Mengen $Q^p_{S_iT}$ leer, so hat das zur Folge, daß sich die undefinierte Operation als Lösung für S berechnet. Dieser Fall dient gleichsam als Basis für die rekursive Auswertung von S:

$$\begin{aligned} S_0 &= \perp \\ S_k &= \max_{\prec} \{S_{k-1}, e_k\} \qquad 0 < k \leq L \end{aligned} \tag{5.1.0.7}$$

Um diese Rekursion berechnen zu können, wird letztlich noch ein Konstrukt eingeführt, das es ermöglicht, die Ergebnisse aufeinanderfolgender Maximierungen sowie deren Gültigkeitsbedingungen darzustellen. Dabei handelt es sich um den sogenannten *quasi affin selection tree*, kurz: Quast. Abbildung 5.2 zeigt die zugehörige BNF.

Zu beachten ist hierbei, daß $\div$ die Integer-Division darstellt, die innerhalb eines Quasts eine höhere Priorität als die Multiplikation besitzt.

Wird nun die Menge S_k, $0 \leq k \leq L$, durch einen Quast $T(S_k)$ repräsentiert, so ergibt sich aus (5.1.0.7)

$$\begin{aligned} T(S_0) &= \perp \\ T(S_k) &= \max_{\prec} \{T(S_{k-1}), e_k\} \qquad 0 < k \leq L \end{aligned} \tag{5.1.0.8}$$

und die gesuchte Lösung für S wird durch $T(S_L)$ dargestellt.

Hierbei erfolgt die Auswertung von $T(S_k)$, $0 < k \leq L$, gemäß den folgenden drei Regeln:

1.: $T(S_{k-1}) = \perp$

$$T(S_k) = e_k$$

2.: $T(S_{k-1}) = <r, f(\vec{b})>$ (r Statement, f form vgl. Abbildung 5.2)

$$\begin{array}{rl} T(S_k) = & \texttt{if } <r, f(\vec{b})> \prec e_k \texttt{ then} \\ & \quad e_k \\ & \texttt{else} \\ & \quad <r, f(\vec{b})> \end{array}$$

3.: $T(S_{k-1}) = \texttt{if } \pi_{k-1}(\vec{b}) \texttt{ then } T_{if} \texttt{ else } T_{else}$ (π_{k-1} Bedingung in Abhängigkeit von $\vec{b}$.)

$$\begin{array}{rl} T(S_k) = & \texttt{if } \pi_{k-1}(\vec{b}) \texttt{ then} \\ & \quad \max_{\prec}\{T_{if}, e_k\} \\ & \texttt{else} \\ & \quad \max_{\prec}\{T_{else}, e_k\} \end{array}$$

5.1.4 Beispiel

Gegeben sei das Programm aus Abbildung 5.3.

Aus der Eingabe ist zu ersehen, daß $N_{S1S2} = 0$ und $N_{S2S2} = 2$ gilt. Damit ergeben sich (mit den Bezeichnungen aus (5.1.0.1) sowie (5.1.0.4)) die folgenden Mengen:

$$\begin{aligned} Q_{S1S2}(i,j) = Q^1_{S1S2} &= \{k \mid \underbrace{k = i+j}_{(I)},\ \underbrace{\overbrace{S1 <_{txt} S2}^{TRUE}}_{(II)},\ \underbrace{0 \le k \le 10}_{(III)}\} \\ &= \{k \mid k = i+j,\ 0 \le k \le 10\} \end{aligned}$$

$$Q_{S2S2}(i,j) = \dot{\bigcup}\ p\ :\ 1 \le p \le N_{S2\,S2+1}\ :\ Q^p_{S1S2}$$

mit

$$\begin{aligned} Q^1_{S2S2}(i,j) &= \{(i',j') \mid \underbrace{i'+j' = i+j}_{(I)},\ \underbrace{i' < i \vee \overbrace{(i' = i \wedge j' < j)}^{FALSE}}_{(II)},\ \underbrace{0 \le i', j' \le 10}_{(III)}\} \\ &= \{(i',j') \mid i'+j' = i+j,\ i' < i,\ 0 \le i \le 10,\ 0 \le j \le 10\} \end{aligned}$$

$$Q^2_{S2S2}(i,j) = Q^3_{S2S2}(i,j) = \emptyset \qquad \text{(wegen } (II)\text{)}.$$

$$
\begin{array}{ll}
\textbf{for}\ (\ k=0;\ k \leq 10;\quad k{+}{+}\ )\ \{ & \\
\quad c[k] = 0; & \{\text{S1}\} \\
\} & \\
\textbf{for}\ (\ i=0;\ i \leq 10;\quad i{+}{+}\ )\ \{ & \\
\quad \textbf{for}\ (\ j=0;\ j \leq 10;\quad j{+}{+}\ )\ \{ & \\
\quad\quad c[i+j] = c[i+j] + a[i] * b[j]; & \{\text{S2}\} \\
\quad \} & \\
\} &
\end{array}
$$

Abbildung 5.3: Beispielprogramm zum ursprünglichen Ansatz

Wendet man (5.1.0.5) hierauf an, so folgt:

$$
K^1_{S1S2}(i,j) = \begin{cases} (i+j) & \text{falls } 0 \leq i+j \leq 10 \\ \perp & \text{falls } 10 < i+j \end{cases}
$$

$$
K^1_{S2S2}(i,j) = \begin{cases} (i-1, j+1) & \text{falls } 1 \leq i \wedge j \leq 9 \\ \perp & \text{falls } i = 0 \vee j = 10 \end{cases}
$$

$$
K^2_{S2S2}(i,j) = K^3_{S2S2}(i,j) = \perp
$$

Aus der obigen Ableitung für S ergibt sich wegen $K^2_{S2S2} = K^3_{S2S2}$

$$
S = \max\nolimits_{\prec}\{<S1, K^1_{S1S2}>, <S2, K^1_{S2S2}>, <S2, K^2_{S2S2}>\}.
$$

Die Auswertung gemäß (5.1.0.8) liefert:

$$
\begin{aligned}
S_0 &= \perp \\
S_1 &= <S1, K^1_{S1S2}> \\
S_2 &= \texttt{if} \quad <S1, K^1_{S1S2}> \prec <S2, K^1_{S2S2}> \texttt{ then} \\
&\qquad <S2, K^1_{S2S2}> \\
&\quad \texttt{else} \\
&\qquad <S1, K^1_{S1S2}> \\
S_3 &= \texttt{if} \quad <S1, K^1_{S1S2}> \prec <S2, K^1_{S2S2}> \texttt{ then} \\
&\qquad \max_\prec\{<S2, K^1_{S2S2}>, <S2, K^2_{S2S2}>\} \\
&\quad \texttt{else} \\
&\qquad \max_\prec\{<S1, K^1_{S1S2}>, <S2, K^2_{S2S2}>\} \\
&= \texttt{if} \quad <S1, K^1_{S1S2}> \prec <S2, K^1_{S2S2}> \texttt{ then} \\
&\qquad \texttt{if } <S2, K^1_{S2S2}> \prec <S2, K^2_{S2S2}> \texttt{ then} \\
&\qquad\quad <S2, K^2_{S2S2}> \\
&\qquad \texttt{else} \\
&\qquad\quad <S2, K^1_{S2S2}> \\
&\quad \texttt{else} \\
&\qquad \texttt{if } <S1, K^1_{S1S2}> \prec <S2, K^2_{S2S2}> \texttt{ then} \\
&\qquad\quad <S2, K^2_{S2S2}> \\
&\qquad \texttt{else} \\
&\qquad\quad <S1, K^1_{S1S2}>
\end{aligned}
$$

Durch Einsetzen und anschließende Vereinfachung des Quasts erhält man:

$$
\begin{aligned}
S = \texttt{if} \ & (1 < i \wedge j < 9) \texttt{ then} \\
& \quad <S2, (i-1, j+1)> \\
& \texttt{else} \\
& \quad \texttt{if} \ (i + j \leq 10) \texttt{ then} \\
& \qquad <S1, (i+j)> \\
& \quad \texttt{else} \\
& \qquad \perp
\end{aligned}
$$

Abbildung 5.4 veranschaulicht dieses Ergebnis.

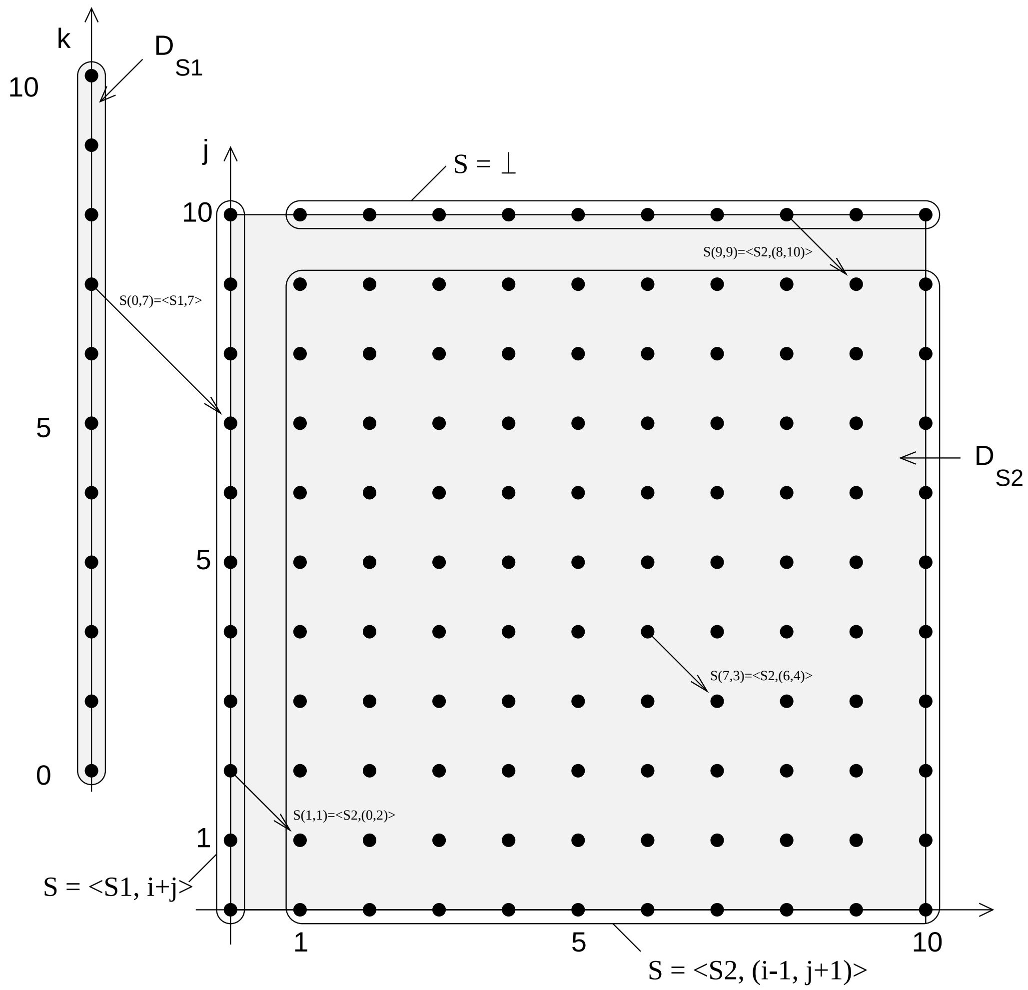


Abbildung 5.4: Abhängigkeiten des Programms in Abbildung 5.3

5.2 Die Erweiterung des Ansatzes

5.2.1 Anti- und Output-Abhängigkeiten

Nachdem die Flußabhängigkeiten des Eingabeprogrammes bestimmbar sind, soll der Algorithmus in diesem Abschnitt um die Berechnung von anti- und output-Abhängigkeiten erweitert werden. Ziel hierbei wird sein, auch weiterhin die Direktheit aller Abhängigkeiten zu garantieren.

5.2.1.1 Möglichkeit 1

Bedenkt man das Verfahren zur Berechnung einer Flußabhängigkeit, bei dem zu einer lesenden Operation die zuletzt ausgeführte schreibende gesucht wird, so ist es naheliegend, bei der nun anstehenden Ermittlung von anti- und output-Abhängigkeiten analog vorzugehen. Man bestimmt zu einer Schreiboperation die letzte aller vorangegangenen lesenden (bzw. schreibenden) Operation, um auf diese Weise alle anti- (bzw. output-)Abhängigkeiten zu erhalten.

Jedoch ist dieses Vorgehen bezüglich der oben genannten Zielsetzung, die Direktheit der Abhangigkeiten zu garantieren, falsch. Wie aus dem einfachen Beispielprogramm in Abbildung 5.5 zu ersehen ist, werden auf diese Weise die direkten Abhängigkeiten $S1\delta S2$ und $S2\delta S3$ ebenso wie die indirekte Abhängigkeit $S1\delta S3$ ermittelt.

Erforderlich ist an dieser Stelle eine Nachbearbeitungsphase, in der alle indirekten Abhängigkeiten eliminiert werden.

Eine solche Elimination gestaltet sich jedoch nicht so einfach, wie das Beispiel aus Abbildung 5.5 vermuten läßt. Das veranschaulicht das Programm aus Abbildung 5.6.

$a[0] = 0;$ {S1}
$b[0] = a[0];$ {S2}
$a[0] = 1;$ {S3}

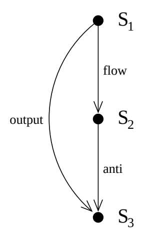


Abbildung 5.5: Gegenbeispiel 1 mit Abhängigkeitsgraphen

```
for ( i = 1;  i ≤ 4;  i++ ) {
    a[i] = ...                {S1}
}
for ( j = 1;  i ≤ 2;  j++ ) {
    a[i] = ...                {S2}
}
for ( k = 1;  i ≤ 4;  k++ ) {
    a[i] = ...                {S3}
}
```

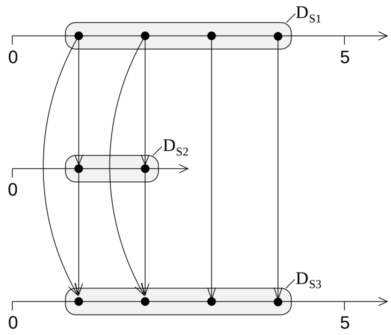


Abbildung 5.6: Beispiel zur Elimination indirekter Abhängigkeiten

Durch einen Teil der Operationen von $S1$ und $S3$ werden indirekte Abhängigkeiten erklärt $(1 \leq i, k \leq 2)$. Für die übrigen Operationen $(1 \leq i, k \leq 2)$ sind die Abhängigkeiten jedoch direkt. Folglich betrifft die Löschung indirekter Abhängigkeiten nur eine Teilmenge der Iterationsräume von $S1$ und $S2$. An dieser Stelle soll jedoch nicht weiter auf diese Problematik eingegangen werden. Ein Verfahren zur vollständigen Elimination indirekter Abhängigkeiten findet sich in [PW92a, PW92b].

Es gibt jedoch eine andere Methode, mit der die Berechnung indirekter Abhängigkeiten von vorneherein ausgeschlossen wird.

5.2.1.2 Möglichkeit 2

5.2.1.2.1 Die Idee

Der Grundgedanke dieses Ansatzes ist es, zunächst nicht zwischen anti- und output-Abhängigkeiten zu unterscheiden. Erst zum spätestmöglichen Zeitpunkt wird die Art der Abhängigkeit festgelegt.

Zu einem gegebenen Schreibzugiff wird der letzte aller vorangegangenen Zugiffe ohne Berücksichtigung des Typs (lesend/schreibend) ermittelt. Ist dieser Zugriff gefunden, ergibt sich aus seinem Typ auch die Art der durch beide Zugriffe definierten Abhängigkeit.

5.2.1.2.2 Wechsel von statement- zu zugriffsbasierter Sicht

Die Umsetzung dieser Methode erfordert eine Verfeinerung der gesamten Sichtweise der Analyse. Wurden bislang vorangehende Zugriffe ermittelt, so genügte die Betrachtung der Statements, weil es je Statement genau einen schreibenden Zugriff gibt. Da jetzt auch lesende Zugriffe betrachtet werden, muß von einer statement-basierten Betrachtungsweise zu einer auf Zugriffen beruhenden Sicht übergegangen werden.

Die Menge $\mathcal{Z}_{S,X}$ bezeichnet die Zugriffe auf ein Array X innerhalb eines Statements S. Sie zefällt in die (nicht notwendigerweise von $\emptyset$ verschiedene) Menge der lesenden Zugriffe $\mathcal{Z}^r_{S,X}$ und die aus dem Schreibzugriff bestehende Menge $\mathcal{Z}^w_{S,X}$:

$$\begin{aligned} \mathcal{Z}_{S,X} &= \mathcal{Z}^r_{S,X} \,\dot{\bigcup}\, \mathcal{Z}^w_{S,X} \\ &\qquad \text{mit } \mathcal{Z}^r_{S,X} = \{z \mid z \text{ ist lesender Zugriff auf } X \text{ in } S\} \\ &\qquad\qquad \mathcal{Z}^w_{S,X} = \{z \mid z \text{ ist schreibender Zugriff auf } X \text{ in } S\} \end{aligned}$$

Allerdings kommt nicht jedem Zugriff die gleiche Bedeutung zu. Gleichen sich zwei oder mehr Lesezugriffe innerhalb eines Statements, das heißt

- greifen sie auf dasselbe Array zu,
- sind vom gleichen Typ (lesend/schreibend) und
- besitzen dieselbe Indexfunktion,

so überlegt man sich leicht, daß nur jeweils einer dieser Zugriffe einen Beitrag liefert, während die Berücksichtigung der übrigen lediglich einen deutlich erhöhten Rechenaufwand mit sich bringt. Ein Beispiel hierzu stellen die beiden Lesezugriffe auf D im Beispielprogramm in Abbildung 5.8 dar. Wünschenswert wäre es also, nur den Repräsentanten einer Gruppe gleicher Zugriffe zu betrachten.

Sei für ein Statement S und ein darin referenziertes Array X die Menge $\mathcal{Z}_{S,X}$ aller Zugriffe z in S auf X, sowie deren Zerlegung $\mathcal{Z}^r_{S,X}$, $\mathcal{Z}^w_{S,X}$ gegeben. Die zugehörigen Indexfunktionen werden wie üblich mit g_z bezeichnet. Ferner sei G die Funktion, die einem Zugriff seine Indexfunktion zuordnet:

$$G(z) = g_z \qquad z \in \mathcal{Z}_{S,X}$$

Dann wird durch G eine Äquivalenzrelation sowohl auf $\mathcal{Z}^r_{S,X}$ als auch auf $\mathcal{Z}^w_{S,X}$ definiert durch:

$$z_1 \sim_G z_2 \iff G(z_1) = G(z_2) \qquad z_1, z_2 \in \mathcal{Z}^t_{S,X},\ t \in \{r,\, w\}$$

Man erhält auf diese Weise eine Zerlegung von $\mathcal{Z}^r_{S,X}$ (bzw. $\mathcal{Z}^w_{S,X}$) in $M^r_{S,X}$ (bzw. $M^w_{S,X}$) Äquivalenzklassen:

$$\mathcal{Z}^t_{S,X} = \dot{\bigcup}\ i\ :\ 1 \leq i \leq M^t_{S,X}\ :\ \left[Z^{t\,(i)}_{S,X}\right] \qquad t \in \{r,\, w\}$$

Hierbei stellen $Z^{r\,(i)}_{S,X}$ (bzw. $Z^{w\,(i)}_{S,X}$), $1 \leq i \leq M^r_{S,X}$ (bzw. $M^w_{S,X}$), genau die gesuchten Repräsentanten gleicher Lese- (bzw. Schreib-)Zugriffe dar. (Auf die Bedeutung für schreibende Zugriffe wird ausführlich in Kapitel 7 eingegangen).

Infolge der so gefundenen Zerlegung von $\mathcal{Z}_{S,X}$ ist es nun möglich, jeden Zugriff auf X in S, der auch einen Beitrag liefert, durch Angabe des Zugriffstyps und der zugehörigen Indexfunktion zu identifizieren und auf diese Weise die Menge $Z_{S,X}$ aller Repräsentanten wie folgt zu definieren:

$$Z_{S,X} = \left\{(t, G(z)) \mid z \in Z^{t\,(i)}_{S,X},\ 1 \leq i \leq M^t_{S,X},\ t \in \{r,\, w\}\right\}$$

5.2.1.2.3 Formale Darstellung

Auf der Grundlage dieses Ergebnisses kann Feautriers Analyse in nahezu unveränderter Form durchgeführt werden. Der einzige Unterschied ergibt sich aus der Tatsache, daß nun zu einem schreibenden Zugriff auf ein Array jeder einzelne Zugriff auf dasselbe Array betrachtet wird.

Gegeben sei ein Schreibzugriff auf ein Array X innerhalb eines Statements T. Seien – bei sonst gleichbleibenden Bezeichnungen – $S_1, \ldots, S_n$ diejenigen Statements des Eingabeprogrammes, in denen schreibend oder lesend auf X zugegriffen wird. Statt wie bisher für jedes Statement S_i die Menge Q_{S_iT} zu bestimmen (vgl. (5.1.0.1)), werden nun in gleicher Weise für jeden Zugriff in S_i die Mengen $Q_{(S_i,z)T}$, $z \in Z_{S_i,X}$, gebildet. In völliger Analogie zu (5.1.0.4), (5.1.0.2) und (5.1.0.5) ergeben sich dann die Mengen $Q^p_{(S_i,z)T}$, sowie die Funktionen $K_{(S_i,z)T}$ und $K^p_{(S_i,z)T}$.

Auch alle sich anschließenden Überlegungen in Abschnitt 5.1.3 behalten ihre volle Gültigkeit bei.

$$a[0] = b[0]; \qquad \{\mathrm{S1}\}$$
$$a[0] = c[0] * a[0]; \qquad \{\mathrm{S2}\}$$

Abbildung 5.7: Gegenbeispiel 2

Es folgt:

$$S = \max\nolimits_{\prec} \left\{ < S_i, K^p_{(S_i,z)T}(\vec{b}) > \;\middle|\; 1 \le p \le N_{S_i T} + 1, 1 \le i \le n, z \in Z_{S_i,X} \right\}$$

Nach Umnumerierung der Elememte bleibt letztlich auch die rekursive Auswertung nach (5.1.0.7) bzw. (5.1.0.8) unverändert.

