Programmanalyse und Compileroptimierung

Michael Neuendorf

28. Januar 2005

Diese meine studentische Mitschrift der Vorlesung "Programmanalyse und Compileroptimierung" habe ich zunächst zu dem Zwecke begonnen, bei der Wiederholung des behandelten Stoffes in der Diplomprüfungszeit eine bessere Mitschrift als die meiner eigenen Handschrift zu haben.
Da ich häufig in diversen Vorlesungen beim Mittexen gefragt wurde, ob ich die Mitschrift auch ins Netz stellen würde, stelle ich auch diese Mitschrift zur Verfügung; ich werde sie im Laufe der Vorlesung fortlaufend erweitern und in gewissen Zeitabständen in der aktuellen Version ins Netz stellen.
Der Lehrstuhl für Informatik 2 gestattet mir freundlicherweise, die zur Vorlesung gehörigen Folien mit in ein ZIP-Archiv zu diesem Dokument zu stellen. An dieser Stelle möchte ich mich dafür herzlich bedanken!
Um den zeitlichen Ablauf zu dokumentieren und somit vielleicht das Vergleichen mit der eigenen Mitschrift zu erleichtern, habe ich Datumsangaben eingefügt, die jeweils den Termin des nachfolgenden Vorlesungstextes angeben.
Wie üblich werden auch wieder einige Tipp- und Abschreibfehler ihren Weg in diese Mitschrift gefunden haben. Solltet Ihr einen finden, so seid doch bitte so nett und schickt mir eine eMail mit Abschnittname und Fehlerbeschreibung an diese Adresse: *michael.neuendorf@rwth-aachen.de*.

Michael Neuendorf

Inhaltsverzeichnis

1 Einleitung

[2004/10/14]

Ziel der Programmanalyse (PA): Berechnung der Laufzeiteigenschaften zur Übersetzungszeit, abhängig von der Programmsemantik, z.B. Terminierung, i.a. nicht entscheidbare Eigenschaften $\rightsquigarrow$ PA berechnet Approximation dieser Eigenschaften; Mittel: Abstrakte Interpretation

Anwendung der PA:

- Compileroptimierung durch Programmtransformation
- Konstruktion von Software-Tools für Verifikation, Testen, Debugging, ...

hier: Compileroptimierung Verbesserung von Programmen mithilfe von Programmeigenschaften; dabei werden verschiedene Kostenmaße betrachtet:

- Laufzeit
- Speicherbedarf der Laufzeitdaten (Größe von Stack und Heap)
- Speicherbedarf des Codes (Programmgröße): Kompakter Code ist wichtig bei Palmtops, Mobiltelefonen, SW für eingebettete Systeme

Programmiersprachen (PS):

- imperative
  - Objektorientierung (OO, Java: Klassenoptimierung)
  - Funktionale Programmierung (FP, Striktheitsanalyse)
  - Logikprogrammierung (LP)

Möglichkeiten für PA und CO:

1. Attributierter abstrakter Syntaxbaum (AST, Quellprogramm) erfordert PA für strukturierte Programme
2. Zwischencode: elementare Ausdrücke, 3-Adreßcode, keine Kontrollstrukturen bis auf Sprungbefehle, keine Datenstrukturen, Adreßberechnung, Pointer
3. Maschinencode: Registerzuteilung, Codeselection, Instruktionsanordnung

Inhalt: PACO für Zwischencode

1. Straight-Line-Code (ohne Verzweigung, Iteration, Rekursion, Datenstrukturen): wichtige Teilklasse, da dieser z.B. in Schleifenrümpfen auftritt
   90/10-Regel: Ein Programm benutzt 90% der Ausführungszeit 10% seines Codes $\rightsquigarrow$ lokale Analyse
2. Iterative Programme:
   a) Basisblock-Darstellung und Flußgraphen $\rightsquigarrow$ globale Analyse, intra-prozedural
   Technik: Datenflußanalyse mit abstrakter Interpretation
   b) Kontrollflußanalyse und Schleifenoptimierung
3. Rekursive Programme $\rightsquigarrow$ interprozedurale Analyse

4. Programme mit Datenstrukturen
   statische und dynamische Datenstrukturen (DS), Shape-Analyse

Optimierungen: dead code elimination, common subexpression elimination, constand propagation / content folding, reduction in strength, code motion
Analysen: live variables, avalaible expression, pointer, shape, alias

2 Analyse und Optimierung von Straight-Line-Code

Programme ohne Verzweigung, Iteration, Rekursion, Datenstrukturen, Ausdrücke entschachtelt

- Bedeutung: wiederholte Ausführung in Schleifen
- Ziel: Codeoptimierung (Laufzeit)
- Themen:
  - Live Variable (LV) – Dead Code (DC) Elimination
  - Available Expression (AE) Analyse – Common Subexpression (CS) Elimination
  - Reading Definition (RD) Analyse – Konstantenpropagation / Faltung