5.2.1.2.4 Zugriffsbasierte Ordnungen, statementgetragene anti-Abhängigkeiten

Durch das im vorherigen Abschnitt vorgestellte Verfahren können nun auch die für die Parallelisierung erforderlichen anti- bzw. output-Abhängigkeiten berechnet werden. Jedoch garantiert dieses Verfahren derzeit in einem Punkt noch keine Direktheit. Ein einfaches Beispiel hierzu liefert Abbildung 5.7. Zu der direkten true-Abbhängigkeit $S1\delta S2$ wird ebenfalls die indirekte output-Abhängigkeit $S1\delta S2$ berechnet.

Der Grund hierfür ist, daß das zugriffsbasierte Vorgehen noch keine Berücksichtigung in der textuellen Ordnung und damit auch nicht in der durch das Sequenzierungsprädikat gegebenen Ordnung gefunden hat. Als unmittelbare Folge hiervon kann die direkte, statementgetragene anti-Abhängigkeit $S2\delta S2$, welche zusammen mit der Flußabhängigkeit eine transitive Darstellung der output-Abhängigkeit ergibt, nicht berechnet werden.

Um nun auch dieses Problem zu lösen und damit wieder die Direktheit aller Abhängigkeiten garantieren zu können, ist also eine Verfeinerung des Sequenzierungsprädikat bzw. der textuellen Ordnung unter der besonderen Berücksichtigung des zugriffsbasierten Vorgehens erforderlich.

Die Änderung der bestehenden textuellen Ordnung hat zum Ziel, alle lesenden Zugriffe eines Statements als textuell vor dem Schreibzugriff desselben Statements erkennen zu können.

Als intuitive Lösung läge es nahe, eine Funktion *is_read* zu definieren, die entscheidet, ob ein Zugriff lesend ist:

$$is\_read : Z_{S,X} \rightarrow \mathbb{B}_2 : z \rightarrow \delta_{\pi_1(z),r}$$

Sei $z_i \in Z_{S_i,X}$ und $v_i \in D_{S_i}$ für ein $i \in \{1, 2\}$. Dann ergäbe sich für die zugriffsbasierte textuelle Ordnung (i. Z.: $<_{acc}$):

$$z_1 <_{acc} z_2 \iff (S_1 <_{txt} S_2) \vee (S_1 = S_2 \wedge is\_read(z_1))$$

Man sieht jedoch leicht, daß diese Relation insofern nicht verträglich mit der bisherigen ist, als durch sie keine lineare Striktordnung (vgl. Bemerkung 3.2.5) definiert wird. Es gilt nämlich für zwei lesende Zugriffe z_1 und z_2 eines Statements sowohl $z_1 <_{acc} z_2$ als auch $z_2 <_{acc} z_1$. Aus der Transitivität folgt dann $z_1 <_{acc} z_1$, welches zum Widerspruch zur Irreflexivität steht.

Aus diesem Grund ist eine Verfeinerung der textuellen Ordnung mit Hilfe der Funktion *is_read* abzulehnen.

Stattdessen wird nun jedem Zugriff $z \in Z_{S,X}$ eine eindeutige Nummer in der Weise zugeordnet, daß der schreibende Zugriff die höchste Nummer erhält. Definiere

$$\begin{aligned} &\psi : Z_{S,X} \rightarrow \left\{1, ..., M^r_{S,X} + M^w_{S,X}\right\} \qquad (5.2.0.9)\\ \text{mit} \quad &\psi \text{ Bijektion} \wedge (\forall\, z_1, z_2 : z^r \in Z^r_{S,X},\, z^w \in Z^w_{S,X} : \psi(z_1) < \psi(z_2)). \end{aligned}$$

Jetzt überträgt sich die lineare Striktheit der <-Relation auf die verfeinerte textuelle Ordnung. Mit den zuvor eingeführten Bezeichnungen gilt:

$$z_1 <_{acc} z_2 \iff (S_1 <_{txt} S_2) \vee (S_1 = S_2 \wedge \psi(z_1) < \psi(z_2)) \qquad (5.2.0.10)$$

Hierauf aufbauend stellt sich die durch das zugriffsbasierte Sequenzierungsprädikat (i. Z.:$\prec_{acc}$) definierte Ordnung als Disjunktion $N_{S_1S_2} + 2$ linearer Prädikate dar:

$$\begin{aligned} &< S_1, v_1, z_1 > \prec_{acc} < S_2,\, v_2,\, z_2 > \\ &\iff \qquad (5.2.0.11)\\ &\quad (v_1 \ll v_2)\\ &\vee \quad ((v_1 = v_2) \wedge (S_1 <_{txt} S_2))\\ &\vee \quad ((v_1 = v_2) \wedge (S_1 = S_2) \wedge \psi(z_1) < \psi(z_2)) \end{aligned}$$

Um nun das zu Beginn dieses Abschnittes aufgezeigte Problem zu lösen und damit die Direktheit aller Abhängigkeiten zu garantieren, wird nun in 5.2.1.2.3 jede statementbasierte Ordnung durch ihre Verfeinerung unter Berücksichtigung des neu hinzugekommenen Prädikates ersetzt.

Man erhält:

$$S = \max_{\prec_{acc}} \left\{ < S_i, K^p_{(S_i,z)T}(\vec{b}) > \,\middle|\, 1 \leq p \leq N_{S_i T} + 2, 1 \leq i \leq n, \, z \in Z_{S_i,X} \right\}$$

Auch die Auswertung von S vollzieht sich nach Substitution des Sequenzierungsprädikats durch seine Verfeinerung in (5.1.0.7), (5.1.0.8) sowie den Rekursionsregeln 2 und 3 auf Seite 33 in gleicher Weise.

5.2.1.2.5 Bemerkung

Durch die im vorherigen Abschnitt eingeführte Numerierung der Zugriffe wird eine Reihenfolge derselben zum Ausdruck gebracht. Weil aber die Numerierung lediglich eindeutig sein und dem Schreibzugiff die höchste Nummer zuordnen muß, kann sie weitgehend frei gewählt werden. Somit ist die durch sie ausgedrückte Reihenfolge willkürlich und spiegelt im allgemeinen nicht die real vom Rechner gewählte Folge von Zugriffen wieder. Vielmehr dient die Numerierung hier als ein technisches Hilfsmittel zur rekursiven Auswertung der Quelle.

5.2.2 Beispiel

Anhand des Eingabeprogrammes aus Abbildung 5.8 wird die Berechnung der anti- und output-Abhängigkeiten vorgestellt. Hingegen wird auf Flußabhängigkeiten, die in gewohnter Weise ermittelt werden, nicht mehr näher eingegangen. Ein Darstellung des Ergebnisses findet sich in Abbildung 5.9(a).

Aus dem Eingabeprogramm ergibt sich $N_{S1S1} = 2$ und

$$\begin{aligned} Z_{S1,D} \; = \; & Z^r_{S1,D} \dot{\bigcup} Z^w_{S1,D} \\ & \text{mit } Z^r_{S1,D} = \{(r, i-j+2)\}\,, \\ & \phantom{\text{mit }} Z^w_{S1,D} = \{(w, i+j)\}\,. \end{aligned}$$

Mit der Definition von ψ wie in (5.2.0.9) gilt dann:

$$\begin{aligned} \psi\left((r, i-j+2)\right) &= 1, \\ \psi\left((w, i+j)\right) &= 2. \end{aligned}$$

```
for ( i=1; i≤5;  i++ ) {
    for ( j=1; j≤5;  j++ ) {
        D[i+j] = D[i−j+2] * D[i−j+2];                {S1}
    }
}
```

Abbildung 5.8: Beispielprogramm zum erweiterten Ansatz

Entsprechend (5.1.0.1) ergeben sich nun für einzelne Zugriffe als mögliche Quellen die Mengen

$$\begin{aligned} Q_{(S1,(r,i-j+2))S1} &= \dot{\bigcup}\ p\ :\ 1 \leq p \leq 4\ :\ Q^p_{(S1,(r,i-j+2))S1}, \\ Q_{(S1,(w,i+j))S1} &= \dot{\bigcup}\ p\ :\ 1 \leq p \leq 4\ :\ Q^p_{(S1,(w,i+j))S1}. \end{aligned}$$

Zum Zweck der notationellen Vereinfachung wird im folgenden der lesende Zugriff $(r, i-j+2)$ durch z_r und der schreibende $(w, i+j)$ durch z_w abgekürzt. Durch Anwendung von (5.2.0.11) erhält man

$$\begin{aligned} Q^1_{(S1,z_r)S1} &= \{(i',j') \mid i'-j'+2 = i+j,\ (i',j') \in D_{S1},\ i' < i\} \\ Q^2_{(S1,z_r)S1} &= \{(i',j') \mid i'-j'+2 = i+j,\ (i',j') \in D_{S1},\ i' = i,\ j' < j\} \\ Q^3_{(S1,z_r)S1} &= \{(i',j') \mid i'-j'+2 = i+j,\ (i',j') \in D_{S1},\ (i'j') = (i,j), \\ &\qquad S1 <_{txt} S1\} \\ Q^4_{(S1,z_r)S1} &= \{(i',j') \mid i'-j'+2 = i+j,\ (i',j') \in D_{S1},\ (i'j') = (i,j), \\ &\qquad S1 = S1,\ \psi(z_r) < \psi(z_w)\} \end{aligned}$$

Mach Auswertung der einzelnen Bedingungen wird ersichtlich, daß die Mengen $Q^i_{(S1,z_r)S1}$, $i \in \{1,2,3\}$, leer sind.

Die Bestimmung der Extrema bezüglich der lexikografischen Ordnung nach (5.1.0.5) liefert

$$\begin{aligned} K^1_{(S1,z_r)S1} &= K^2_{(S1,z_r)S1} = K^3_{(S1,z_r)S1} = \perp \\ K^4_{(S1,z_r)S1} &= \begin{cases} (i,j) & \text{falls } j = 1 \\ \perp & \text{falls } j \geq 2 \end{cases}. \end{aligned}$$

Für den schreibenden Zugriff z_w verfährt man analog und erhält

$$\begin{aligned} Q^1_{(S1,z_w)\,S1} &= \{(i',j') \mid i'+j' = i+j,\ (i',j') \in D_{S1},\ i' < i\} \\ Q^2_{(S1,z_w)\,S1} &= Q^3_{(S1,z_w)\,S1} = Q^4_{(S1,z_w)\,S1} = \emptyset, \end{aligned}$$

sowie

$$\begin{aligned} K^1_{(S1,z_w)S1} &= \begin{cases} (i-1,j+1) & \text{falls } (2 \le i) \wedge (j \le 4) \\ \perp & \text{falls } (i=1) \vee (j=5) \end{cases} \\ K^2_{(S1,z_w)S1} &= K^3_{(S1,z_w)S1} = K^4_{(S1,z_w)S1} = \perp. \end{aligned}$$

Aus der Verfeinerung von S in (5.1.0.3) folgt dann

$$S = \max_{\prec_{acc}}\{<S1, K^4_{(S1,z_r)S1}>, <S1, K^1_{(S1,z_w)S1}>, <S1, K^1_{(S1,z_r)S1}>\}.$$

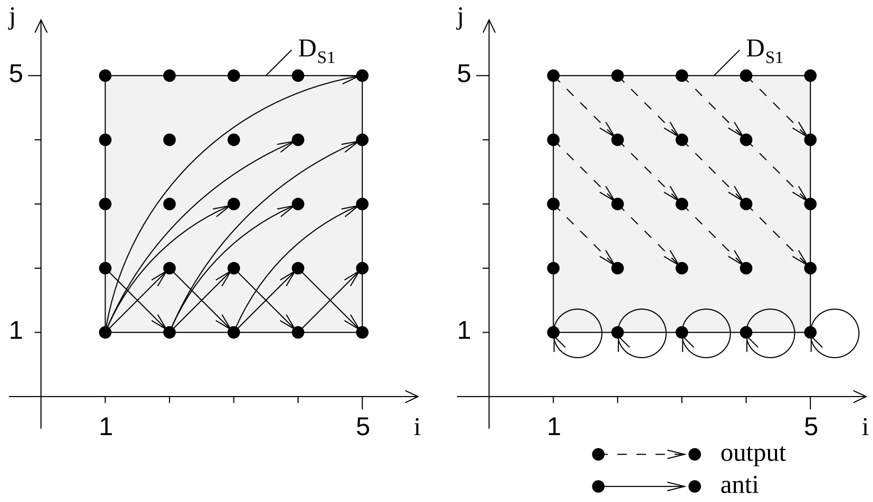
Die rekursive Bestimmung der Quelle liefert

$$\begin{aligned} S_3 = \texttt{if}\ \ & <S1, K^4_{(S1,z_r)S1}> \prec_{acc} <S1, K^1_{(S1,r_w)S1}> \texttt{ then} \\ & \quad \texttt{if } <S1, K^1_{(S1,z_w)S1}> \prec_{acc} <S1, K^1_{(S1,z_r)S1}> \texttt{ then} \\ & \quad\quad <S1, K^1_{(S1,z_r)S1}> \\ & \quad \texttt{else} \\ & \quad\quad <S1, K^1_{(S1,z_w)S1}> \\ & \texttt{else} \\ & \quad \texttt{if } <S1, K^4_{(S1,z_r)S1}> \prec_{acc} <S1, K^1_{(S1,z_r)S1}> \texttt{ then} \\ & \quad\quad <S1, K^1_{(S1,z_r)S1}> \\ & \quad \texttt{else} \\ & \quad\quad <S1, K^4_{(S1,z_r)S1}>. \end{aligned}$$

Und schließlich ergibt sich durch Einsetzen und Vereinfachung als Endergebnis

$$\begin{aligned} S = \texttt{if}\ \ & (2 \le i \le 5) \wedge (2 \le j \le 4) \texttt{ then} \\ & \quad <S1, (i-1, j+1)> \\ & \texttt{else} \\ & \quad \texttt{if } (j=1) \texttt{ then} \\ & \quad\quad <S1, (i,j)> \\ & \quad \texttt{else} \\ & \quad\quad \perp \end{aligned}$$

Abbildung 5.9(b) veranschaulicht diese Ergebnisse. Vergleicht man die Abbildungen 5.9(a) und 5.9(b), so wird die transitive Darstellung der indirekten output-Abhängigkeit $< S1, (i, 2) > \delta < S1, (i{+}1, 1) >$ durch die in gleicher Weise definierte Flußabhängigkeit und die statementgetragene anti-Abhängigkeit $< S1, (i + 1, 1) > \delta < S1, (i + 1, 1) >$, $1 \leq i \leq 4$, deutlich.



(a) Flußabhängigkeiten (b) anti/output-Abhängigkeiten

Abbildung 5.9: direkte Abhängigkeiten

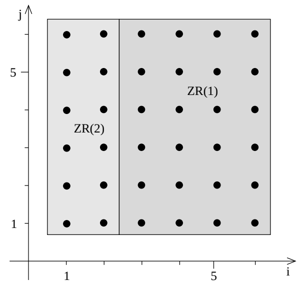
5.2.3 Aufspaltung der h-Transformation

Bisher wird die Quelle einer Abhängigkeit stets rekursiv ermittelt und mit Hilfe eines Quasts dargestellt. Die h-Transformation wird also durch diese Struktur repräsentiert. Jedoch wird sich im folgenden Kapitel herausstellen, daß bei Banerjees Ansatz eine solche verzweigte Darstellung nicht möglich ist. Um einen Vergleich der Ansätze zu ermöglichen, wird in diesem Abschnitt auf die Zerlegung eines Quasts eingegangen. Ziel ist es, eine aus den Blättern dieses Quasts bestehende Menge von h-Transformationen zu erzeugen.

Die grundlegende Idee besteht darin, die im Quast auf dem Weg von der Wurzel zu einem Blatt enthaltenen Bedingungen dem Zielraum hinzuzufügen. Prinzipiell ist das möglich, weil diese Bedingungen nach Voraussetzung (vgl. Abbildung 5.2) affin lineare Funktionen in den Schleifenvariablen und Strukturparametern sind und somit die gleiche Form besitzen, wie die Grenzen der Iterationsräume (vgl. 4(2)) besitzen.

Beispielsweise ergibt sich für einen Quast der Form

if $(i \geq 3)$ **then**
$< S, \, function\_1 >$
else
$< T, \, function\_2 >$



in einem Programm mit $[1, 6] \times [1, 6]$ als Iterationsraum des Statements, das Ziel der Abhängigkeit ist, die folgende Zerlegung:

Abhängigkeit	Zielraum	Quelle	h-Transformation
1	$3 \leq i \leq 6$ $1 \leq j \leq 6$	S	$function\_1$
2	$1 \leq i \leq 2$ $1 \leq j \leq 6$	T	$function\_2$

Bei Quasts mit größerer Verschachtelungstiefe ergeben sich durch dieses Verfahren zu komplexeren Ausdrücke in der Form von Konjunktionen einzelner Bedingungen.

Für Konjunktionen wird in diesem Zusammenhang angenommen, daß sie aus einer minimalen Zahl von Klauseln (mindestens jedoch zwei) bestehen.

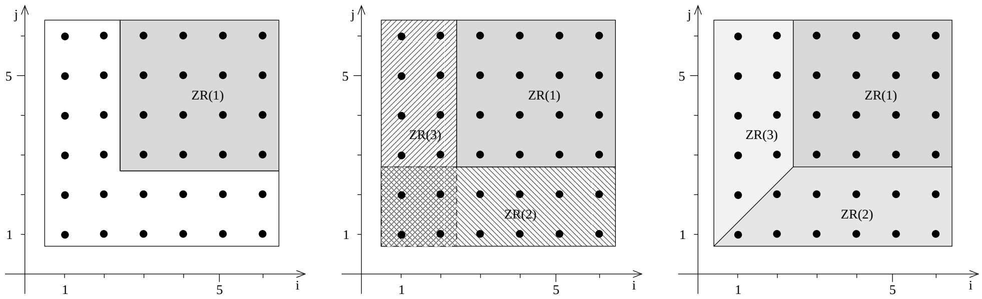
Zur Veranschaulichung sei der folgende Quast gegeben:

if $(i \geq 3)$ **then**
  if $(j \geq 3)$ **then**
    $< S, \, function\_1 >$
  else
    $< T, \, function\_2 >$
else
  $< T, \, function\_2 >$

$\xrightarrow{Vereinfachung}$ (⟹)

if $(i \geq 3) \wedge (j \geq 3)$ **then**
  $< S, \, function\_1 >$
else
  $< T, \, function\_2 >$

Die Zielräume, die sich bei der Zerlegung des Quasts ergeben würden, sind in Abbildung 5.10(a) dargestellt.



(a) (b) (c)

Abbildung 5.10: Zerlegung der Zielräume in konvexe Teilmengen

Die Negation der Konjunktion beschreibt eine Menge, die infolge von Satz 2.2.23 nicht konvex und damit kein Polyeder ist.

Um Konvexität herzustellen, werden hier zwei Wege aufgezeigt:

1.) Das Negieren einer Konjunktion $(\wedge : i : i \leq i \leq n : condition\_i)$ führt zu einer Disjunktion mit Normalform $(\vee : i : 1 \leq i \leq 1 : \neg condition\_i)$. Nun können durch Aufspaltung des else-Zweiges n verschiedene Abhängigkeiten erzeugt werden, wobei man den Zielraum der i-ten Abhängigkeit durch Hinzufügen von *condition_i* zum Iterationsraum des Zielstatements erhält. Weil jetzt nur jeweils eine Ungleichung zum Iterationsraum hinzukommt, bleibt nach Satz 2.2.23 die Konvexität erhalten.

   Bezogen auf das zuvor genannte Beispiel erhält man auf diese Weise drei Abhängigkeiten:

Abhängigkeit	Zielraum	Quelle	h-Transformation
1	$3 \leq i \leq 6$ $3 \leq j \leq 6$	S	$function\_1$
2	$1 \leq i \leq 6$ $1 \leq j \leq 2$	T	$function\_2$
3	$1 \leq i \leq 2$ $1 \leq j \leq 6$	T	$function\_2$

   Vorteilhaft ist dieses Verfahren, weil es einfach umzusetzen ist. Allerdings wird aus der grafischen Darstellung in Abbildung 5.10(b) auch der

Nachteil sofort deutlich. Dieser besteht in der mehrfachen Aufzählung aller Punkte des Iterationsraumes im Durchschnitt von je zwei Halbräumen, die die Bedingungen $\neg condition\_i$ und $\neg condition\_k$, $i \neq k$, repräsentieren. (Im Beispiel sind das die Punkte (i, j) mit $1 \leq i, j \leq 2$.)

2.) Die Zerlegung des Zielraumes in konvexe Teilmengen kann aber auch durch einen gesonderten Algorithmus vorgenommen werden, der zumindest garantiert, daß die berechneten Teilmengen paarweise disjunkt sind, der gegebenenfalls aber auch die Anzahl dieser Teilmengen minimiert.

Für das vorherige Beispiel läßt sich auf diese Weise die folgende Zerlegung (grafisch dargestellt in Abbildung 5.10(c)) bestimmen:

Abhängigkeit	Zielraum	Quelle	h-Transformation
1	$3 \leq i \leq 6$ $3 \leq j \leq 6$	S	$function\_1$
2	$1 \leq i \leq 6$ $1 \leq j \leq i$	T	$function\_2$
3	$1 \leq i \leq 2$ $i+1 \leq j \leq 6$	T	$function\_2$

Eine derartige Zerlegung in einem beliebigen endlich-dimensionalen Raum zu finden, stellt sich als eine nicht-triviale Aufgabe heraus. Ein Programm, das dieses leistet, ist im Rahmen des *Omega*-Projektes [PW96] entstanden.

5.2.4 Quellraumberechnung

Nachdem die Zielräume berechnet worden sind, lassen sich die zugehörigen Quellräume durch Funktionsminimierung und -maximierung gemäß dem in Abschnitt 2.4 vorgestellten Prinzip bestimmen.

Als Zielfunktion dient hierbei die h-Transformation der jeweiligen Abhängigkeit.

Man beachte, daß das sich ergebende Polyeder aufgrund der Bedingung (III) in (5.1.0.1) eine Teilmenge vom Iterationsraum des Quellstatements ist. Auf die Bildung des Durchschnitts beider Räume kann daher verzichtet werden.

5.2.5 Richtungs- und Distanzvektor

Die h-Transformation ordnet einem Vektor des Zielraumes den zugehörigen (und gemäß Korollar 3.3.9 eindeutig bestimmbaren) Vektor des Quellraumes zu. Nach Definition 3.3.13 ist die Differenz beider Vektoren gerade der Distanzvektor. Unter Berücksichtigung von Bemerkung 3.3.19 gilt:

$$\vec{d}\left(\vec{b}\right) = \pi_{<1,\ldots,N_{ST}>}\,\vec{b} - \pi_{<1,\ldots,N_{ST}>}\,S\left(\vec{b}\right) \tag{5.2.0.12}$$

Der Richtungsvektor ergibt sich dann durch Anwendung von Definition 3.3.18 auf $\vec{d}\left(\vec{b}\right)$.

Kapitel 6

Datenabhängigkeitsanalyse nach Banerjee

6.1 Der ursprüngliche Ansatz

6.1.1 Die Idee

In [Ban93] stellt Banerjee ein Verfahren vor, mit dem prinzipiell alle Arten von Abhängigkeiten berechnet werden können. Hierbei unterscheidet sich sein Ansatz grundlegend von dem Feautriers. Während bei letztgenanntem zu einem gegebenen Lesezugriff die Gesamtheit aller Schreibzugriffe auf dasselbe Array analysiert wird, betrachtet Banerjee je zwei Zugriffe isoliert. Schon hier ist erkennbar, daß bei der Betrachtung eines solchen Paares durch den Verzicht auf die Einbeziehung aller Zugriffe, die potentiell Quelle der zu berechnenden Abhängigkeit sind, schwächere Ergebnisse zu erwarten sind.

Zu Beginn werden zusätzliche Restriktionen der Eingabespache eingeführt, ohne die Banerjees Ansatz nicht anwendbar ist. Diese Beschränkungen werden im Zuge der Erweiterung der bestehenden Theorie (6.2) aufgehoben werden.

Unterteilen läßt sich das Verfahren in drei Schritte. Zu Beginn erfolgt für jedes Paar von Zugriffen die Berechnung der Menge aller Speicherkonflikte. Im darauffolgenden Schritt wird diese Menge durch Bezugnahme auf den Indexraum (dieser ist wegen 6.1.2 (8) eindeutig und infolge von 6.1.2 (7) nichttrivial) insoweit reduziert, als dadurch nur noch Speicherkonflikte tatsächlich ausgeführter Operationen enthalten sind. Schließlich wird die Richtung der resultierenden Abhängigkeiten festgelegt, so daß ihre Distanzvektoren gemäß Korollar 3.3.14 lexikographisch nicht negativ sind.

6.1.2 Zusätzliche Restriktionen

Ein Eingabeprogramm ist zulässig, wenn es die in Kapitel 4 genannten Voraussetzungen erfüllt und darüberhinaus folgenden Bedingungen genügt:

(7) Jedes Statement des Programms wird von mindestens einer Schleife umgeben.

(8) Das Programm ist perfekt verschachtelt.

(9) Es treten keine null-dimensionalen Datenräume auf.

(10) Das Programm ist frei von Strukturparametern.

(11) Während der Analyse wird ein Lösungspolyeder für Integervektoren berechnet werden (vgl. (6.1.1.5)). Dieses Polyeder ist entweder leer oder hat höchstens die Dimension 1.[3]

(12) In den Grenzen der Schleifen tritt kein min- oder max-Operator auf.

6.1.3 Formale Darstellung

Seien S und T, $S \leq T$, zwei nicht notwendigerweise verschiedene Statements eines zulässigen und aus m Schleifen bestehenden Eingabeprogrammes, mit Zugriff auf dasselbe Array. Die Datenraumdimension des Arrays sei n. Seien g_S, g_T die zugehörigen Indexfunktionen und ferner $\vec{i} \in D_S, \vec{j} \in D_T$.

Schritt 1:

Die Indexfunktionen g_S und g_T lassen sich durch Matrizen $A, B \in \mathbb{K}^{m \times n}$ und $a_0, b_0 \in \mathbb{K}^{1 \times n}$ darstellen, welche die linearen und konstanten Anteile dieser Abbildungen vermitteln.

[3] Zwar behandelt Banerjee ebenfalls höherdimensionale Lösungsräume, jedoch werden Berechnungen nur für den Fall zweidimensionaler Polyeder und unter zusätzlichen Beschränkungen durchgeführt.

Es gelte:

$$
\begin{aligned}
g_S\left(\vec{i}\right) &= \vec{i}\,(A) + a_0 \\
g_T\left(\vec{j}\right) &= \vec{j}\,(B) + b_0
\end{aligned}
\tag{6.1.1.1}
$$

Die Gesamtheit aller Speicherkonflikte wird durch die Menge der Integervektoren beschrieben, die (3.3.1.2) genügen. Zusammen mit (6.1.1.1) folgt dann:

$$
\left(\vec{i}, \vec{j}\right) \begin{pmatrix} A \\ -B \end{pmatrix} = (b_0 - a_0) \tag{6.1.1.2}
$$

Jede ganzzahlige Lösung von (6.1.1.2) erzeugt einen Speicherkonflikt. Diese Lösungen können gemäß Theorem 2.2.10 berechnet werden: durch Echelon-Reduktion (Definition 2.2.9) erhält man Matrizen $U \in \mathbb{K}^{2m \times 2m}$, $S \in \mathbb{K}^{2m \times n}$, für die gilt:

$$
U \begin{pmatrix} A \\ -B \end{pmatrix} = S \tag{6.1.1.3}
$$

Gibt es nun Integervektoren $\vec{t} \in \mathbb{Z}^{1 \times 2m}$ mit

$$
\vec{t} S = (b_0 - a_0)\,, \tag{6.1.1.4}
$$

so werden die gesuchten ganzzahligen Lösungen durch $\vec{t} U$ beschrieben. Der so bestimmte Lösungsraum stellt das unter 6.1.2 (11) genannte Polyeder dar, dessen Dimension vorläufig nicht größer als 1 ist.

Spaltet man die Matrix U in zwei Matrizen $U_1, U_2 \in \mathbb{K}^{2m \times m}$ auf, so ergibt sich als Lösung für die Iterationsvektoren $\vec{i}, \vec{j}$:

$$
\left(\vec{i}, \vec{j}\right) = \vec{t}\,(U_1, U_2) \tag{6.1.1.5}
$$

Schritt 2:

Im folgenden werden unter den zuvor ermittelten Lösungsvektoren diejenigen ausgewählt, die zu tatsächlich ausgeführten Operationen gehören. Hierzu geht nun der Indexraum in die Berechnung ein. Dieser läßt sich durch obere

Dreiecksmatrizen P, Q und Spaltenvektoren p_0, q_0 derart darstellen, daß für den Vektor $I = (i_1, ..., i_n)$ aller Schleifenindizes gilt:

$$\begin{aligned} p_0 \ &\leq I\,P \\ I\,Q \leq \ &q_0 \end{aligned} \tag{6.1.1.6}$$

(Wegen 6.1.2 (12) gilt für die soeben eingeführten Matrizen: $P, Q \in \mathbb{K}^{m \times m}$ und $p_0, q_0 \in \mathbb{K}^{1 \times m}$.)

Folglich erfüllen die Integervektoren $\vec{i}$ und $\vec{j}$ die Bedingungen (3.3.1.1) und (3.3.1.2) genau dann, wenn sie sowohl (6.1.1.5) als auch (6.1.1.6) genügen.

Hieraus ergibt sich ein Ungleichungssystem bestehend aus $4m$ Ungleichungen, das sich nach Transposition mit einem der unter 2.3 genannten Algorithmen lösen läßt:

$$\vec{t}\,(U_1P, -U_1Q, U_2P, -U_2Q) + (-p_0, q_0, -p_0, q_0) \geq 0 \tag{6.1.1.7}$$

Die Lösung dieses Systems wird durch ein Polyeder beschrieben, das mit L bezeichnet wird.

Schritt 3:

Als mögliche Lösungen für (6.1.1.7) lassen sich die folgenden zwei Fälle unterscheiden:

1) Das Lösungspolytop L ist leer. In diesem Fall werden durch das betrachtete Zugriffspaar keine Abhängigkeiten zwischen S und T erzeugt.

2) Die Dimension des Lösungspolytops ist 0 oder 1. Durch das analysierte Paar von Zugriffen wird mindestens eine Abhängigkeit zwischen S und T erklärt.

Im zweiten Fall muß die Richtung der Abhängigkeit(en) bestimmt werden. Aus mathematischer Sicht etspricht dieses der Erfüllung der letzten der drei Bedingungen in der Definition 3.3.1 einer Abhängigkeit und sichert somit die Nicht-Negativität der Distanzvektoren gemäß Korollar 3.3.14. Wiederum werden zwei Fälle unterschieden:

2a) $S <_{txt} T$.
Statement T hängt von Statement S ab, falls $\vec{i} \,\underline{\ll}\, \vec{j}$. Wegen (6.1.1.5) ist diese Aussage äquivalent zu

$$0 \,\underline{\ll}\, \vec{t}(U_2 - U_1)\,. \tag{6.1.1.8}$$

Die Menge aller $\vec{t}$, die (6.1.1.8) erfüllen, ist Teilmenge von L und wird mit $L_{\underline{\ll}}$ bezeichnet.

Umgekehrt hängt S von T ab, falls

$$0 \ll \vec{t}(U_1 - U_2)\,. \tag{6.1.1.9}$$

Die Lösungsmenge von (6.1.1.9) wird mit $L_{\gg}$ bezeichnet.

2b) $S = T$.
Zwischen zwei Operationen von S besteht eine Abhängigkeit, falls

$$0 \ll \vec{t}(U_2 - U_1)\,. \tag{6.1.1.10}$$

In diesem Fall ist $L_{\ll}$ die Lösungsmenge von (6.1.1.10).

Die in (6.1.1.10), (6.1.1.9) und (6.1.1.10) berechneten Vektoren sind genau die Distanzvektoren der jeweiligen Abhängigkeit.

Schließlich ergibt sich die Art der erzeugten Abhängigkeit aus dem Typ der Zugriffe in S und T.

6.1.2 Bemerkung (statementgetragene Abhängigkeiten)
Banerjee berechnet keine statementgetragenen Abhängigkeiten eines Statements auf sich selbst. Um nun Ergebnisse zu erhalten die mit dem erweiterten Ansatz von Feautrier vergleichbar sind (vgl. Beispiel 5.2.2), kann im Fall 2b anstelle von $L_{\ll}$ die Menge $L_{\underline{\ll}}$ aus 2a verwendet werden. Jedoch ist auf die Reihenfolge der Zugriffe entsprechend der zugriffsbasierten textuellen Ordnung aus Abschnitt 5.2.1.2 zu achten. Dabei vereinfacht sich (5.2.0.10) wegen $S = T$ zu

$$z_1 <_{acc} z_2 \quad \Longleftrightarrow \quad \psi(z_1) < \psi(z_2)$$

Folglich ist bei 2b für $z_1 <_{acc} z_2$ die Menge $L_{\underline{\ll}}$, andernfalls die Menge $L_{\ll}$ zu verwenden.

6.1.3 Bemerkung (Konvexität)
Die erste Komponente von $\vec{i}$ (bzw. $\vec{j}$) wird durch (6.1.1.5) als Funktion f_1^i (bzw. f_1^j) in Abhängigkeit von $\vec{t}$ dargestellt.

Offensichtlich existiert genau ein $t_\alpha \in \mathbb{Q}^{2m}$, mit $f_1(\vec{t_\alpha}) := (f_1^j - f_1^i)(\vec{t_\alpha}) = 0$. t_α beschreibt eine Hyperebene im Raum aller Lösungen von (6.1.1.5) für $\vec{t}$ und grenzt $L_{\underline{\ll}}$ von $L_{\gg}$ ab. Nach Satz 2.2.23 sind dann sowohl $L_{\underline{\ll}}$ als auch $L_{\gg}$ konvex, und sie lassen sich als Differenz der jeweils anderen Menge und L darstellen.

In Konsequenz von Satz 2.2.14 sind auch die Mengen der Iterationsvektoren $\vec{i}$ und $\vec{j}$ (das sind – je nach Richtung der Abhängigkeit– der Quell- und der Zielraum) konvex.

6.1.4 Bemerkung (Auswertung)
Die vorstehende Bemerkung induziert ein Verfahren zur Berechnung von $L_{\gg}$ und $L_{\ll}$:

Mit den Bezeichnungen aus 6.1.3 sei $L_{f_1} = \left\{\vec{t} \mid f_1\left(\vec{t}\right) \geq 0, \vec{t} \in L\right\}$ und ferner $t_\alpha \in \ker(f_1)$.

Dann gilt

$$L_{\gg} = \begin{cases} L_{f_1} \backslash t_\alpha & \text{falls } \ker(f_1) \cap L \neq \emptyset \wedge 0 = t_\alpha(U_1 - U_2) \\ L_{f_1} & \text{sonst} \end{cases} \tag{6.1.4.1}$$

beziehungsweise

$$L_{\ll} = \begin{cases} L \backslash (L_{f_1} \backslash t_\alpha) & \text{falls } \ker(f_1) \cap L \neq \emptyset \wedge 0 \ll t_\alpha(U_2 - U_1) \\ L \backslash L_{f_1} & \text{sonst.} \end{cases} \tag{6.1.4.2}$$

Die Menge $L_{\underline{\ll}}$ in 2a ergibt sich sofort aus der Differenz $L \backslash L_{\gg}$.

6.1.4 Beispiel

Gegeben sei das Programm aus Abbildung 6.1. Aus den Indexfunktionen der Zugriffe auf X wird deutlich, daß für beide Statements keine Abhängigkeit auf sich selbst existieren kann. Es folgt die Berechnung der Abhängigkeiten zwischen $S1$ und $S2$.

Schritt 1:

Die folgenden Matrizen ergeben sich unmittelbar aus der Eingabe:

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} \qquad a_0 = \begin{pmatrix} -4 & -9 & 2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b_0 = \begin{pmatrix} -1 & 3 & 173 \end{pmatrix}$$

Damit lassen sich die Indexfunktionen (6.1.1.1) sowie die Menge aller Speicherkonflikte (6.1.1.2) darstellen.
Die Echelon-Reduktion (6.1.1.3) liefert für U und S:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 4 & 1 & -8 & 2 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 & 2 \\ 0 & -1 & 4 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Für $\vec{t} = (t_1, ..., t_4)$ ergibt sich dann aus (6.1.1.4):

$$t_1 = 3,\ t_2 = -12,\ t_3 = 213 \tag{6.1.4.3}$$

Folglich gilt nach (6.1.1.5) für $\vec{i}$ und $\vec{j}$:

$$\begin{aligned} \vec{i} &= (4t_4 - 21, t_4) \\ \vec{j} &= (-8t_4 + 213, 2t_4 - 12) \end{aligned} \tag{6.1.4.4}$$

Schritt 2:
Ebenfalls aus der Eingabe ermitteln lassen die Matrizen P, p_0, Q und q_0:

$$P = Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad p_0 = \begin{pmatrix} 20 & 10 \end{pmatrix} \qquad q_0 = \begin{pmatrix} 130 & 30 \end{pmatrix}$$

Zusammen mit den Ergebnissen für $\vec{i}$ und $\vec{j}$ ergibt sich aus (6.1.1.7) das folgende Ungleichungssystem:

$$\begin{pmatrix} 4t_4 - 21 \\ t_4 \\ -4t_4 + 21 \\ -t_4 \\ -8t_4 + 213 \\ 2t_4 - 12 \\ 8t_4 - 213 \\ -2t_4 + 12 \end{pmatrix} + \begin{pmatrix} -20 \\ -10 \\ 130 \\ 30 \\ -20 \\ -10 \\ 130 \\ 30 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

DO $i_1 = 20,\ 130$
  DO $i_2 = 10,\ 30$
    $X[i_1 - 4, 2i_2 - 9, 2i_1 + 2] = i_1 * i_2$ $\{S1\}$
    $Y[i_1, i_2] = X[2i_2 - 1, i_2 + 3, 173 - i_1]$ $\{S2\}$
  END DO
END DO

Abbildung 6.1: Beispielprogramm zum ursprünglichen Ansatz

Nach Auflösung dieses Systems erhält man für t_4:

$$11 \leq t_4 \leq 21 \tag{6.1.4.5}$$

Also:

$$L = \{11, 12, ..., 21\}$$

Schritt 3:
Offensichtlich trifft Fall 2a zu. Die Auswertung nach 6.1.4 liefert:

$$\begin{aligned} f_1\left(\vec{t}\right) &= \left(f_1^j - f_1^i\right)\left(\vec{t}\right) \\ &= (4t_4 - 21) - (-8t_4 + 213) \\ &= 12t_4 - 231 \end{aligned}$$

Hieraus ergibt sich für L_{f_1} und den Kern von f_1:

$$L_{f_1} = \{20, 21\}, \quad \ker(f_1) = \left\{ t_\alpha = \frac{231}{12} \right\}$$

Damit führt (6.1.4.1) zu

$$L_{\gg} = L_{f_1} = \{20, 21\}.$$

Aus $L \backslash L_{\gg}$ ergibt sich dann

$$L_{\underline{\ll}} = \{11, 12, 13, 14, 15, 16, 17, 18, 19\}.$$

Eine Auflistung der sich ergebenden Abhängigkeiten sowie eine grafische Darstellung finden sich auf Seite 60.

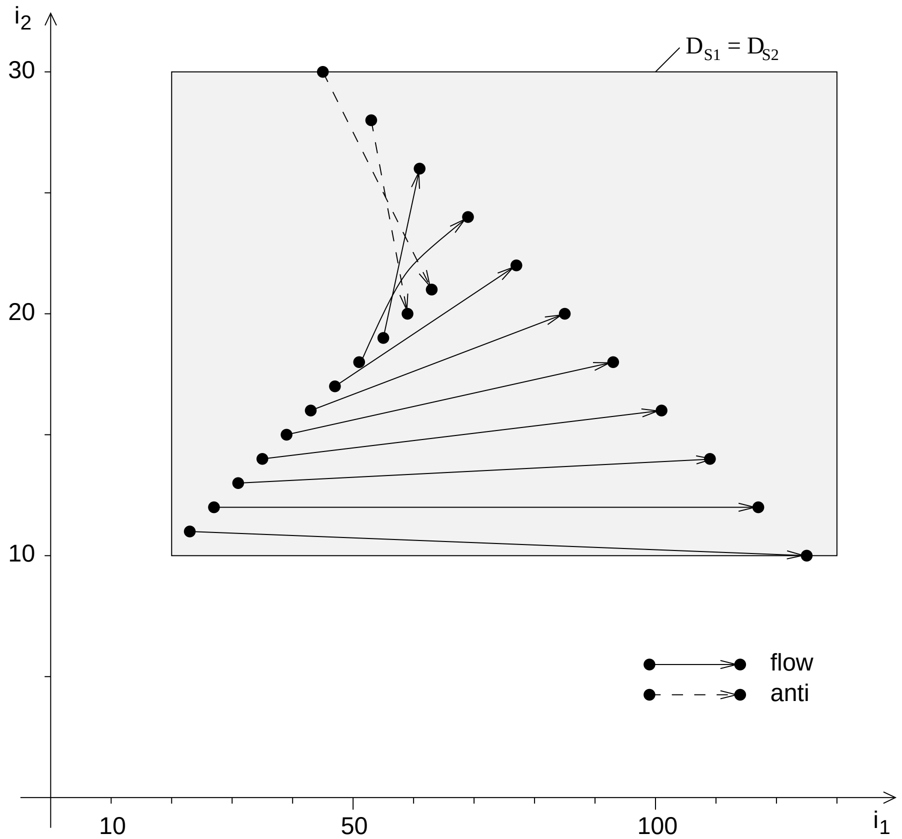


Abbildung 6.2: anti/flow-Abhängigkeiten

t_4	Abhängigkeit/Art	Distanzvektor	Richtungsvektor
11	$<S,\ (23,11)>\ \delta^t\ <T,\ (125,10)>$	$(102,-1)$	$(1,-1)$
12	$<S,\ (27,12)>\ \delta^t\ <T,\ (117,12)>$	$(90,\ 0)$	$(1,\ 0)$
13	$<S,\ (31,13)>\ \delta^t\ <T,\ (109,14)>$	$(78,\ 1)$	$(1,\ 1)$
14	$<S,\ (35,14)>\ \delta^t\ <T,\ (101,16)>$	$(66,\ 2)$	$(1,\ 1)$
15	$<S,\ (39,15)>\ \delta^t\ <T,\ (93,18)>$	$(54,\ 3)$	$(1,\ 1)$
16	$<S,\ (43,16)>\ \delta^t\ <T,\ (85,20)>$	$(42,\ 4)$	$(1,\ 1)$
17	$<S,\ (47,17)>\ \delta^t\ <T,\ (77,22)>$	$(30,\ 5)$	$(1,\ 1)$
18	$<S,\ (51,18)>\ \delta^t\ <T,\ (69,24)>$	$(18,\ 6)$	$(1,\ 1)$
19	$<S,\ (55,19)>\ \delta^t\ <T,\ (61,26)>$	$(6,\ 7)$	$(1,\ 1)$
20	$<S,\ (59,20)>\ \delta^a\ <T,\ (53,28)>$	$(6,\ 8)$	$(1,\ 1)$
21	$<S,\ (63,21)>\ \delta^a\ <T,\ (45,30)>$	$(18,\ 9)$	$(1,\ 1)$

Tabelle 6.1: Abhängigkeiten des Beispielprogrammes

6.1.5 Bemerkung

Das vorangegangene Beispiel legt die Vermutung nahe, daß es stets eine eindeutige Zuordnung zwischen Elementen des Quell- und des Zielraumes im Sinne von Korollar 3.3.9 gibt.

Diese Annahme stimmt für den hier beschriebenen Ansatz und unter den in 6.1.2 genannten Bedingungen, ist im allgemeinen jedoch falsch.

Theorem
Mit den in diesem Kapitel eingeführten Bezeichnungen sei f_S die durch S vermittelte lineare Abbildung und e eine mit diesem Ansatz berechnete Abhängigkeit. Es gilt:

$$\dim(\ker(f_S)) < 2 \quad \Longrightarrow \quad (\exists\, \varphi \,:\, \varphi \text{ Bijektion} \,:\, QR(e) \stackrel{\varphi}{\to} ZR(e))$$

Beweis:

Zeige: φ nicht bijektiv $\Longrightarrow$ $\dim(\ker(f_S)) \geq 2$.
Es genügt, den Fall zu betrachten, in dem φ nicht injektiv ist (vgl. Beweis Korollar 3.3.9).

O.B.d.A. gelte

$$\begin{array}{ll}
 & \vec{s_1}, \vec{s_2} \in QR(e),\ \vec{t} \in ZR(e),\ \vec{s_1} \neq \vec{s_2},\ \varphi\left(\vec{s_1}\right) = \varphi\left(\vec{s_1}\right) = \vec{t} \\
\Rightarrow & \{\ (6.1.1.5)\ \} \\
 & (\exists\, \vec{t_1}, \vec{t_2}, \vec{t_3} \,:\, \vec{t_i} \in L, 1 \leq i \leq 3 \,:\, \left(\vec{s_1}, \vec{s_2}, \vec{t}\right) = \left(\vec{t_1}, \vec{t_2}, \vec{t_3}\right) U) \\
\Rightarrow & \{\ U \text{ unimodular}\ \} \\
 & Rg\left(\vec{s_1}, \vec{s_2}, \vec{t}\right) = Rg\left(\vec{t_1}, \vec{t_2}, \vec{t_3}\right) \\
\Rightarrow & \{\ \text{Definition } L, \text{ Seite } 55\ \} \\
 & Rg\left(\vec{t_1}, \vec{t_2}, \vec{t_3}\right) = \dim(\ker(f_S)) \\
\Rightarrow & \{\ \ Rg\left(\vec{s_1}, \vec{s_2}, \vec{t}\right) \geq 2\ \ \} \\
 & \dim(\ker(f_S)) \geq 2
\end{array}$$

□

Nun wird deutlich, daß der Grund für die im ursprünglichen Ansatz stets existierende Bijektion in der Restriktion 6.1.2 (11) zu suchen ist. Weil dort ein Lösungsraum (Kern von f_S) mit höchstens Dimension 1 gefordert wird, folgt aus obigem Theorem die Existenz von φ.

Im Zuge der nun anstehenden Erweiterung wird gezeigt werden, daß durch die Aufhebung von Restriktion (11) tatsächlich der Fall eintreten kann, in dem die Bijektion φ nicht existiert. Eine Darstellung sowie die Lösung der sich hieraus ergebenden Probleme wird Hauptbestandteil von Abschnitt 6.2.2 sein.

6.2 Die Erweiterung des Ansatzes

In diesem Abschnitt erfolgt die Erweiterung des ursprünglichen Ansatzes, in deren Verlauf zunächst sämtliche unter 6.1.2 angeführten Restriktionen aufgehoben werden. Es schließt sich das Generieren einer h-Transformation an, wobei ein Verfahren vorgestellt wird, das die Aufhebung der Restriktionen bereits berücksichtigt.

6.2.1 Aufhebung der zusätzlichen Restriktionen

6.2.1.1 Parametrisierung, min/max-Operatoren und Lösungspolyeder höherer Dimension mit Beispiel

Zunächst werden die Beschränkungen (10) und (12) aufgehoben, so daß alle Eingabeprogramme, die einem statischen Kontrollfluß gemäß Bedingung 4 (2) unterliegen, zugelassen sind. Gleichzeitig muß die Möglichkeit geschaffen werden, beliebigdimensionale Lösungspolyeder zu handhaben, weil im Falle einer Verallgemeinerung der Eingabe im soeben beschriebenen Sinn Bedingung (11) nicht länger garantiert werden kann, wie man im folgenden sehen wird.

Das Eingabeprogramm enthalte nun mit den Bezeichnungen aus 6.1.3, jedoch ohne die Bedingungen 6.1.2 (10, 11, 12), p Strukturparameter $sp_1, ..., sp_p$.

Desweiteren ist nach Aufhebung von Bedingung 6.1.2 (12) die Verwendung von $\min / \max -$Operatoren in der Beschreibung der oberen bzw. unteren Grenzen möglich. Diese erfolgt nach der Maßgabe von 4 (2), und mit den in Definition 3.1.1 eingeführten Bezeichnungen gilt:

$$B_l := \sum_{j=1}^{m} K_{(j)}$$

$$B_u := \sum_{j=1}^{m} K^{(j)}$$

```
DO i1 = 1, 6
    DO i2 = 1, min(i1, 5)
        X[i1, i2] = X[i1-n-1, i2+2]    {S1}
    END DO
END DO
```

Abbildung 6.3: Beispielprogramm zu 6.2.1.1

Zum besseren Verständnis werden neben der formalen Darstellung die Auswirkungen dieser Erweiterung anhand des Beispielprogrammes in Abbildung 6.3 dargestellt. Für dieses Programm gilt offensichtlich $p=1$, $B_l=2$, $B_u=3$, $m=2$ und $n=2$.

Schritt 1:

Der konstante Anteil der Indexfunktionen g_S und g_T in (6.1.1.1) wird nun durch Matrizen $a_0, b_0 \in \mathbb{K}^{(p+1)\times n}$ vermittelt:

$$\begin{aligned} g\left(\vec{i}\right) &= \vec{i}\,(A) + (sp_1, ..., sp_p, 1)\; a_0 \\ g\left(\vec{j}\right) &= \vec{j}\,(B) + (sp_1, ..., sp_p, 1)\; b_0 \end{aligned}$$

Aus dem Beispielprogramm ergibt sich dann:

$$\begin{aligned} g\left(\vec{i}\right) &= \vec{i}\,(A) + (n,1)\; a_0 \;=\; \vec{i} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + (n,1) \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ g\left(\vec{j}\right) &= \vec{j}\,(B) + (n,1)\; b_0 \;=\; \vec{j} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + (n,1) \begin{pmatrix} -1 & 0 \\ -1 & 2 \end{pmatrix} \end{aligned}$$

Gleichung (6.1.1.2) bleibt erhalten, doch beschreibt nun auch die Differenz $(b_0 - a_0)$ nun eine Matrix in $\mathbb{K}^{(p+1)\times n}$.

Die Berechnung von U und S erfolgt in der gewohnten Weise. Für das Beispiel ergibt sich:

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Man beachte, daß der Rang von S in (6.1.1.3) durch den Wegfall von Bedingung 6.1.2 (12) frei zwischen 0 und $2m$ variieren kann. Dieses ist gleichbedeutend damit, daß in (6.1.1.4) nun auch Lösungspolyeder höherer Dimension auftreten können.

Da in dieser Gleichung auch Parameter auftreten können, erfolgt die Darstellung des Vektors $\vec{t}$ – wie schon bei a_0 und b_0 – spaltenweiseweise durch $(p+1) \times 1$-Matrizen.

Damit läßt sich (6.1.1.4) wie folgt schreiben:

$$\vec{t}S = (t_1, ..., t_{2m})\, S = (b_0 - a_0)\,, \qquad \vec{t_i} \in \mathbb{Z}^{(p+1)\times 1},\ 1 \le i \le 2m. \tag{6.2.0.6}$$

Hierdurch wird ein System von $p+1$ inhomogenen Gleichungssystemen dargestellt.

Im konkreten Beispiel:
Das System

$$\vec{t}S = \begin{pmatrix} t_{11} & t_{21} & t_{31} & t_{41} \\ t_{12} & t_{22} & t_{32} & t_{42} \end{pmatrix} S = \begin{pmatrix} -1 & 0 \\ -1 & 2 \end{pmatrix}$$

beschreibt die Gleichungssysteme

$$\begin{pmatrix} t_{11} & t_{21} & t_{31} & t_{41} \end{pmatrix} S = \begin{pmatrix} -1 & 0 \end{pmatrix}$$

und

$$\begin{pmatrix} t_{12} & t_{22} & t_{32} & t_{42} \end{pmatrix} S = \begin{pmatrix} -1 & 2 \end{pmatrix}.$$

Durch Lösen dieser Systeme ergibt sich für $\vec{t}$:

$$\vec{t} = \begin{pmatrix} \begin{matrix} 1 \\ 1 \end{matrix}, \begin{matrix} 0 \\ -2 \end{matrix}, t_3, t_4 \end{pmatrix}, \quad \begin{matrix} \leftarrow n \\ \leftarrow \textit{offset} \end{matrix}$$

welches dem Ausdruck $\vec{t} = (n{+}1, -2, t_3, t_4)$ entspricht.

Aus Gleichung (6.1.1.5) folgt dann bei Fortsetzung der komponentenweisen Darstellung durch $(p{+}1) \times 1$-Matrizen für $\left(\vec{i}, \vec{j}\right)$:

$$\begin{aligned}\left(\vec{i}, \vec{j}\right) &= \left((i_1, i_2), (j_1, j_2)\right) \\ &= \vec{t}\,(U_1, U_2) \qquad\qquad (6.2.0.7) \\ &= \left((t_3, t_4),\, t_3 + \begin{pmatrix} 1 \\ 1 \end{pmatrix},\, t_4 + \begin{pmatrix} 0 \\ -2 \end{pmatrix}\right),\end{aligned}$$

welches sich wiederum schreiben läßt als

$$\left(\vec{i}, \vec{j}\right) = \left((t_3, t_4), (t_3{+}n{+}1, t_4{-}2)\right). \qquad (6.2.0.8)$$

Durch die Menge aller so gefundenen Vektoren $\vec{i}$ (bzw. $\vec{j}$) wird dann ein Raum mit maximaler Dimension m (der Dimension des zugehörigen Indexraumes) beschrieben.

Schritt 2:

Prinzipiell werden die Matrizen P und Q aus (6.1.1.6) wie gewohnt ermittelt. Jedoch ist zu bemerken, daß sie nicht mehr notwendigerweise quadratisch sind, weil durch die Verwendung von min/max-Operatoren eine beliebige Anzahl unterer sowie oberer Grenzen zulässig ist. Gleiches gilt für die Matrizen p_0 und q_0, doch kommt hier zusätzlich noch die Parametrisierung hinzu, die wiederum eine komponentenweise Darstellung in $(p{+}1) \times 1$-Matrizen erforderlich macht.

Für die unteren und oberen Grenzen ergibt sich

$$\begin{aligned}(sp_1, \ldots., sp_p, 1)\, p_0 &\le I\,P \\ I\,Q &\le (sp_1, \ldots., sp_p, 1)\, q_0\end{aligned}$$

mit $P \in \mathbb{K}^{m \times B_l}$, $p_0 \in \mathbb{K}^{(p+1) \times B_l}$ und $Q \in \mathbb{K}^{m \times B_u}$, $q_0 \in \mathbb{K}^{(p+1) \times B_u}$.

Bezogen auf das Beispiel heißt das:

$$\begin{aligned}(n,\, 1)\, p_0 &\le (i_1,\, i_2)\, P \\ (i_1,\, i_n)\, Q &\le (n,\, 1)\, q_0\end{aligned}$$

wobei

$$p_0 = \begin{array}{c} \xrightarrow{B_l} \\ \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \end{array} \Big\downarrow p{+}1\, , \qquad P = \begin{array}{c} \xrightarrow{B_l} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \Big\downarrow m\, ,$$

$$q_0 = \begin{array}{c} \xrightarrow{B_u} \\ \begin{pmatrix} 0 & 0 & 0 \\ 6 & 5 & 3 \end{pmatrix} \end{array} \Big\downarrow p{+}1\, , \qquad Q = \begin{array}{c} \xrightarrow{B_u} \\ \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \end{pmatrix} \end{array} \Big\downarrow m\, .$$

Die Lösung von (6.1.1.5) läßt sich dann in Analogie zum ersten Schritt durch Lösen $p{+}1$ inhomogener Ungleichungssyteme bestimmen.

Für das Beispiel erhält man

$$(t_{31},\, t_{41}) \begin{pmatrix} 1 & 0 & 1 & 0 & -1 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & -1 & -1 & 0 & -1 & -1 \end{pmatrix} + (0,0,1,0,0,0,0,-1,0,1) \geq 0$$

und

$$(t_{32},\, t_{42}) \begin{pmatrix} 1 & 0 & 1 & 0 & -1 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & -1 & -1 & 0 & -1 & -1 \end{pmatrix} + (-1,-1,0,3,6,5,3,5,7,6) \geq 0$$

Als Lösung dieser Ungleichungssysteme findet sich:

$$\begin{aligned} \max(1,-n) \leq\ & t_3 \leq \min(6, 5-n) \\ 3 \leq\ & t_4 \leq \min(5, t_3+3, t_3+n+6) \end{aligned}$$

Dieses Ergebnis läßt sich durch Anwendung von 4 (6) vereinfachen:

$$\begin{aligned} 1 \leq\ & t_3 \leq 5-n \\ 3 \leq\ & t_4 \leq \min(5, t_3+3) \end{aligned} \tag{6.2.0.9}$$

Schritt 3:

Die allgemeine Behandlung beliebig-dimensionaler Lösungspolytope erfordert besondere Aufmerksamkeit.

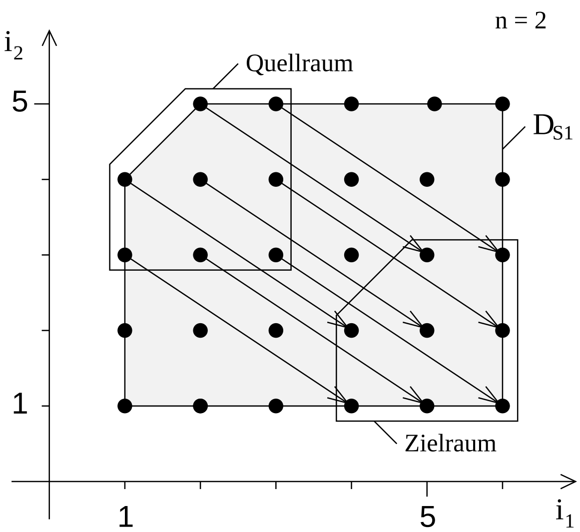


Abbildung 6.4: Abhängigkeiten zum Beispielprogramm in Abbildung 6.3

Weil die Bedingung 6.1.2 (11) aufgehoben wurde und deshalb nicht mehr mit Hilfe des Theorems in Abschnitt 6.1.5 auf die Existenz einer Bijektion zwischen Quell- und Zielreum geschlossen werden kann, ergeben sich neuerliche Probleme, deren Lösung Gegenstand der vier Abschnitte 6.2.2 – 6.2.5 am Ende dieses Kapitels sein wird.

Dabei wird sich herausstellen, daß die genannten Abschnitte gemeinsam einen vollständigen Ersatz für den hier nicht ausgeführten dritten Schritt bilden.

Zu diesem Zeitpunkt wird ebenfalls klar sein, weswegen es bei dem Beispiel dieses Abschnittes nicht zu diesen Schwierigkeiten kommt. Hier lassen sich die abhängigen Operationen punktweise, also durch Einsetzen einzelner Lösungen von (6.2.0.9) in (6.2.0.8), ermitteln. Die Richtung der Abhängigkeit sowie der (konstante) Distanzvektor $(n{+}1,\ -2)$ und damit der Richtungsvektor $(1, -1)$ ergeben sich sofort. Abbildung 6.3 stellt diese Ergebnisse für $n\,{=}\,2$ dar.

6.2.1.2 Imperfekte Verschachtelung

In diesem Abschnitt wird eine der einschneidendsten Beschränkungen, die Annahme einer perfekten Verschachtelung, beseitigt. Um dieses zu erreichen, werden die Iterationsvektoren in folgender Weise vereinheitlicht:

```
DO i1 = 1, 5
   DO i2 = 1, 5
      S
   END DO
   T
END DO
```

Wird das Statement T im Gegensatz zum Statement S nicht von der Schleife L umgeben, so kann T ein fiktiver Ausführungszeitpunkt α_L von L zugeordnet werden. T erhält so den Iterationsraum $D_T \times \{\alpha_L\}$. In diesem Punkt kommt Bemerkung 3.1.4 zum Tragen. Zwar bleibt die Dimension des neuen Indexraumes in seiner grafischen Darstellung unverändert, jedoch wurde sie gemäß Definition 3.1.3 um 1 erhöht. Damit besitzen S und T Iterationsräume in denselben Schleifenindizes, wodurch die Voraussetzung geschaffen wurden, ein beliebiges Programm wie ein perfekt verschachteltes zu behandeln.

Die spezielle Wahl von α_L ist hierbei ohne Bedeutung. Der im jeweiligen Fall gewählte Wert stellt – wie zuvor erläutert – einen fiktiven Zeitpunkt dar und nimmt keinen Einfluß auf die tatsächliche Ausführungsreihenfolge im Programm.

Im Anschluß an die Berechnung kann die Vereinheitlichung der Iterationsvektoren durch Streichen aller zuvor eingefügten Komponenten wieder zurück genommen werden.

Schritt 1:

Seien $L_1, ...L_n$ die Schleifen, die mindestens eines der Statements S und T umgeben. Die Indizierung erfolgt in der Reihenfolge der textuellen Ordnung. Statement S (bzw. T) wird der Iterationsvektor $\vec{i} = (i_1, ..., i_n)$ (bzw. $\vec{j} = (j_1, ..., j_n)$) zugeordnet. Wird S (bzw. T) von der Schleife L_l, $1 \le l \le n$ nicht umgeben, so ist i_l (bzw. j_l) auf einen beliebigen Ausführungszeitpunkt α_L zu setzen. In diesem Fall bildet α_L also die obere und untere Grenze der Schleife L_l.

Schritt 2:

Die Matrizen P, Q, p_0 und q_0 sind respektive der soeben durchgeführten Erweiterung der Iterationsvektoren anzugeben. Das bedeutet insbesondere, daß es nicht mehr möglich ist, diese Matrizen wie bisher für $\vec{i}$ und $\vec{j}$ gemeinsam zu bilden. Vielmehr muß jede dieser vier Matrizen für $\vec{i}$ und $\vec{j}$ getrennt angegeben werden, wobei die hochgestellte Indizierung (etwa $P^{\vec{i}}$) den zugehörigen Iterationsvektor bezeichnet. Wurde einer dieser Vektoren im

$$
\begin{array}{ll}
\textbf{DO } i_1 = 1,\, 4 & \{L_1\} \\
\quad \textbf{DO } i_2 = 1,\, 4 & \{L_2\} \\
\quad\quad A[i_2] = ... & \{\mathrm{S}\} \\
\quad \textbf{END DO} & \\
\quad A[i_1] = ... & \{\mathrm{T}\} \\
\textbf{END DO} &
\end{array}
$$

Abbildung 6.5: Beispielprogramm zur imperfekten Verschachtelung

vorherigen Schritt um die l-te Komponente erweitert, so ist diese beidseitig durch α_L beschränkt.

Formal gilt in Abwandlung von (6.1.1.6):

$$
\begin{aligned}
p_0^{\vec{i}} &\leq I\,P^{\vec{i}} \\
p_0^{\vec{j}} &\leq I\,P^{\vec{j}} \\
\\
I\,Q^{\vec{i}} &\leq q_0^{\vec{i}} \\
I\,Q^{\vec{j}} &\leq q_0^{\vec{j}}
\end{aligned}
\tag{6.2.0.10}
$$

Hieraus ergibt sich zusammen mit (6.1.1.5) eine Ersetzung für (6.1.1.7):

$$
\vec{t}\,\left(U_1P^{\vec{i}}, -U_1Q^{\vec{i}}, U_2P^{\vec{j}}, -U_2Q^{\vec{j}}\right) + \left(-p_0^{\vec{i}}, q_0^{\vec{i}}, -p_0^{\vec{j}}, q_0^{\vec{j}}\right) \geq 0 \tag{6.2.0.11}
$$

6.2.1.3 Beispiel

Gegeben sei das Programm aus Abbildung 6.5. Es sollen die Abhängigkeiten zwischen S und T berechnet werden. Enthält ein Vektor oder eine Matrix Komponenten, die durch die Auffüllung in Schritt 1 hinzu gekommen sind, so werden diese in eckige Klammern (etwa: $(i_1, [i_2])$) gesetzt.

Schritt 1:

Offensichtlich muß der Iterationsvektor $\vec{j}$ des Statements T erweitert werden. T wird der Ausführungszeitpunkt α_2 der zweiten Schleife L_2 zugeordnet. Für den Iterationsraum von T gilt also

$$D_T = \{1, 2, 3, 4\} \times \{\alpha_2\},$$

und der Iterationsvektor von T läßt sich schreiben als

$$\vec{j} = (i_1, [i_2]), \;\; i_2 := \alpha_2.$$

Mit dieser Vereinbarung ergeben sich aus der Eingabe die folgenden Matrizen:

$$A = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 \\ [0] \end{pmatrix} \qquad a_0 = b_0 = (0)$$

In gewohnter Weise gibt die Echelon-Reduktion (6.1.1.3) für U und S:

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Hieraus folgt $t_1 = 0$ und

$$\begin{aligned} \vec{i} &= (t_2, t_3) \\ \vec{j} &= (t_3, [t_4]) \end{aligned}$$

Schritt 2:
Nun werden gemäß (6.2.0.10) die Matrizen P, p_0, Q und q_0 sowohl für $\vec{i}$ als auch $\vec{j}$ bestimmt. Es gilt

$$P^{\vec{i}} = P^{\vec{j}} = Q^{\vec{i}} = Q^{\vec{j}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

und

$$p_0^{\vec{i}} = (1, 1)\,, \quad p_0^{\vec{j}} = (1, \alpha_2)\,, \quad q_0^{\vec{i}} = (4, 4)\,, \quad q_0^{\vec{j}} = (4, \alpha_2)\,.$$

Das durch (6.2.0.11) definierte Ungleichungssystem hat als Lösung:

$$\begin{aligned} 1 \leq \; & t_2 \; \leq 4 \\ 1 \leq \; & t_3 \; \leq 4 \\ \alpha_2 \leq \; & t_4 \; \leq \alpha_2 \end{aligned}$$

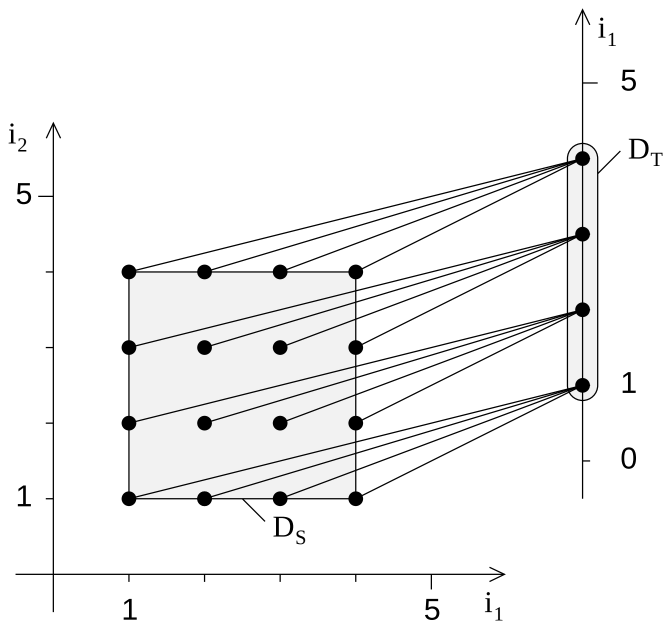


Abbildung 6.6: Iterationsvektoren abhängiger Operationen

und damit

$$
\begin{array}{rcl}
(1,\,1) \leq & (i_1,\,i_2) & \leq (4,\,4) \\
(1,\,[\alpha_2]) \leq & (j_1,\,[j_2]) & \leq (4,\,[\alpha_2])
\end{array}
$$

Abbildung 6.6 zeigt, zwischen welchen Elementen von D_S und D_T Abhängigkeiten bestehen. Wichtig hierbei ist, daß die Richtung im Einzelfall noch nicht feststeht. Tatsächlich gibt es sowohl Abhängigkeiten von S nach T als auch umgekehrt. Genau diese Analyse der Richtungen wird Gegenstand von Abschnitt 6.2.5 sein.

6.2.1.4 Null-dimensionale Iterationsräume

Mit dem im vorangegangenen Abschnitt eingeführten Verfahren für die imperfekte Verschachtelung sind auch Abhängigkeiten solcher Programme behandelbar, in denen nur eines der beiden Statements von mindestens einer Schleife umgeben wird. Insofern ist die Restriktion 6.1.2 (7) bereits teilweise aufgehoben worden. Nun soll der Fall betrachtet werden, bei dem keines der beiden Statement von einer Schleife umgeben werden.

$$P \quad \Longrightarrow \quad \begin{array}{l} \textbf{for}\ (\ i_1 = f_1;\ i_1 \leq f_1;\quad i_1{+}{+}\ )\ \{ \\ \quad \vdots \\ \qquad \textbf{for}\ (\ i_n = f_n;\ i_n \leq f_n;\quad i_n{+}{+}\ )\ \{ \\ \qquad\qquad P; \\ \qquad \} \\ \quad \vdots \\ \} \end{array}$$

Abbildung 6.7: Transformation für 0-dimensionale Iterationsräume

Die Idee hierbei ist, daß ein Programm P semantisch äquivalent zu einem perfekt verschachtelten Programm P' mit Schleifenrumpf P ist, falls P genau einmal ausgeführt wird.

Eine Veranschaulichung gibt Abbildung 6.7. Hier stellen $f_1, ..., f_n$, $1 \leq n$, beliebige Schleifengrenzen dar, die Bedingung 4 (2) genügen.

Den Statements des Programms P wird durch diese Transformation ein einelementiger Iterationsraum zugeordnet, wodurch die Bezeichnung des „nulldimensionalen Iterationsraumes“ nachträglich gerechtfertigt wird.

6.2.1.5 Skalare

Schließlich muß nur noch die aus 6.1.2 (9) resultierende Beschränkung aufgehoben werden.

Wie schon im vorangegangenen Abschnitt wird auch hier ein noch nicht analysierbares Programm auf ein bereits behandelbares zurückgeführt.

Grundgedanke hierbei ist, daß ein Skalar auch als ein bestimmtes Element eines Arrays aufgefaßt werden kann. Ersetzt man in einem Programm mit einem Skalar a jedes Vorkommen dieses Skalars durch den gleichbleibenden Zugriff auf ein Arrayelement, wie es in Abbildung 6.8 dargestellt ist, so erhält man ein semantisch äquivalentes Programm. Dabei ist $g_a = (g_{a_1}, ..., g_{a_m})$ (mit $1 \leq m$) eine affin lineare Funktion in den Strukturparametern, jedoch nicht in den Variablen umgebender Schleifen. Auf das transformierte Programm ist das bereits bekannte Analyseverfahren anwendbar.

$$
\begin{array}{lcl}
\vdots & & \vdots \\
a \ = ... & & a[g_{a_1}, ..., g_{a_m}] = ... \\
... = a & \Longrightarrow & ... \qquad\qquad = a[g_{a_1}, ..., g_{a_m}] \\
\vdots & & \vdots
\end{array}
$$

Abbildung 6.8: Transformation von Skalaren

6.2.1.6 Beispiel

Zur Veranschaulichung der beiden vorangegangenen Abschnitte wird das Eingabeprogramm aus Abbildung 6.9(a) analysiert.

Weil keines der beiden Statements von einer Schleife umgeben wird, ist die vorübergehende Annahme einer perfekten Verschachtelung gemäß 6.2.1.4 erforderlich. Diese besteht zur Anschauung aus nur einer Schleife ($n = 1$). Die obere und untere Grenze f_1 der Schleifenvariablen i_1 wird (willkürlich) auf 0 gesetzt.

Desweiteren muß jedes Auftreten des Skalars B in einen konstanten Zugriff auf ein m-dimensionales Array umgewandelt werden. Der Einfachkeit wegen wird $m = 1$ und $g_B = 0$ als Indexfunktion angenommen.

Durch diese Transformation ergibt sich das Programm aus Abbildung 6.9(b).

Aus der Eingabe ist zu entnehmen, daß gilt:

$$A = B = a_0 = b_0 = (0)$$

Hieraus folgt dann:

$$U = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad S = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Also gilt:

$$(i_1,\, j_1) = (t_1,\, t_2)$$

Ebenfalls aus der Eingabe zu ersehen ist:

$$P^{\vec{i}} = P^{\vec{j}} = Q^{\vec{i}} = Q^{\vec{j}} = p_0^{\vec{i}} = p_0^{\vec{j}} = q_0^{\vec{i}} = q_0^{\vec{j}} = (0)$$

Wie man leicht erkennt, folgt $t_1 = t_2 = 0$.

$$
\begin{array}{ll}
B = 2 * C + 7 & \{\mathrm{S}\} \\
D = 5 - B & \{\mathrm{T}\}
\end{array}
$$

(a) Eingabeprogramm

$$
\begin{array}{lll}
\textbf{for}\ (i_1\!=\!0;\, i_1 \le 0;\, i_1\!+\!+)\ \{ & & \\
\quad B[0] & = 2 * C + 7 & \{\mathrm{S}\} \\
\quad D & = 5 - B[0] & \{\mathrm{T}\} \\
\} & &
\end{array}
$$

(b) transformiertes Eingabeprogramm

Abbildung 6.9: Beispiel zu 6.2.1.4 und 6.2.1.5

Damit besteht die statementgetragene Abhängigkeit $<S,\,(0)>\,\delta^f\,<T,\,(0)>$ aus der sich für das Eingabeprogramm bei Streichung der durch die Transformation hinzugekommenen Komponenten als Endergebnis findet:

$$<S,\,()>\,\delta^f\,<T,\,()>$$

6.2.2 Generieren der h-Transformation

Wie bereits in Abschnitt 6.1.5 dargestellt, führt die Aufhebung von Restriktion 6.1.2 (11) zu dem Problem, daß nicht mehr auf die Existenz einer Bijektion zwischen Quell- und Zielraum geschlossen werden kann.

Ein einfaches Beispiel hierfür liefert das Programm in Abbildung 6.10. Hier gibt es Abhängigkeiten von S nach T zwischen den Operationen $<S,(1,i_2)>$ und $<T,(2,1)>$. Offensichtlich ist die Funktion

$$\varphi\ :\ \{1\}\times[1,5] \to \{2\}\times\{1\}\ :\ (i_1,\, i_2) \to (i_1+1,\, 1),$$

die den Elementen des Quellraumes die zugehörigen Elemente des Zielraumes zuordnet, nicht injektiv.

In umgekehrter Richtung führt die Ermittlung der Quellpunkte zu einem gegebenen Zielpunkt, also zur Berechnung der Urbildmenge des Zielpunktes unter φ gemäß Satz 2.2.1 über die Umkehrrelation.

Hierbei ergibt sich die Urbildmenge φ^{-1} aus einem Repräsentanten, etwa $(i_1,i_2) \to (i_1-1,0)$, sowie dem Kern von φ:

$$\varphi^{-1}(i_1,\, i_2) = (i_1-1,\, 0) + \lambda(0,\, 1), \qquad \lambda \in \mathbb{K} \tag{6.2.0.12}$$

DO $i_1 = 1, 2$
  DO $i_2 = 1, 5$
    $A[i_1] = ...$ {S}
    $... = A[i_1 - i_2]$ {T}
  END DO
END DO

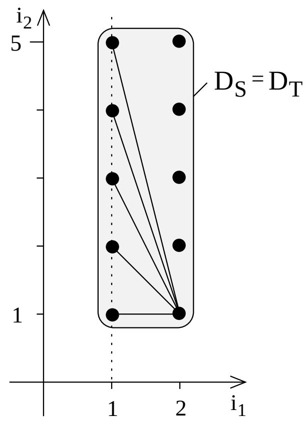


Abbildung 6.10: Beispiel zum Generieren von h-Transformationen

Um nur Punkte aus D_S aufzuzählen, ist eine Beschränkung von λ erforderlich. Hierauf wird später in diesem Abschnitt und ausführlich in Abschnitt 6.2.3 eingegangen.

Zur Beschreibung der Abhängigkeiten durch das Konzept der h-Transformation muß die soeben bestimmte Urbildmenge in einzelne Punkte zerlegt werden, weil eine Maximierung der möglichen Quellen wie in (5.1.0.2) beschrieben zu diesem Zeitpunkt nicht durchgeführt wird. Sie erfolgt später und stellt die in Abschnitt 6.2.7 aufgezeigte Möglichkeit zur Optimierung dar.

Bei der Zerlegung der Urbildmenge geht aus jedem Paar, bestehend aus einem Element des Zielraumes und einem Punkt der Urbildmenge eine h-Transformation hervor. Bezogen auf das Beispiel dieses Abschnittes ergeben sich die folgenden Transformationen:

$$\begin{aligned}
h_1 &: \{2\} \times \{1\} \rightarrow \{1\} \times \{1\} : (i_1, i_2) \rightarrow (1, 1) \\
h_2 &: \{2\} \times \{1\} \rightarrow \{1\} \times \{2\} : (i_1, i_2) \rightarrow (1, 2) \\
&\vdots \\
h_5 &: \{2\} \times \{1\} \rightarrow \{1\} \times \{5\} : (i_1, i_2) \rightarrow (1, 5)
\end{aligned}$$

Durch Zerlegung der Urbildmenge findet sich als Lösung also eine Schar von fünf h-Transformationen.

Das führt zu der Frage einer allgemeinen Darstellbarkeit solcher Lösungen, weil die Aufzählung aller Transformationen (wie oben exemplarisch durchgeführt) offensichtlich nicht praktikabel ist. Im Falle einer Parametrisierung (man ersetze die obere Grenze von i_2 durch den Strukturparameter n) ist eine solche Aufzählung sogar unmöglich.

Jedoch legt (6.2.0.12) bereits eine Möglichkeit nahe, wie eine Schar von h-Transformationen in allgemeiner Form beschrieben werden kann. Grundlegende Idee hierbei ist die Einbeziehung der in (6.2.0.12) eingeführten Größe λ. Im konkreten Fall kann die Schar $H = h_1, ..., h_5$ wie folgt ausgedrückt werden:

$$H(\lambda) : \{2\} \times \{1\} \rightarrow \{1\} \times \{\lambda\} : (i_1, i_2) \rightarrow (1, 0) + \lambda(0, 1) \tag{6.2.0.13}$$

Offensichtlich bedarf es in diesem Beispiel der folgenden Beschränkung für λ:

$$1 \leq \lambda \leq 5, \; \lambda \in \mathbb{Z} \tag{6.2.0.14}$$

Die Größe λ dient hier zur Parametrisierung von H. Jedoch ist hervorzuheben, daß es sich hierbei nicht um einen Strukturparameter im Sinne von Bedingung 4 (2) handelt, da der Wert von λ in den Grenzen von (6.2.0.14) variieren kann. Folglich besitzt λ sowohl einige Eigenschaften einer Variablen, aber auch solche eines Parameters und ist somit zunächst keiner dieser beiden Gruppen zuzurechnen.

An dieser Stelle wird nun die Festlegung getroffen, daß die Größe λ den Parametern zugeordnet wird. Zur Abgrenzung von Strukturparametern, sowie zur Unterstreichung der besonderen Rolle dieser Größe wird für sie die Bezeichnung des *Pseudoparameters* eingeführt.

Dieses Konzept des Pseudoparameters ermöglicht die Erzeugung der h-Transformation im allgemeinen Fall. Hierzu werden im folgenden zwei voneinander unabhängige Möglichkeiten vorgestellt.

6.2.2.1 Rückwärtssubstitution

Für das Beispielprogramm aus Abbildung 6.10 ergibt sich:

$$U = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

und hieraus folgt

$$t_1 = 0, \qquad (i_1,\, i_2,\, j_1,\, j_2) = \underbrace{(t_2,\, t_3,\, t_4)}_{=:\ \vec{t_0}} \left(\begin{array}{cc|cc} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ \underbrace{0 \quad 0}_{=:\ U_i} & & \underbrace{1 \quad 1}_{=:\ U_j} & \end{array} \right).$$

Als Grenzen für $\vec{t_0}$ finden sich:

$$t_2 = 1, \qquad t_3 \in \{1,\, 2,\, 3,\, 4,\, 5\}, \qquad t_4 \in [t_2 + 1,\, 2] \tag{6.2.0.15}$$

Damit lassen sich die zueinander in Konflikt stehenden Operationen schreiben als:

$$<S,\, (t_2,\, t_3)> \qquad <T,\, (t_4,\, t_4 - t_2)>$$

Folglich muß die zu generierende h-Transformation die Iterationsvektoren des Statements T auf die von S abbilden.

Es ist also:

$$(i_1,\, i_2) - \vec{t_0} = 0 = -\,(j_1,\, j_2) + \vec{t_0}.$$

Hieraus ergibt sich als Matrix eines inhomogenen Gleichungssystems:

i_1	i_2	j_1	j_2	t_2	t_3	t_4	*offset*
1	0	0	0	−1	0	0	0
0	1	0	0	0	−1	0	0
0	0	−1	0	0	0	1	0
0	0	0	−1	−1	0	1	0

Jetzt liefert eine Rückwärtssubstitution für $\vec{i}$ schrittweise:

i_1	i_2	j_1	j_2	t_2	t_3	t_4	*offset*
1	0	0	1	−1	0	−1	0
0	1	0	0	0	−1	0	0
0	0	−1	0	0	0	1	0

und

1	0	−1	1	0	0	0	0
0	1	0	0	0	−1	0	0

Damit ist:

$$\begin{aligned} i_1 &= j_1 - j_2 \\ i_2 &= t_3 \end{aligned}$$

Das Auftreten von Komponenten des Vektors $\vec{t_0}$, die in den Grenzen von (6.2.0.15) frei wählbar sind, stellt den Kern der Abbildung φ (vgl. (6.2.0.12)) dar. Dieser liegt, wie ebenfalls aus (6.2.0.12) bzw. Abbildung 6.10 zu ersehen ist, im Beispiel parallel zur i_2-Achse.

Obiges Ergebnis läßt sich somit schreiben als

$$(i_1,\, i_2) = h(j_1,\, j_2) = (j_1 - j_2,\, 0) + (0,\, t_3),$$

wobei t_3 die Rolle von λ übernimmt. Man erhält auf diese Weise in Analogie zu (6.2.0.13):

$$H(t_3)(j_1,\, j_2) = (j_1 - j_2,\, 0) + t_3\,(0,\, 1). \tag{6.2.0.16}$$

Formal:

Setzt man in Gleichung (6.1.1.5) (bzw. (6.2.0.7)) für die Komponenten des Vektors $\vec{t}$, die sich zuvor in (6.1.1.4) (bzw. (6.2.0.6)) als konstant herausgestellt haben, ihre jeweiligen Werte ein, so erhält man:

$$\left(\vec{i} + \vec{c_i},\, \vec{j} + \vec{c_j}\right) = \vec{t_0}\,(U_i,\, U_j) \tag{6.2.0.17}$$

Die Vektoren $\vec{c_i}$ und $\vec{c_j}$ stellen also eine konstante Verschiebung von $\vec{i}$ und $\vec{j}$ dar.

Damit ergibt sich das folgende inhomogene Gleichungssystem:

$$\begin{aligned} \vec{i} \;-\; \phantom{\vec{j} +{}} \vec{t_0}\,U_i &= -\vec{c_i} \\ -\,\vec{j} + \vec{t_0}\,U_j &= \vec{c_j} \end{aligned}$$

Durch Rückwärtssubstitutionerhält man für $\vec{i}$ eine Lösung, die sich schreiben läßt als:

$$\vec{i} = M_j \left(\vec{j}, 1\right)^{\top} + M_{t_0} \left(\vec{t_0}\right)^{\top}$$

Hierbei enthält M_j die Koeffizienten von $\vec{j}$, sowie den konstanten Anteil der affin linearen Abbildung. Die Menge B aller nicht-trivialen Spaltenvektoren von M_{t_0} stellt eine Basis des Kerns von φ dar.

Als Lösung ergibt sich eine Schar H von h-Transformationen:

$$H(\lambda) : \vec{j} \to M_j \left(\vec{j}, 1\right)^{\top} + B\,\lambda, \quad \lambda \in \mathbb{K}^{Rg(M_{t_0})}$$

In Abschnitt 6.2.3 wird sich die Darstellung von H in Abhängigkeit von Komponenten des Vektors $\vec{t_0}$ als nützlich erweisen. Bezeichne hierzu $M_{t_0'}$ die Matrix, die aus M_{t_0} durch Streichen aller Nullspalten hervorgeht. Durch Streichung derselben Spalten in $\vec{t_0}$ erhält man dann $\vec{t_0'}$.

Es gilt:

$$H\left(\vec{t_0'}\right)\left(\vec{j}\right) = M_j \left(\vec{j}, 1\right)^{\top} + M_{t_0'} \left(\vec{t_0'}\right)^{\top} \tag{6.2.0.18}$$

6.2.2.2 Pseudoinversion

Wie im Abschnitt zuvor gilt für das Beispiel aus Abbildung 6.10:

$$(i_1,\, i_2) = \underbrace{(t_2,\, t_3,\, t_4)}_{=:\ \vec{t_0}} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}}_{=:\ U_i} \quad \text{und} \quad (j_1,\, j_2) = \vec{t_0} \underbrace{\begin{pmatrix} 0 & -1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}}_{=:\ U_j} \tag{6.2.0.19}$$

Die auf diese Weise beschriebenen Mengen von Iterationsvektoren $(i_1,\, i_2)$ bzw. $(j_1,\, j_2)$ werden mit $\mathcal{I}$ bzw. $\mathcal{J}$ bezeichnet. Sie lassen sich gemäß Abschnitt 2.4 aus $(i_1,\, i_2) - \vec{t_0}\, U_i = 0$ bzw. $(j_1,\, j_2) - \vec{t_0}\, U_j = 0$ und $\vec{t_0} \in L$ durch Zielfunktionsminimierung und -maximierung gewinnen. Hierbei ist L die auf Seite 55 definierte Lösung von (6.1.1.7).

So ergibt sich beispielsweise für $(i_1,\, i_2)$ aus dem Ungleichungssystem

$$\begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 5 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ t_2 \\ t_3 \\ 1 \end{pmatrix} \geq 0,$$

durch Anwendung eines der unter Abschnitt 2.3 genannten Algorithmen:

$$\mathcal{I} = \{(i_1,\, i_2) \,|\, i_1 = 1,\, 1 \leq i_2 \leq 5\} \tag{6.2.0.20}$$

In völliger Analogie hierzu erhält man:

$$\mathcal{J} = \{(2,\, 1)\} \tag{6.2.0.21}$$

Zu U_i und U_j existieren pseudoinverse Matrizen

$$U_i{}^{\#} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{und} \quad U_j{}^{\#} = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix},$$

und es gilt

$$\begin{aligned} (i_1,\, i_2)\, U_i{}^{\#}\, U_i &= (i_1,\, i_2) = \vec{t_0}\, U_i \\ (j_1,\, j_2)\, U_j{}^{\#}\, U_j &= (j_1,\, j_2) = \vec{t_0}\, U_j. \end{aligned}$$

Hieraus ergibt sich:

$$\begin{aligned} (i_1,\, i_2)\, U_i{}^{\#} &= \vec{t_0} \\ (j_1,\, j_2)\, U_j{}^{\#} &= \vec{t_0} \end{aligned}$$

Zusammen mit (6.2.0.19) erhält man:

$$\begin{aligned} (i_1,\, i_2) &= (j_1,\, j_2)\, U_j{}^{\#}\, U_i \\ (j_1,\, j_2) &= (i_1,\, i_2)\, U_i{}^{\#}\, U_j \end{aligned}$$

Damit lassen sich die folgenden nicht injektiven Abbildungen auf $\mathcal{I}$ und $\mathcal{J}$ definieren:

$$\begin{aligned} \varphi_1 &: (i_1,\, i_2) \to (i_1,\, i_2)\, U_i{}^{\#}\, U_j \\ \varphi_2 &: (j_1,\, j_2) \to (j_1,\, j_2)\, U_j{}^{\#}\, U_i \end{aligned}$$

Für das Beispiel gilt:

$$\begin{aligned} \varphi_1(i_1,\, i_2) &= (i_1,\, i_2) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} = (0,\, -i_1) \\ \varphi_2(j_1,\, j_2) &= (j_1,\, j_2) \begin{pmatrix} 1 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} = (j_1 - j_2,\, 0) \end{aligned}$$

In Fortführung der Überlegungen zu (6.2.0.12) läßt sich φ^{-1} darstellen als:

$$\varphi^{-1}(j_1\, j_2) = \varphi_2(j_1\, j_2) + \lambda \ker(\varphi_1)\,, \quad \lambda \in \mathbb{Q}^{\dim(\ker(\varphi_1))}$$

Für das Beispiel heißt das:

$$\varphi^{-1}(j_1\, j_2) = (j_1 - j_2,\, 0) + \lambda\, (0,\, 1)\,, \quad \lambda \in \mathbb{Q}$$

Daraus ergibt sich übereinstimmend mit (6.2.0.16) für die gesuchte Schar H von h-Transformationen:

$$H\,(\lambda)\,(j_1\, j_2) = (j_1 - j_2,\, 0) + \lambda\, (0,\, 1)\,, \quad \lambda \in \mathbb{Q}$$

Formal:

Wie schon bei der formalen Darstellung im vorangegangenen Abschnitt (vgl. (6.2.0.17)) gilt nach Einsetzen der festen Komponenten von $\vec{t}$ in (6.1.1.5):

$$\left(\vec{i} + \vec{c_i}\right) = \vec{t_0}\, U_i, \quad \left(\vec{j} + \vec{c_j}\right) = \vec{t_0}\, U_j \tag{6.2.0.22}$$

Aus U_i, U_j sowie der in Schritt 2 ermittelten Lösungsmenge L ergeben sich durch Funktionsminimierung und -maximierung die Mengen $\mathcal{I}$ und $\mathcal{J}$ aller Iterationsvektoren der Statements S und T, für die ein Speicherkonflikt vorliegt.

Sind die zu U_i und U_j pseudoinversen Matrizen ${U_i}^{\#}$ und ${U_j}^{\#}$ bestimmt, so sind zunächst zwei Fälle zu unterscheiden:

Fall 1: ${U_i}^{\#}$ (bzw. ${U_j}^{\#}$) ist Linksinverse. Dann gilt für eine Einheitsmatrix E geeigneter Größe:

$${U_i}^{\#}\, U_i = E$$

Zusammen mit (6.2.0.22) folgt:

$$\vec{t_0}\, U_i = \left(\vec{i} + \vec{c_i}\right) = \left(\vec{i} + \vec{c_i}\right)\, E = \left(\vec{i} + \vec{c_i}\right)\, {U_i}^{\#}\, U_i$$

Es ergibt sich:

$$\vec{t_0} = \left(\vec{i} + \vec{c_i}\right)\, {U_i}^{\#}$$

Analog erhält man $\vec{t_0} = \left(\vec{j} + \vec{c_j}\right)\, {U_j}^{\#}$.

Fall 2: ${U_i}^{\#}$ (bzw. ${U_j}^{\#}$) ist Rechtsinverse. In diesem Fall folgt aus (6.2.0.22) unmittelbar:

$$\vec{t_0} = \left(\vec{i} + \vec{c_i}\right)\, {U_i}^{\#}, \quad \vec{t_0} = \left(\vec{j} + \vec{c_j}\right)\, {U_j}^{\#}$$

Unabhängig von der Frage, ob $U_i^{\#}$ und $U_j^{\#}$ links- oder rechtsinvers sind, ergibt sich derselbe Zusammenhang.

Damit können die Abbildungen φ_1 und φ_2 auf $\mathcal{I}$ und $\mathcal{J}$ wie folgt definiert werden:

$$\begin{aligned}
\varphi_1 \,:\, \mathcal{I} \to \mathbb{K}^{\dim(D_T)} &: \vec{i} \to \left(\vec{i} + \vec{c_i}\right) U_i^{\#} U_j - \vec{c_j} \\
\varphi_2 \,:\, \mathcal{J} \to \mathbb{K}^{\dim(D_S)} &: \vec{j} \to \left(\vec{j} + \vec{c_j}\right) U_j^{\#} U_i - \vec{c_i}
\end{aligned}$$

Auf diese Weise findet sich eine Schar F von Funktionen:

$$F(\lambda) : \mathcal{J} \to \mathbb{K}^{\dim(D_S)} \,:\, \vec{j} \to \varphi_2\left(\vec{j}\right) + \lambda \ker(\varphi_1), \qquad \lambda \in \mathbb{K}^{\dim(\ker(\varphi_1))}$$

Für geeignete λ nimmt F den Wertebereich $\mathcal{I}$ an und ist dann identisch mit der gesuchten Schar von h-Transformationen. Dieser Sachverhalt wird durch die folgende Bemerkung veranschaulicht.

6.2.2.3 Bemerkung (Wahl von λ)

Die besondere Bedeutung, die der Wahl geeigneter Werte für λ zukommt, hat ihre Ursache in der Definition von F. Der Wertebereich dieser Schar von Funktionen ist im allgemeinen nicht $\mathcal{I}$. Wichtig hierbei ist, daß nicht nur Elemente des von $\mathcal{I}$ aufgespannten Raumes aufgezählt werden können, sondern möglicherweise Punkte eines höherdimensionalen Raumes.

Zur Veranschaulichung werden die beiden bereits behandelten Programme aus Abbildung 6.1 und 6.10 gegenübergestellt.

Für das Beispiel aus Abbildung 6.10 wurden die Mengen $\mathcal{I}$ und $\mathcal{J}$ in (6.2.0.20) und (6.2.0.21) bestimmt. Durch den zur i_2-Achse parallel liegenden Kern werden Punkte des von $\mathcal{I}$ aufgespannten Unterraums aufgezählt.

Hingegen stellt der Kern im Beispiel 6.1.4 das orthogonale Komplement dieses Unterraumes zu $\mathbb{Q}^2$ dar. Um das zu erkennen, werden zunächst $\mathcal{I}$ und $\mathcal{J}$ sowie φ_1 und φ_2 berechnet.

Die Mengen $\mathcal{I}$ und $\mathcal{J}$ ergeben sich aus den Gleichungen (6.1.4.3), (6.1.4.4) und (6.1.4.5) durch Funktionsminimierung und -maximierung:

$$\begin{aligned}
\mathcal{I} &= \left\{(i_1,\, i_2) \,|\, 23 \le i_1 \le 63,\, i_2 = \frac{1}{4}(i_1 + 21)\right\} \\
\mathcal{J} &= \left\{(j_1,\, j_2) \,|\, 45 \le i_1 \le 125,\, j_2 = \frac{1}{4}(-j_1 + 165)\right\} \qquad (6.2.0.23)
\end{aligned}$$

Zu den Matrizen

$$U_i = (4,\, 1)\,, \quad U_j = (-8,\, 2)$$

werden ihre Pseudoinversen bestimmt:

$$U_i{}^{\#} = \frac{1}{17}\begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad U_j{}^{\#} = \frac{1}{68}\begin{pmatrix} -8 \\ 2 \end{pmatrix}.$$

Damit lassen sich φ_1 und φ_2 wie folgt definieren:

$$\begin{aligned}
\varphi_1\,(i_1,\, i_2) &= ((i_1,\, i_2) + (21,\, 0))\,\frac{1}{17}\begin{pmatrix} 4 \\ 1 \end{pmatrix}\begin{pmatrix} -8 & 2 \end{pmatrix} + (213,\, -12) \\
&= \frac{1}{17}\left(((i_1,\, i_2)\; + (21,\, 0))\begin{pmatrix} -32 & 8 \\ -8 & 21 \end{pmatrix} + (3621,\, -204)\right) \\
&= \frac{1}{17}\,(-32i_1 - 8i_2 + 2949,\, 8i_1 + 2i_2 - 36)
\end{aligned}$$

$$\begin{aligned}
\varphi_2\,(j_1,\, j_2) &= ((j_1,\, j_2) + (-213,\, 12))\,\frac{1}{68}\begin{pmatrix} -8 \\ 2 \end{pmatrix}\begin{pmatrix} 4 & 1 \end{pmatrix} - (21,\, 0) \\
&= \frac{1}{68}\left(((j_1,\, j_2)\; + (-213,\, 12))\begin{pmatrix} -32 & -8 \\ 8 & 21 \end{pmatrix} - (1428,\, 0)\right) \\
&= \frac{1}{68}\,(-32j_1 + 8j_2 + 5484,\, -8j_1 + 2i_2 + 1728) \qquad (6.2.0.24)
\end{aligned}$$

Der Kern von φ_1, der sich zu $\mathbb{Q}\,(1,\, -4)$ berechnet, steht orthogonal auf dem von $\mathcal{I}$ aufgespannten (affinen) Unterraum. Eine grafische Darstellung gibt Abbildung 6.11.

Hieraus ergibt sich die Funktionenschar F:

$$F(\lambda)\;:\;\mathcal{J} \to \mathbb{Q}^2\;:\;\vec{j} \to \varphi_2\left(\vec{j}\right) + \lambda(1, -4), \quad \lambda \in \mathbb{Q}.$$

Ein Element aus $\mathcal{J}$ wird durch $F(\lambda)$ genau dann auf $\mathcal{I}$ abgebildet, wenn λ verschwindet. Es gilt:

$$\mathcal{I} = F\,(0)\,(\mathcal{J})\,. \qquad (6.2.0.25)$$

Hieraus ergibt sich die aus einer einzigen Transformation bestehende „Schar“ H:

$$H\,(0)\left(\vec{j}\right) = \varphi_2\left(\vec{j}\right)$$

Die Berechnung zulässiger Werte für λ wird ausführlich im nächsten Abschnitt vorgestellt.

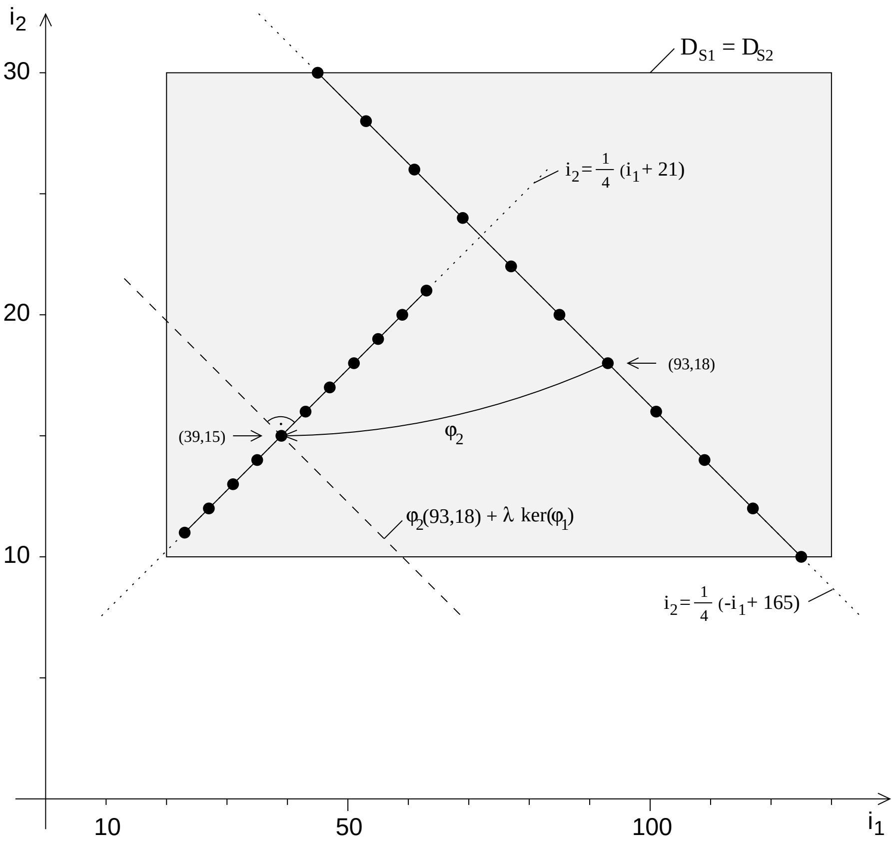


Abbildung 6.11: Kern von φ_1 als orthogonales Komplement

6.2.2.4 Vergleich

In den Abschnitten 6.2.2.1 und 6.2.2.2 sind zwei Verfahren zum Generieren der h-Transformation vorgestellt worden.

Beide Verfahren liefern letztendlich gleichwertige Ergebnisse und sind deshalb formal äquivalent.

Für die praktische Anwendung ist jedoch aus mehreren Gründen die Methode der Rückwärtssubstitution vorzuziehen. Zum einen handelt es sich um ein einfaches (und daher schnell zu berechnendes) Verfahren. Zum anderen erhält man als Ergebnis sofort eine zwischen $\mathcal{I}$ und $\mathcal{J}$ definierte Transformationsschar. Eine Einschränkung des Wertebereiches wie bei der Pseudoinversion entfällt. Ein weiterer Vorteil der Rückwärtssubstitution wird im folgenden Abschnitt (6.2.3, Fall 1) deutlich und besteht darin, daß sich die Berechnung der Grenzen für die Pseudoparameter erübrigt. Letztlich sind die Ergebnisse der Rückwärtssubstitution in vielen Fällen leichter verständlich als die der Pseudoinversion. Der Grund ist in der Matrixinvertierung selbst zu sehen, da hierbei im Falle nicht unimodularer Matrizen zwangsläufig Brüche entstehen. Infolge der zweimaligen Matrixinvertierung (Berechnung von φ_1 und φ_2) können dann beim Ausmultiplizieren im Endergebnis hohe Werte in Zähler und Nenner auftreten, sodaß oftmals weniger anschauliche Transformationsscharen entstehen.

Ein Beispiel gibt das Programm aus Abbildung 6.1. Im Vergleich zum Ergebnis der Pseudoinversion in (6.2.0.24) liefert die Rückwärtssubstitution eine intuitiv leicht zugängliche Funktion:

$$h \,:\, (j_1,\, j_2) \to \left(2j_2 + 3,\, \frac{1}{2}j_2 + 6\right)$$

6.2.3 Kontext

In diesem Abschnitt werden die Grenzen der zuvor eingeführten Pseudoparameter ermittelt. Dabei läßt sich die Menge aller zulässigen Werte als Lösungsmenge eines Ungleichungssystems beschreiben.

Wie in der vorangegangenen Bemerkung gezeigt wurde, kommt dieser Berechnung eine hohe Bedeutung zu, weil nur bei der Wahl bestimmter Werte für die Pseudoparameter die auf $\mathcal{J}$ definierten Transformationen den Wertebereich $\mathcal{I}$ annehmen. Folglich muß eine vollständige Beschreibung einer Abhängigkeit künftig immer auch die Menge zulässiger Parameterwerte beinhalten. Durch die Beschränkung auf diese Menge werden Nebenbedingungen, im folgenden *Kontext* genannt, für die Gültigkeit der Abhängigkeit aufgestellt. Die Menge selbst wird mit $\mathcal{C}$ bezeichnet werden.

Für das Beispiel 6.1.4, das in Bemerkung 6.2.2.3 aufgegriffen und weiter ausgeführt wurde, läßt sich der Kontext wie folgt berechnen:
Wegen $\vec{i} = H(\lambda)\left(\vec{j}\right)$ und $\vec{i} \in \mathcal{I}$ muß gelten:

$$H(\lambda)\left(\vec{j}\right) \in \mathcal{I}, \qquad \vec{j} \in \mathcal{J}$$

Mit den Ergebnissen aus (6.2.0.23) läßt sich das nachstehende (Un-)Gleichungssystem aufstellen:

$$\begin{array}{rcccl}
23 & \leq & i_1 = \frac{1}{68}(-32j_1 + 8j_2 + 5484 + 68\lambda) & \leq & 63 \\
 & & i_2 & = & \frac{i_1+21}{4} \\
45 & \leq & j_1 & \leq & 125 \\
 & & j_2 & = & \frac{-j_1+165}{4}
\end{array}$$

Eine Zusammenfassung liefert:

$$\frac{1}{68}\begin{pmatrix} 68 & -32 & 8 & 3920 \\ -68 & 32 & -8 & -1200 \\ -1156 & 0 & 0 & 0 \\ 1156 & 0 & 0 & 0 \\ 0 & 68 & 0 & -3060 \\ 0 & -68 & 0 & 8500 \\ 0 & 17 & 68 & -2805 \\ 0 & -17 & -68 & 2805 \end{pmatrix} \begin{pmatrix} \lambda \\ j_1 \\ j_2 \\ 1 \end{pmatrix} \geq 0$$

Offensichtlich ist dieses System für λ nur trivial lösbar. Hierdurch wird (6.2.0.25) nachträglich bestätigt.

Formal:

Grundsätzlich sind zwei Fälle zu unterscheiden.

Fall 1: Es wurde durch die Methode der Rückwärtssubstitution in Abschnitt 6.2.2.1 eine Schar H von h-Transformationen erzeugt.
Wird H wie in (6.2.0.18) in Abhängigkeit von Komponenten des Vektors $\vec{t}$ dargestellt, so stellen die in Schritt 2 bestimmten Grenzen dieser Komponenten den Kontext dar.

Fall 2: Durch die Methode der Pseudoinversion in Abschnitt 6.2.2.2 sind die Mengen $\mathcal{I}$ und $\mathcal{J}$ sowie eine Funktionsschar F berechnet worden.
Die Elemente von $\mathcal{I}$ (bzw. $\mathcal{J}$) lassen sich in gewohnter Weise als Lösung eines Ungleichungssystems darstellen:

$$\vec{i}\, M_{\mathcal{I}} \geq 0 \quad \text{bzw.} \quad \vec{j}\, M_{\mathcal{J}} \geq 0 \qquad (6.2.0.26)$$

Soll $F(\lambda)$ den Wertebereich $\mathcal{I}$ annehmen, so muß gelten:

$$F(\lambda)\left(\vec{j}\right) \in \mathcal{I}$$

Zusammen mit (6.2.0.26) folgt

$$\vec{j}\;(M_F\,M_{\mathcal{I}}) \geq 0,$$

wobei M_F diejenige Matrix ist, die F vermittelt.

Nimmt man $\vec{j} \in \mathcal{J}$ an, so erhält man schließlich das folgende Ungleichungssystem:

$$\vec{j}\begin{pmatrix} M_F\,M_{\mathcal{I}} \\ M_{\mathcal{J}} \end{pmatrix} \geq 0$$

Hieraus ergibt sich der Kontext durch Funktionsminimierung und -maximierung mit Zielfunktion λ.

6.2.4 Richtungs- und Distanzvektor

Nachdem die h-Transformationen berechnet worden sind, lassen sich die Richtungs- und Distanzvektoren auf einfache Weise bestimmen. Im Gegensatz zu Abschnitt 5.2.5 (bzw. Bemerkung 3.3.19) ist es jedoch möglich, den Distanzvektor mit einer einzigen Projektion zu berechnen. Der Grund hierfür ist die zur Behandelbarkeit von imperfekt verschachtelten Programmen durchgeführte Dimensionsangleichung der Iterationsräume in Abschnitt 6.2.1.2.

Für ein $\lambda \in \mathcal{C}$ gilt:

$$\begin{aligned} \vec{d}\left(\vec{j}\right) &= \pi_{<1,\ldots,N_{ST}>}\left(\vec{j}-\vec{i}\right) \\ &= \pi_{<1,\ldots,N_{ST}>}\left(\vec{j}-H(\lambda)\left(\vec{j}\right)\right) \\ &= \pi_{<1,\ldots,N_{ST}>}\left(id-H(\lambda)\right)\left(\vec{j}\right) \end{aligned}$$

Der Richtungsvektor ergibt sich dann durch Anwendung von Definition 3.3.18 auf $\vec{d}\left(\vec{j}\right)$.

6.2.5 Raumzerlegung

Da die Richtung der Abhängigkeit bislang noch nicht berücksichtigt wurde, kann das Auftreten lexikografisch negativer Distanzvektoren derzeit nicht ausgeschlossen werden.

Zur Veranschaulichung dieses Problems sowie dessen Lösung wird an dieser Stelle das Beispiel 6.2.1.3 fortgeführt. Wie dort zu Schritt 3 bereits angemerkt, existieren Abhängigkeiten in beiden Richtungen.

Betrachtet man die Menge aller Operationen $< S, (i_1, 2) >$, durch die $A[2]$ referenziert wird, so erkennt man, daß zwei dieser Operationen ($1 \leq i_1 \leq 2$) vor dem Zugriff von T auf $A[2]$ erfolgen, wohingegen die anderen beiden Operationen erst nach T in diese Speicherstelle schreiben.

Also gilt:

$$\begin{array}{rcll} <S, (i_1, 2)> & \delta^o & <T, (2, [\alpha_2])> & \quad 1 \leq i_1 \leq 2 \\ <T, (2, [\alpha_2])> & \delta^o & <S, (i_1, 2)> & \quad 3 \leq i_1 \leq 4 \end{array}$$

Bezieht man die übrigen Elemente des Arrays A in diese Überlegung ein, so wird ersichtlich, daß sich die Richtung der Abhängigkeit für den Zugriff auf $A[k]$ nach der k-ten Iteration der inneren Schleife umkehrt.

Ziel muß es also sein, den Definitionsbereich von $H(\lambda)$ in der Weise zu beschränken, daß anschließend in Übereinstimmung mit Korollar 3.3.14 nur noch lexikografisch nicht-negative Distanzvektoren auftreten.

Wesentliches Kriterium ist also die lexikografische Ordnung, wodurch sich ein Lösungsweg bereits abzeichnet.

Für das genannte Beispiel, zu dem sich im Abschnitt 6.2.3 der Kontext

$$\mathcal{C} = \{\lambda \,|\, 1 \leq \lambda \leq 4\}$$

sowie in Abschnitt 6.2.4 der Distanzvektor

$$\vec{d}\left(\vec{j}\right) = (j_1 - \lambda)$$

ergeben haben, ist die Menge aller $\vec{j} \in \mathcal{J}$ zu bestimmen, so daß $0 \underline{\ll} \vec{d}$ gilt. Offensichtlich ist das genau dann der Fall, wenn die Bedingung $j_1 \geq \lambda$ erfüllt ist. Diese Bedingung ist die mathematische Formulierung der obigen Beobachtung, derzufolge die Abhängigkeit für einen Zugriff auf $A[k]$ bis einschließlich zur k-ten Iteration der inneren Schleife von S anch T gerichtet ist. Für

#	Quelle/Ziel/Art	H	Kontext	Def.-bereich von H	$\vec{d}$
I	$S\delta^o T$	$(\lambda,\, j_1)$	$1 \leq \lambda \leq 4$	$\lambda \leq j_1 \leq 4$	$(j_1 - \lambda)$
II	$T\delta^o S$	(i_2)	-	$2 \leq i_1 \leq 4$ $1 \leq i_2 < i_1$	$(i_1 - i_2)$

Tabelle 6.2: Abhängigkeiten des Beispielprogrammes

alle Werte von j_1, die diese Bedingung nicht erfüllen, kehrt sich die Richtung der Abhängigkeit um.

Zu dem Beispiel ergeben sich die beiden Abhängigkeiten in Tabelle 6.2.

Formal:

Für das Eingabeprogramm ist die Zerlegung der Zugriffe in Äquivalenzklassen gemäß Abschnitt 5.2.1.2 des vorigen Kapitels zu berechnen und eine Bijektion ψ, die (5.2.0.9) genügt, zu wählen.

Zu den Zugriffen z_S in S und z_T in T wurden $H(\lambda)$, $\mathcal{C}$ sowie $\mathcal{J}$ bestimmt.

Um sicherzustellen, daß der Distanzvektor $\vec{d} = (d_1, ..., d_{N_{ST}})$ lexikografisch nicht negativ ist, müssen die Komponenten dieses Vektors entsprechend der Definition 3.2.1 untersucht werden.

Gilt $d_1 > 0$ für geeignete $\vec{j} \in \mathcal{J}$, so ist Korollar 3.3.14 für diese $\vec{j}$ erfüllt. Kann darüber hinaus der Fall $d_1 = 0$ eintreten, ist $\mathcal{J}$ auf die Elemente einzuschränken, die diese Gleichung eifüllen, und anschließend mit derselben Untersuchung für die folgende Komponente fortzufahren. Stellt sich schließlich heraus, daß $\vec{d} = 0$ gelten kann, so ist für diesen Fall zusätzlich die Reihenfolge der Zugriffe z_S und z_T zu beachten. In diesem Punkt kommt die in Kapitel 5 eingeführte zugriffsbasierte Ordnung zum Tragen.

Abbildung 6.12 zeigt einen Algorithmus zu der soeben beschriebenen Aufteilung von $\mathcal{J}$.

Dabei wird für die Funktion *solve*$(P,\, D,\, N) \to L$ angenommen, daß sie das Problem P

$$d_l = \pi_l(id - H)\left(\vec{j}\right) \; op \; 0, \qquad 1 \leq l \leq N_{ST},\; op \in \{>, \geq\}$$

für $\vec{j} \in D$ unter der Nebenbedingung $\lambda \in N$ löst und die Lösungsmenge L als Rückgabe liefert.

Durch *add*(P, H) werden dem Polytop P die Ungleichungen, die H darstellen, hinzugefügt und die Funktion *append*(L, l) hängt an die Liste L ein weiteres Element l an.

6.2.6 Quell- und Zielraumberechnung

Auf der Grundlage der bisherigen Ergebnisse lassen sich der Quell- und der Zielraum einer Abhängigkeit als Teilmenge der Iterationsräume berechnen.

Es gilt

$$\vec{j} \in \mathcal{J}', \quad \lambda \in \mathcal{C},$$

wobei $\mathcal{J}'$ den Definitionsbereich von $H(\lambda)$ bezeichnet, der durch Beschränkung gemäß Abschnitt 6.2.5 aus $\mathcal{J}$ hervorgeht.

Durch Funktionsminimierung und -maximierung bezüglich $\vec{j}$ ergibt sich der Zielraum ZR dieser Abhängigkeit.

Bezogen auf Abhängigkeit I in Tabelle 6.2 heißt das:

$$\begin{aligned} \vec{j} &\in \mathcal{J}' = \{j_1 \,|\, \lambda \leq j_1 \leq 4\} \\ \lambda &\in \mathcal{C} = \{\lambda \,|\, 1 \leq \lambda \leq 4\}\,. \end{aligned}$$

Daraus folgt:

$$ZR(I) = \{j_1 \,|\, 1 \leq j_1 \leq 4\}\,.$$

Für den Quellraum QR gilt allgemein:

$$\begin{aligned} \vec{i} &= H(\lambda)\left(\vec{j}\right) \\ \vec{j} &\in \mathcal{J}' \\ \lambda &\in \mathcal{C} \end{aligned}$$

Der Quellraum läßt sich dann ebenfalls durch Funktionsminimierung und -maximierung, dieses Mal mit Zielfunktion $\vec{i}$, bestimmen.

Eingabe: $H(\lambda)$, $\mathcal{C}$, $\mathcal{J}$, z_S, z_T

Ausgabe: Liste L mit Teilpolytopen $\mathcal{J}_1$, $\mathcal{J}_2$.... von $\mathcal{J}$ derart, daß für den Distanzvektor gilt:

$$0 \underline{\ll} \vec{d}\left(\vec{j}\right), \qquad \vec{j} \in \mathcal{J}_i,\ 1 \leq i \leq |Z|$$

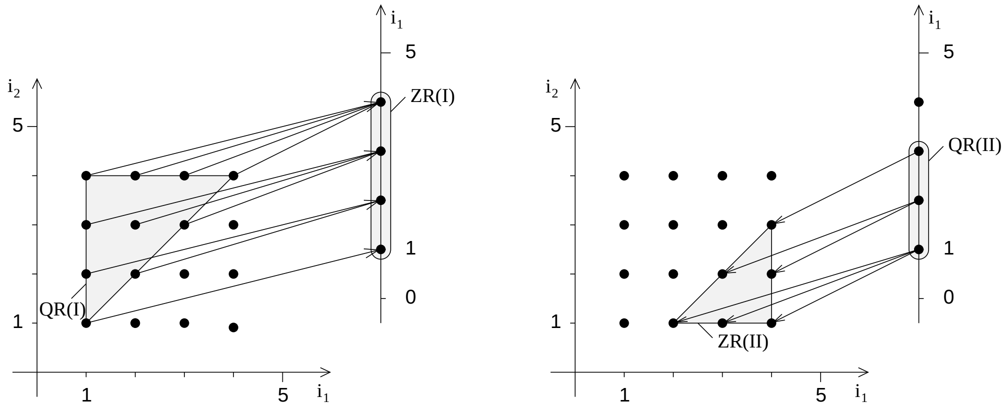
/∗ Initialisierung ∗/
L := $\perp$
done := FALSE
$\vec{d}$:= $(d_1, ..., d_{N_{ST}}) = \pi_{<1,....,N_{ST}>}(id - H(\lambda))$
level := 1
$\mathcal{J}^+$:= $\mathcal{J}$

/∗ schleifengetragene Abhängigkeiten ∗/
while ($level \leq N_{ST} \ \wedge\ not(done)$) **do**
    $T := solve(d_{level} > 0,\ \vec{j} \in \mathcal{J}^+,\ \mathcal{C})$
    if ($T \neq \emptyset$)
        $append(L,\ \mathcal{J}^+ \cap d_{level} > 0)$
    fi
    $T := solve(d_{level} = 0,\ \vec{j} \in \mathcal{J}^+,\ \mathcal{C})$
    if ($T = \emptyset$)
        done := TRUE
    else
        $add(\mathcal{J}^+,\ d_{level} = 0)$
    fi
    level :=*level* + 1
od

/∗ statementgetragene Abhängigkeiten ∗/
if ($not(done) \ \wedge\ z_S <_{acc} z_T$)
    $append(L,\ \mathcal{J}^+)$
fi

return Z

Abbildung 6.12: Algorithmus zur Aufteilung von $\mathcal{J}$



(a) Abhängigkeit I (b) Abhängigkeit II

Abbildung 6.13: Abhängigkeiten aus Beispiel 6.2.1.3

Im Beispiel folgt aus dem Ungleichungssystem

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 4 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ \lambda \\ 1 \end{pmatrix} \geq 0$$

für den Quellraum

$$QR(I) = \{(i_1,\, i_2) \,|\, 1 \leq i_1 \leq 4,\, i_1 \leq i_2 \leq 4\}$$

Auf dieselbe Weise erhält man für die andere Abhängigkeit:

$$\begin{aligned} ZR(II) &= \{(i_1,\, i_2) \,|\, 2 \leq i_1 \leq 4,\, 1 \leq i_2 < i_1\} \\ QR(II) &= \{j_1 \,|\, 1 \leq j_1 \leq 4\}\,. \end{aligned}$$

Abbildung 6.13 veranschaulicht die gewonnenen Ergebnisse.

6.2.7 Optimierung

Nachdem die Abhängigkeiten vollständig bestimmt worden sind, soll in diesem Abschnitt eine Möglichkeit zur Optimierung vorgestellt werden.

Betrachtet man nochmals Abbildung 6.10, so wird deutlich, daß $A[1]$ in diesem Beispiel durch die Operation $< S,\ (1,\ 5) >$ letztmalig vor $< T,\ (2,\ 1) >$ referenziert wird. Die Zugriffe $< S,\ (1,\ i_2) >$, $1 \leq i_2 \leq 4$, können bei der Betrachtung der Abhängigkeit vernachlässigt werden.

Wünschenswert wäre es also, den Kontext $\mathcal{C}$ einer Abhängigkeit e auf eine Teilmenge $\mathcal{C}^+ \subset \mathcal{C}$ einzuschränken, so daß sich für

$$H(\lambda)\left(\vec{j}\right), \qquad \vec{j} \in \mathcal{J},\ \lambda \in \mathcal{C}^+,$$

die jeweils letzte Iteration des Quellstatements ergibt, die zu einem Konflikt führt.

Hierdurch wird jedoch *nicht* der Datenfluß berechnet. Ein Vergleich mit Feautriers Ansatz verdeutlicht das: dort sah die erste von zwei Phasen der Auswertung (vgl. Abbildung 5.1) vor, zu einer Menge Q_{S_T} aus Iterationsvektoren das lexikografische Maximum zu bestimmen. Damit findet die hier vorgestellte Optimierung ihre direkte Entsprechung in dieser ersten Phase von Feautriers Ansatz. Jedoch werden bei Banerjee Iterationsvektoren anderer Statements als S und T nicht berücksichtigt, so daß hier eine der zweiten Phase entsprechende Optimierung nicht vorgenommen werden kann. Infolgedessen ist Direktheit der berechneten Abhängigkeiten im allgemeinen nicht zu garantieren. Um diese zu erreichen ist eine Nachbearbeitungsphase erforderlich, wie sie in Abschnitt 5.2.1.1 vorgestellt worden ist.

Formal:

Für eine Schar $H(\lambda)$ von h-Transformationen einer Abhängigkeit e von S nach T mit zugehörigem Kontext $\mathcal{C}$ soll zu einem gegebenen $\vec{j} \in \mathcal{J}$ die letzte mit $<T, \vec{j}>$ in Konflikt stehende Operation von S bestimmt werden.

Das führt zu der Berechnung des lexikografischen Maximums der Menge

$$\left\{\vec{i}\,|\,\vec{i} = H(\lambda)\left(\vec{j}\right),\ \vec{j} \in \mathcal{J},\ \lambda \in \mathcal{C}\right\} \tag{6.2.0.27}$$

Stellt man diese Menge in gewohnter Weise als Lösung eines Ungleichungssystems dar, wobei die Pseudoparameter für die Dauer dieser Berechnung als Variablen betrachtet werden sollen, so ergibt sich die gesuchte Beschränkung von $\mathcal{C}$ durch Funktionsminimierung und -maximierung bezüglich $\left(\vec{j},\ \lambda\right)$.

Für das Beispiel aus Abbildung 6.10 ergibt sich aus

$$\begin{aligned} (i_1,\, i_2) - (j_1 - j_2,\, \lambda) &= 0 \\ j_1 &= 2 \\ j_2 &= 1 \\ \lambda &\in \{1,\, 2,\, 3,\, 4,\, 5\} \end{aligned}$$

als Lösung für λ:

$$\mathcal{C}^+ = \{5\}.$$

Setzt man diesen Wert für λ ein, so läßt sich der Pseudoparameter nachträglich eliminieren und man erhält als einzige (in diesem Fall direkte) Abhängigkeit mit $H(5) = h$:

Quelle/Ziel/Art	h	Def.-bereich h	Kontext	$\vec{d}$
$S\delta^o T$	$(j_1, j_2) \rightarrow (j_1 - j_2, 5)$	$\{2\} \times \{1\}$	-	(1)

Auf gleiche Weise ergibt sich zu Abhängigkeit I von Beispiel 6.2.1.3 aus

$$\begin{aligned} (i_1,\, i_2) - (\lambda,\, j_1) &= 0 \\ j_1 &\in \{\lambda,\, \ldots.,\, 4\} \\ \lambda &\in \{1,\, 2,\, 3,\, 4\} \end{aligned}$$

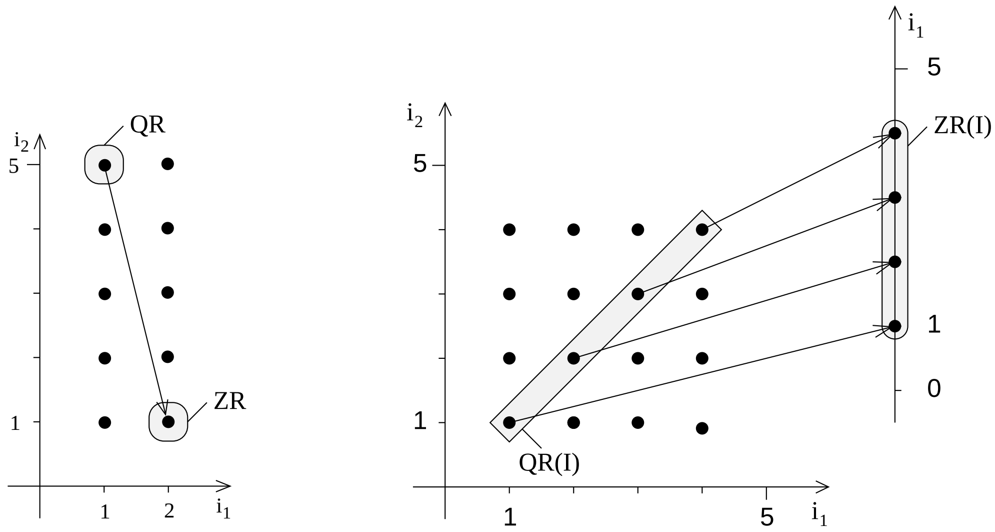
das Ungleichungssystem

$$\begin{pmatrix} 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 4 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} j_1 \\ \lambda \\ i_1 \\ i_2 \\ 1 \end{pmatrix} \geq 0$$

und hieraus

$$\mathcal{C}^+ = \{\lambda \,|\, 1 \leq \lambda \leq 4,\, \lambda = j_1\}.$$

Damit läßt sich auch in diesem Beispiel der Pseudoparameter eliminieren und Abhängigkeit I reduziert sich auf $H(j_1) = h$:



(a) Beispiel zu 6.2.2

(b) Beispiel 6.2.1.3

Abbildung 6.14: Optimierte Ergebnisse

Quelle/Ziel/Art	h	Def.-bereich h	Kontext	$\vec{d}$
$S\delta^f T$	$(j_1) \rightarrow (j_1, j_1)$	$\{1, ..., 4\}$	-	(0)

In Abbildung 6.14 werden diese Ergebnisse grafisch dargestellt.

Einerseits wird in diesem Abschnitt eine Modifikation des Kontextes durchgeführt, welcher andererseits zur Quell-/Zielraumberechnung in Abschnitt 6.2.6 benötigt wird. Folglich muß diese Optimierung – falls erwünscht – vor der Berechnung von Quell- und Zielraum in die Analyse eingegliedert werden.

6.2.8 Übersicht

Abschließend soll der Ablauf einer Datenabhängigkeitsanalyse durch den erweiterten Ansatz zusammengefaßt werden.

Zu Beginn ist eine Transformation des Eingabeprogrammes P erforderlich. Hierbei wird P mit Hilfe der in den Abschnitten 6.2.1.2, 6.2.1.4 und 6.2.1.5 genannten Verfahren in ein perfekt verschachteltes Programm P' überführt, in dem keine Skalare oder null-dimensionale Iterationsräume auftreten. Auf P' sind dann die Schritte 1 und 2 der Analyse in der Form von 6.2.1.1 anwendbar.

In Schritt 3 ist zunächst eine Methode zum Generieren der h-Transformation (6.2.2.1, 6.2.2.2) zu wählen. Um den Wertebereich der Transformationen zu beschränken, schließt sich die Berechnung des Kontextes (6.2.3) an. Daraufhin wird der Definitionsbereich der Transformationen in der Weise eingegrenzt (6.2.5), daß schließlich nur Distanzvektoren auftreten, die Korollar 3.3.14 genügen. Zu diesem Zeitpunkt kann eine Optimierung der bisherigen Ergebnisse (6.2.7) durchgeführt werden. Letztlich sind noch Quell- und Zielraum der Abhängigkeit(6.2.6) zu bestimmen.

Eine grafische Darstellung dieses Ablaufes gibt Abbildung 6.15.

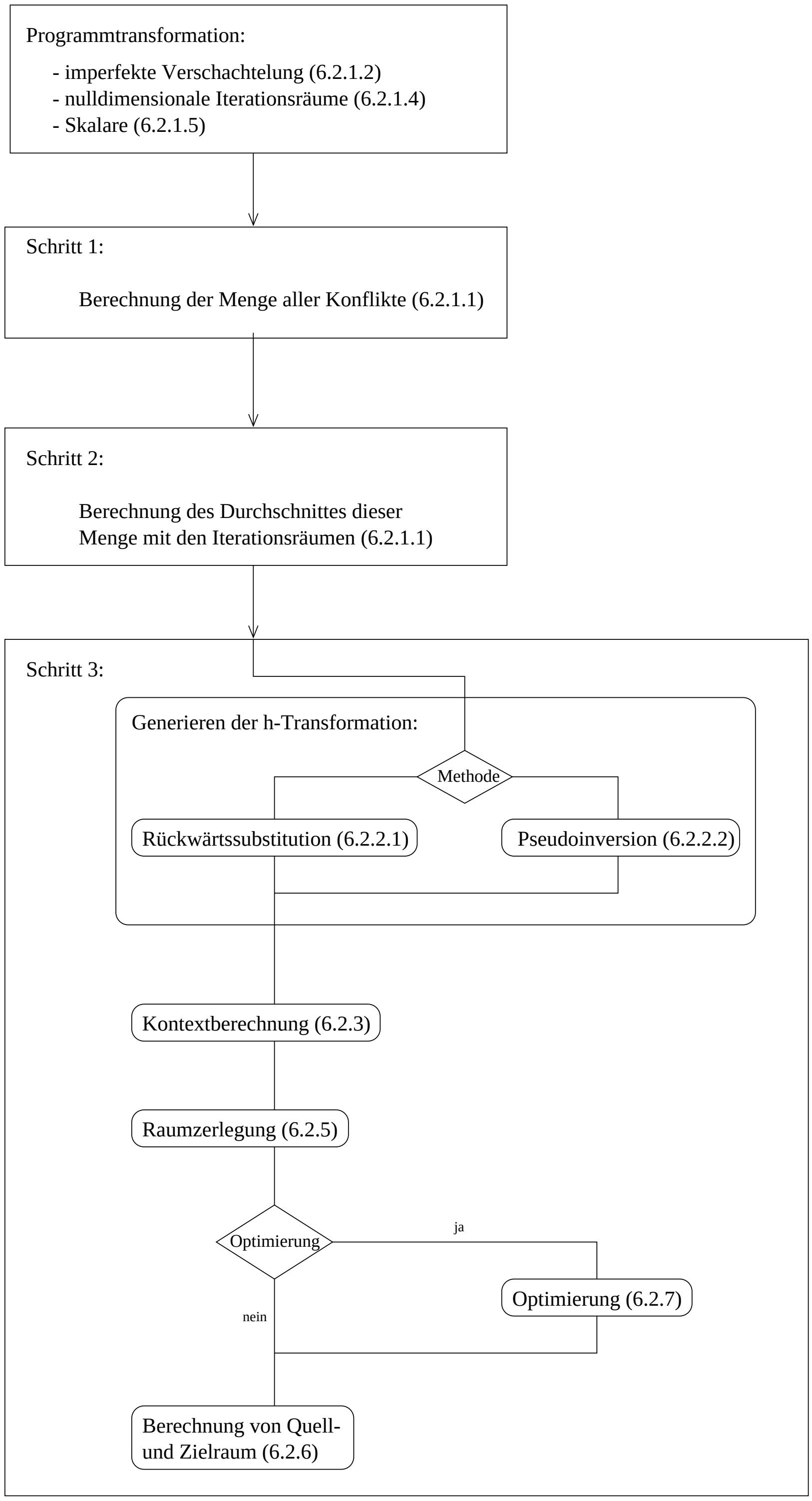


Abbildung 6.15: Ablaufplan für den erweiterten Ansatz

Kapitel 7

Gemeinsame Erweiterungsmöglichkeiten

In diesem Kapitel werden Möglichkeiten zur Erweiterung der Eingabesprache aufgezeigt, die auf beide Ansätze gleichermaßen anwendbar sind. Dabei wird sich herausstellen, daß die Mächtigkeit der in den vorigen beiden Kapiteln durchgeführte Erweiterung der Analyseverfahren eine einfache Integration der neuen Sprachelemente erlaubt.

7.1 Mehrfachindizierung, nicht-affine Indexfunktionen

Bisher waren nur solche Indexfunktionen zulässig, deren einzelne Komponenten sich entsprechend Bedingung (5) der Eingabespezifikation als affin lineare Funktionen in Variablen umgebender Schleifen und Strukturparametern darstellen lassen. Der Grund hierfür besteht zum einen darin, daß Ausdrücke, die diese Bedingung nicht erfüllen, etwa die Indexfunktion von $A[i^2]$ ungleich schwerer auszuwerten sind. Zum anderen ist im Falle einer Mehfachindizierung, etwa $A[B[i]]$, eine Auswertung zur Übersetzungszeit im allgemeinen nicht möglich.

Enthält ein Statement S einen Arrayzugriff mit Mehrfachindizierung bzw. nicht affin linearen Komponenten der Indexfunktion, so ist die exakte Bestimmung derjenigen Operationen von S, für die es zu einem Speicherkonflikt kommt, nur schwer oder überhaupt nicht möglich. Deshalb bietet sich für beide Situationen eine konservative Abschätzung der folgenden Form an.

Genügt die Komponente mindestens einer von zwei Indexfunktionen eines zu analysierenden Zugriffspaares der Bedingung 4 (5)) nicht, so wird sie in beiden Indexfunktionen durch *dieselbe* Funktion f ersetzt. Hierbei ist f affin linear in den Strukturparametern, nicht jedoch den Schleifenvariablen.

Beispielweise erhält man aus $A[i,\ B[j]]$ hierdurch $A[i,\ 0]$, wobei zur Ersetzung der zweiten Komponente für f (willkürlich) die Nullfunktion gewählt wurde.

Infolge dieser Ersetzung sind die betreffenden Komponenten der beiden Indexfunktionen für jede Iteration gleich. Man erhält auf diese Weise eine Abschätzung, die zur Übersetzungszeit die Menge aller möglichen Abhängigkeiten beinhaltet – unabhängig davon, ob diese zur Laufzeit tatsächlich existieren.

7.2 Nicht-affine Schleifengrenzen

Wie im vorigen Abschnitt dargestellt, führt die Verwendung nicht affin linearer Ausdrücke zur Übersetzungszeit zu Auswertungsproblemen. Hiervon sind neben den bereits behandelten Indexfunktionen die Schleifengrenzen (Bedingung 4 (2)) in gleicher Weise betroffen.

Erfüllt eine Schleifengrenze diese Bedingung nicht, so kann sie ersatzlos gestrichen werden. Daß infolgedessen die Iterationsräume zu den entsprechenden Seiten hin (möglicherweise) unbeschränkt sind, hat keinen Einfluß auf die Analyse. Zwar wurden bislang nur Polytope behandelt, jedoch wurde die Beschränktheit an keiner Stelle zur Voraussetzung erklärt.

Beide Analyseverfahren lassen sich unverändert auch auf den unbeschränkten Fall anwenden.

Aus den zur Laufzeit bekannten Grenzen der Schleifen ergeben sich dann auch die Schranken der Polyeder in den Analyseergebnissen.

7.3 Interprozedurale Abhängigkeiten

In diesem Abschnitt soll die Eingabesprache um Prozeduren ereweitert werden.

SUBROUTINE *sqare*(x, result)
  INTEGER, INTENT(IN) :: x
  INTEGER, INTENT(OUT) :: result

  result = x ∗ x

END SUBROUTINE *sqare*

DO i = 1, 10
    $A[i] = i$
    $sqare(A[i], A[i + 10])$
END DO

Abbildung 7.1: Beispiel zu interprozeduralen Abhängigkeiten

Innerhalb von Prozeduren auftretende Abhängigkeiten lassen sich vollständig mit den vorgestellten Ansätzen behandeln. Den kritischen Bestandteil dieses Abschnittes machen Abhängigkeiten zwischen einer Prozedur und dem aufrufenden Programmteil aus.

Interprozedurale Abhängigkeiten können zum einen dadurch enstehen, daß bei einem Prozeduraufruf lesend auf eine Speicherstelle zugeriffen wird. Zum anderen besteht aber auch die Möglichkeit, daß die Prozedur ihrerseits in diese Speicherstelle schreibt.

Ein einfaches Beispiel gibt Abbildung 7.1. Die ersten zehn Arrayelemente werden zunächst durch das Hauptprogramm initialisiert. Nach jedem Schreibzugriff erfolgt durch die Funktion *square* ein Lesezugriff. Hierdurch wird zwischen Hauptprogramm und Prozedur eine anti-Abhängigkeit erklärt. Die Prozedur schreibt das Quadrat des gelesenen Wertes in dasselbe Array zurück, jedoch kommt es hierdurch offensichtlich zu keinem Konflikt. Für die Wertübergabe wird ein call-by-reference Mechanismus vereinbart.

Ist das Ein-/AIIusgabeverhalten einer Funktion erst einmal bekannt, so kann das Hauptprogramm ohne jede Änderung der beiden Ansätze analysiert werden. Bestehende interprozedurale Abhängigkeiten werden hierbei erkannt und mitberechnet.

Nachträglich wird dadurch ebenfalls die Definition der Menge $\mathcal{Z}^w$ aller schreibenden Zugriffe eines Statements gerechtfertigt. Trivialerweise war diese Menge bislang stets einelementig. Jedoch könen bei Verwendung von Prozeduren auch mehrere Schreibzugriffe innerhalb eines Statements auftreten.

7.4 Kontrollflußstrukturen

Eine der einschneidendsten Beschränkungen in der Spezifikation der Eingabesprache ist darin zu sehen, daß bislang keine Kontrollflußstrukturen zulässig waren. Aus diesem Grund wird in diesem Abschnitt die Behandlung von *if*-Statements (bzw. bedingten Ausdrücken in C) vorgestellt.

Man kann sich leicht übelegen, daß die einfachste Möglichkeit darin besteht, diese Strukturen bei der Analyse vollständig auszublenden. Damit wird angenommen, daß die Anweisungen des *if*- und des *else*-Zweiges immer ausgeführt werden. Wie schon in Abschnitt 7.1 erhält man so eine Abschätzung der Abhängigkeiten.

Soll jedoch berücksichtigt werden, ob die Anweisungen eines Zweiges tatsächlich ausgeführt werden, so ist die Bedingung des *if*-Statements auszuwerten. Das führt jedoch unmittelbar zur Frage der Auswertbarkeit einer solchen Bedingung. Diese ist, wie bereits in 7.1 dargestellt, zur Übersetzungszeit nicht immer gegeben.

DO $i = 1,\, 5$
  IF $(i \leq 3)$ **THEN**
    $\{S\}$
  ELSE
    $\{T\}$
  END IF
END DO

Läßt sich die Bedingung wie im nebenstehenden Beispiel als eine Konjunktion von Funktionen, die in gewohnter Weise affin linear in Schleifenvariablen und Strukturparametern sind, darstellen, so kann diese Bedingung dem Iterationsraum des *if*-Statements hinzugefügt werden. Aus dem Komplement zum gesamten durch die Schleifen beschriebenen Raum ergibt sich dann der Iterationsraum des *else*-Statements. Aus dem Beispiel ergibt sich:

$$
\begin{aligned}
D_S &= \{i \,|\, 1 \leq i \leq 5 \;\wedge\; 1 \leq i \leq 3\} = \{1,\, 2,\, 3\} \\
D_T &= \{i \,|\, 1 \leq i \leq 5 \;\wedge\; i > 3\} = \{4, 5\}
\end{aligned}
$$

Als problematisch kann sich jedoch die Bildung des Komplements herausstellen. An dieser Stelle sei auf das Aufspalten von h-Transformationen (Abschnitt 5.2.3) verwiesen. Alle dortigen Ausführungen lassen sich direkt vom Konstrukt des Quasts auf das Problem dieses Abschnittes übertragen, weswegen hier auf weitere Erläuterung verzichtet wird.

Kann die Auswertung der Bedingung eines *if*-Statements nicht (vollständig) vorgenommen werden, so bleibt die Möglichkeit der Elimination von statementgetragenen Abhängigkeiten zwischen Zuweisungen des *if*- und des *else*-Zweiges. Da Zuweisungen beider Zweige nicht innerhalb derselben Iteration ausgeführt werden, können diese Abhängigkeiten nicht existieren.

7.5 Negative Schrittweiten

In Kapitel 4 wurde die Eingabesprache auf Programme beschränkt, deren Schleifen die Schrittweite $+1$ besitzen. Dieser Abschnitt beschäftigt sich mit dem Fall, daß eine Schleife rückwarts, also mit Schrittweite -1, durchlaufen wird.

In [Pug92] (und darauf aufbauend [Wol94]) sowie [Ban96] wird eine Normalisierung des Eingabeprogrammes vorgeschlagen. Eine solche Umformung ist für gebrochen rationale Schrittweiten erforderlich. Jedoch wird gezeigt, daß das Problem dieses Abschnittes allein durch die Modifikation einer bereits vorhandenen Ordnung gelöst werden kann.

DO $i_1 = 1, 5$
  DO $i_2 = 1, 5, -1$
    $A[i, j] = ...$ $\{S\}$
    $.... = A[i, j-1]$ $\{T\}$
  END DO
END DO

Anhand des nebenstehenden Programms wird deutlich, daß nicht länger der schreibende Zugriff Quelle einer true-Abhängigkeit $S\delta T$ ist, sondern Ziel einer anti-Abhängigkeit $T\delta S$. Zu einem schreibenden Zugriff $<T, (i, j)>$ gibt die Operation $<S, (i, j+1)>$ die Quelle an. Lexikografisch betrachtet ist der Iterationsvektor der Quelle also größer als der des Zieles. Es wird aber auch deutlich, daß von dieser Umkehr genau die Vektorkomponenten der Iterationsvektoren betroffen sind, deren zugehörigen Schleifen in umgekehrter Richtung durchlaufen werden.

Es wird zunächst ein Vektor $V \in \mathbb{K}^{N_{ST} \times 1}$ eingeführt, dessen i-te Komponente das Vorzeichen der Schrittweite von Schleife i angibt. (N_{ST} ist wieder die Anzahl der Schleifen, die sowohl S als auch T umgeben). V wird im folgenden als *Vorzeichenvektor* bezeichnet.

Damit läßt sich die Definition 3.2.1 der lexikografischen Ordnung wie folgt erweitern:

Seinen $\vec{i}$ und $\vec{j}$ sowie der Vorzeichenvektor V gegeben. Dann ist $\vec{i}$ bezüglich V lexikografisch kleiner oder gleich $\vec{j}$ (i. Z.: $\underline{\ll}^V$), falls

$$\vec{i}\,V^\top \;\underline{\ll}\; \vec{j}\,V^\top.$$

In ähnlicher Weise wird die zugriffsbasierte textuelle Ordnung erweitert. Mit den Bezeichnungen aus (5.2.0.10) gilt

$$z_1 <_{acc}^V z_2 \quad\Longleftrightarrow\quad \begin{cases} last(V) = \;\;\,1, & (S_1 <_{txt} S_2) \\ & \vee\,(S_1 = S_2 \wedge \psi(z_1) < \psi(z_2)) \\ last(V) = -1, & (S_2 <_{txt} S_1) \\ & \vee\,(S_1 = S_2 \wedge \psi(z_2) < \psi(z_1)) \end{cases}.$$

Hierbei ist *last* eine Funktion, die die letzte Komponente eines Vektors liefert. Für leere Vektoren wird als Rückgabewert dieser Funktion 1 vereinbart.

In Analogie zu Definition 3.2.4 bzw. (5.2.0.11) ergibt sich das zugriffsbasierte Sequenzierungsprädikat bezüglich V (i. Z.: $\prec_{acc}^V$) durch Komposition der lexikografischen sowie der ebenfalls bezüglich V zugriffsbasierten textuellen Ordnung.

Es gilt:

$$< S_1, v_1, z_1 > \prec_{acc}^V < S_2,\ v_2,\ z_2 >$$
$$\Longleftrightarrow$$
$$< S_1, v_1\,V^\top, z_1 > \prec_{acc} < S_2,\ v_2\,V^\top,\ z_2 > .$$

Die bisherigen Ordnungen ergeben sich damit als Spezialfälle dieser Erweiterung für $V = 1$.

Mit Hilfe der Erweiterung des Sequenzierungsprädikates kann das Problem dieses Abschnittes gelöst werden. Für Feautriers Ansatz genügt es, die Auswertung von (5.1.0.7), (5.1.0.8) sowie den Rekursionsregeln 2 und 3 auf Seite 33 anhand dieser neu definierten Ordnung vorzunehmen. Der Distanzvektor $\vec{d}^{\,V}$ ergibt sich dann aus der Definition von $\vec{d}$ in Abschnitt 5.2.5 durch

$$\vec{d}^{\,V}\left(\vec{j}\right) = \vec{d}\left(\vec{j}\right)\,V^\top.$$

Bei Banerjees Ansatz ist es ausreichend, diese erweiterte Definition des Distanzvektores auf 6.2.4 (und damit den Algorithmus zur Raumzerlegung in

Abbildung 6.12) zu übertragen. Weitere Änderungen sind nicht erforderlich! Man beachte, daß die Optimierung in Abschnitt 6.2.7 auch als Minimierung des Distanzvektors aufgefaßt werden kann. Das folgt unmittelbar aus (6.2.0.27), weil die dort definierte Menge genau aus den Distanzvektoren besteht. Formuliert man das Optimierungsproblem zu der Frage nach dem lexikografischen Minimum der Menge

$$\left\{\vec{d}\left(\vec{j}\right) \mid \vec{d} = (id - H(\lambda))\,,\, \vec{j} \in \mathcal{J},\, \lambda \in \mathcal{C}\right\}$$

um, so wird Abschnitt 6.2.7 durch die Definitionserweiterung des Distanzvektors ebenfalls abgedeckt.

7.6 Beispiel

Gegeben sei das Programm aus Abbildung 7.2 mit einem Strukturparameter n. Zu berechnen sind die interprozeduralen Abhängigkeiten zwischen dem Hauptprogramm und der Prozedur *power*.

Programmtransformation:

In diesem Fall braucht keine Transformation aus den Abschnitten 6.2.1.2, 6.2.1.4 oder 6.2.1.5 angewendet zu werden. Jedoch weist Statement S beim Zugriff auf das Array A in der zweiten Komponente eine mehrfache Indizierung auf. Diese Komponente wird daher in beiden Indexfunktionen (vgl. 7.1) auf 0 gesetzt.

Schritt 1:

Damit ergeben sich folgenden Matrizen aus der Eingabe:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$a_0 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad b_0 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{array}{l} \leftarrow n \\ \leftarrow \mathit{offset} \end{array}$$

Die Echelon-Reduktion (6.1.1.3) liefert für U und S:

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

```
SUBROUTINE power(x, exp)
   INTEGER, INTENT( INOUT) :: x
   INTEGER, INTENT( IN)    :: exp
   temp = x
   DO k = 1, exp
      temp = temp * x
   END DO
   x = temp
END SUBROUTINE power

DO i1 = 1, n, -1
   DO i2 = 1, i1^2
      IF (i1 >= i2) THEN
         C[i, j] = A[i, B[j]]          {S}
      END IF (i1 >= i2)
      power(A[i1 + 1, j], n)          {T}
   END DO
END DO
```

Abbildung 7.2: Beispielprogramm zur Erweiterung der Eingabesprache

Für $\vec{t} = (t_1, ..., t_4)$ ergibt sich dann aus (6.2.0.6):

$$t_1 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \begin{matrix} \leftarrow n \\ \leftarrow \mathit{offset} \end{matrix}$$

Folglich gilt nach (6.2.0.7) für $\vec{i}$ und $\vec{j}$:

$$\begin{aligned} \vec{i} &= (t_2,\, t_3) \\ \vec{j} &= (t_2 - 1,\, t_4) \end{aligned} \tag{7.6.0.1}$$

Schritt 2:

Nun werden gemäß (6.2.0.10) die Matrizen P, p_0, Q und q_0 sowohl für $\vec{i}$ als auch $\vec{j}$ bestimmt. Jedoch ist anzumerken, daß die obere Grenze der zweiten Schleife keinen affin linearen Ausdruck darstellt. Entsprechend 7.2 wird diese Grenze gestrichen. Jedoch kann die Bedingung des *if*-Statements ausgewertet und dem Iterationsraum von S hinzugefügt werden (vgl. 7.4). Damit existiert für S, nicht jedoch für T eine obere Grenze der zweiten Schleife.

Entsprechend (6.2.0.10) gilt:

$$P^{\vec{i}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad P^{\vec{j}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Q^{\vec{i}} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \qquad Q^{\vec{j}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

sowie

$$p_0^{\vec{i}} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \qquad p_0^{\vec{j}} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$

$$q_0^{\vec{i}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad q_0^{\vec{j}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Damit stellt sich das durch (6.2.0.11) definierte Ungleichungssystem nach Transposition wie folgt dar:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} t_2 \\ t_3 \\ t_4 \\ n \\ 1 \end{pmatrix} \geq 0$$

Dieses System hat die Lösung

$$\begin{aligned} 1 \leq & t_2 \leq n \\ 1 \leq & t_3 \leq t_2 \\ 1 \leq & t_4 \end{aligned} \tag{7.6.0.2}$$

Schritt 3:

Es wird die Methode der Rückwärtssubstitution gewählt. Aus (7.6.0.1) erhält man:

$$H(t_3)\,(j_1,\, j_2) = (j_1 + 1,\, t_3)$$

Desweiteren ergibt sich für $\mathcal{I}$ und $\mathcal{J}$ aus (7.6.0.1) und (7.6.0.2):

$$\begin{aligned}\mathcal{I} &= \{(i_1, i_2) \mid 1 \leq i_1 \leq n, 1 \leq i_2 \leq i_1\} \\ \mathcal{J} &= \{(j_1, j_2) \mid 1 \leq j_1 \leq n-1, 1 \leq j_2\}\end{aligned}$$

Für den Kontext gilt:

$$\mathcal{C} = \{t3 \mid 1 \leq t_3 \leq t2\} = \{t3 \mid 1 \leq t_3 \leq i_1\} = \{t3 \mid 1 \leq t_3 \leq j_1 + 1\}$$

Mit dem Vorzeichenvektor $V = (-1, 1)$, der sich direkt aus der Eingabe entnehmen läßt, findet sich der Distanzvektor

$$\vec{d}^{\,V}(j_1, j_2) = (1, j_2 - t_3).$$

Da dieser offensichtlich stets Korollar 3.3.14 erfüllt, ergeben sich durch die Raumzerlegung keine Einschränkungen.

Die Optimierung aus 6.2.7, die wie zuvor erläutert äquivalent zur Minimierung von $\vec{d}^{\,V}$ ist, liefert das lexikografische Minimum aller Distanzvektoren für $t_3 = j_1 + 1$. Somit ergibt sich als Endergebnis eine einzelne h-Transformation:

$$h(j_1, j_2) = (j_1 + 1, j_1 + 1)$$

Schließlich finden sich hieraus als Quell- und Zielraum:

$$\begin{aligned}QR &= \{(i_1, i_2) \mid 2 \leq i_1 \leq n, i_2 = i_1\} \\ ZR &= \{(j_1, j_2) \mid 1 \leq j_1 \leq n-1, 1 \leq j_2\}\end{aligned}$$

Abbildung 7.3 stellt diese Ergebnisse für $n = 6$ grafisch dar.

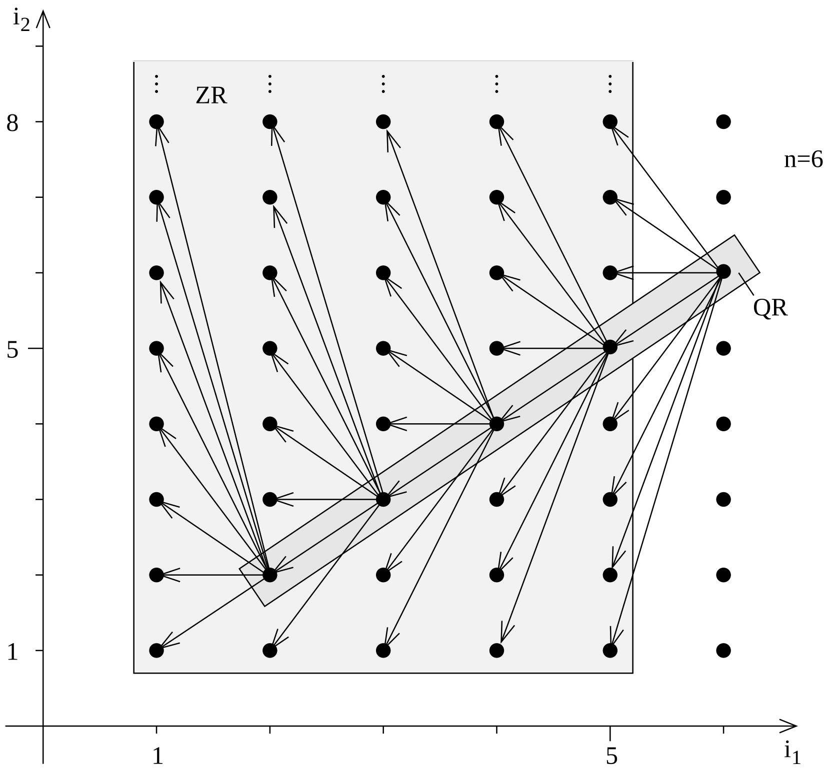


Abbildung 7.3: Abhängigkeiten des Programms aus Abbildung 7.2

Kapitel 8

Zusammenfassung

In dieser Arbeit sind die Ansätze zur Datenabhängigkeitsanalyse von Feautrier und Banerjee vorgestellt worden. Während Feautriers Methode direkte true-Abhängigkeiten, jedoch keine anti- und output-Abhängigkeiten liefert, berechnet Banerjee unter zusätzlichen Restriktionen alle Arten von Abhängigkeiten ohne deren Direktheit zu garantieren.

Um beide Ansätze vergleichen zu können, mußten diese zunächst um die fehlenden Eigenschaften erweitert und eine gemeinsame Form zur Darstellung der Ergebnisse gefunden werden.

Es wurde gezeigt, daß sich Feautriers Ansatz auf die Berechnung aller direkten Abhängigkeiten erweitern läßt. Das hierzu entwickelte Verfahren entspricht uneingeschränkt den Anforderungen.

Um mit Banerjees Methode dieselbe Eingabesprache behandeln zu können, war es zunächst erforderlich, die zusätzlichen Restriktionen aufzuheben. Dieses ist im ersten Teil der Erweiterung vollständig gelungen. Der zweite Teil behandelt die Repräsentation von Banerjees Ergebnissen in dem zur gemeinsamen Darstellung gewählten Konzept der h-Transformation. Es stellt sich heraus, daß eine solche Darstellung durch die Einführung parametrisierter Transformationsscharen stets möglich ist. Ein hierzu entwickeltes Verfahren beinhaltet ebenfalls eine Möglichkeit zur Optimierung, durch die sich im Einzelfall direkte Abhängigkeiten ergeben können. Im allgemeinen kann die Direktheit jedoch nicht garantiert werden.

Schließlich wurde eine Reihe von Möglichkeiten zur Erweiterung der Eingabesprache aufgezeigt, die auf beide Ansätze gleichermaßen anwendbar sind.

Damit stehen zwei leistungsfähige und bis auf den Umstand, daß bei Banerjees Methode letztlich keine Direktheit der Ergebnisse garantiert werden kann, gleichmächtige Verfahren zur Datenabhängigkeitsanalyse zur Verfügung.

Literaturverzeichnis

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

[Ban96] Utpal Banerjee. *Dependence Analysis.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1996.

[Dan66] George B. Dantzig. *Lineare Programmierung und Erweiterungen.* Springer Verlag, 1966.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer Verlag, 1990.

[Fea89] Paul Feautrier. Résolution de systèmes d'inéquations linéaire; mode d'emploi du logiciel pip. Technical report, Laboratoire MASI, Paris, 1989.

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20:23–53, 1991.

[Pug92] William Pugh. Definitions of dependence distance. *ACM Letters on Programming Languages and Systems*, 1(3):261–265, September 1992.

[PW92a] William Pugh and David Wonnacott. Eliminating false data dependences using the Omega test. In *ACM SIGPLAN PLDI'92 conference*, San Francisco, California, July 1992.

[PW92b] William Pugh and David Wonnacott. Going beyond integer programming with the Omega test to eliminate false data dependences.

Technical Report CS-TR-3191, University of Maryland, December 1992.

[PW96] William Pugh and David Wonnacott. The Omega library. Technical report, University of Maryland, April 1996.

[Wie95] Christian Wieninger. Automatische Methoden zur Parallelisierung im Polyedermodell. Master's thesis, Universität Passau, 1995.

[Wol94] Michael Wolfe. The definition of dependence distance. *ACM Transactions on Programming Languages and Systems*, 16(4):1114–1116, July 1994.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, daß diese Diplomarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie daß die Diplomarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt wurde.

Passau, 26. Januar 1997