SLC-Programme (Syntax) Die Syntax von SLC-Programmen ist wie folgt definiert:

$V = \{x, y, z, \ldots\}$	Variablen
$C = \{a, b, c\}$	Konstantensymbole
$\sum = \bigcup_{r=1}^{\infty} \sum^{(r)}$ mit $\mid \sum \mid \leq \infty$	Operationssymbole
$\alpha \in \underline{\text{Anw}}$	(α ist eine Anweisung): $\leftrightarrow \alpha = x \leftarrow e$ und $e \in \{f(u_1, \ldots, u_r) \mid f \in \sum^{(r)}, u_i \in V \cup V\} \cup V \cup C$
$V_\alpha := \{x\} \cup V_e$	(Variablen von α bzw. von e) x wird in α definiert, $v \in V_e$ wird in α benutzt.
$\beta \in Blo$	(β ist ein Block): $\leftrightarrow \beta = \alpha_1; \ldots; \alpha_n$, $\alpha_i \in \underline{\text{Anw}}, n \geq 0$
$V_\beta := \bigcup_{i=1}^{n} V_{\alpha_i}$	(Variablen von β)
$\pi \in SLC$	(π ist ein SLC-Programm): $\leftrightarrow \pi = (IV, \beta, OV), \beta \in Blo, IV \subseteq V$, endliche (Eingabevariablen), vor β definiert, $OV \subseteq V$, endlich, nicht-leer (Ausgabevariablen, nach β benutzt)
$V_\pi := \bigcup_{i=1}^{n} V_{\alpha_i} \cup IV \cup OV$	π erfüllt die Variablenbedingung (π vollständig, wohldefiniert): $\leftrightarrow$ Jede Programmvariable $v \in V_\pi$ wird vor ihrer Benutzung definiert

Generalvoraussetzung: SLC-Programme sollen die Variablenbedingung erfüllen.

SLC-Programme (Semantik) Sei $\pi = (IV, \beta, OV) \in SLC$. Sei $\mathfrak{A} = \langle A; \varphi \rangle$ eine (Σ, C)-Algebra, d.h. A Menge,

$f \in \Sigma^{(r)} \curvearrowright \varphi(f) = f_{\mathfrak{A}} : A^r \to A$

$c \in C \curvearrowright \varphi(c) = c_{\mathfrak{A}} \in A$

Zustandsraum: $Z := \{\sigma \mid \sigma : V_\pi \to A\}$

- $\alpha \in \underline{\text{Anw}} \mapsto [\![\alpha]\!]_{\mathfrak{A}} : Z \to Z$ Zustandstransformation
- $\alpha = x \leftarrow e$
- $[\![x \leftarrow e]\!]_{\mathfrak{A}}(\sigma) = \sigma'$ mit $\sigma'(x) = [\![e]\!]_{\mathfrak{A}}(\sigma) \in A$ (Wert von e im Zustand σ, "Einsetzen und Ausrechnen") und $\sigma'(y) = \sigma(y)$ für $y \neq x$.
- $\beta = \alpha_1; \ldots; \alpha_n \mapsto [\![\beta]\!]_{\mathfrak{A}} := [\![\alpha_n]\!] \circ [\![\alpha_{n-1}]\!] \circ \ldots \circ [\![\alpha_1]\!]$
- $\beta = e \mapsto [\![\beta]\!]_{\mathfrak{A}} := \mathrm{id}_Z$

IV und OV bestimmen entsprechende Teile des Zustandsraumes Z:

- $Z_{in} := \{\sigma_{in} \mid \sigma_{in} : IV \to A\}$ Eingabezustände

- $Z_{out} := \{\sigma_{out} | \sigma_{out} : OV \to A\}$ Ausgabezustände

Damit ergibt sich die Programmsemantik

- $[\![\pi]\!]_{\mathfrak{A}} : Z_{in} \to Z_{out}$
- $[\![\pi]\!]_{\mathfrak{A}}(\sigma_{in}) = \sigma_{out}$
- $\sigma_{in} \mapsto \sigma_0(x) := \sigma_{in}(x)$ für $x \in IV$, $\sigma_0(y) := a \in A$ (beliebig) für $y \notin IV$
- $\sigma_{out}(Z) = [\![\beta]\!]_{\mathfrak{A}}(\sigma_0(Z))$ für $Z \in OV$

Beachte: π erfüllt die Variablenbedingung $\leftrightarrow [\![\pi]\!]_{\mathfrak{A}}$ unabhängig von Anfangswerten für $y \in V_\pi \setminus IV$

Definition (Programmäquivalenz): $\pi_1, \pi_2 \in SLC, \mathfrak{A}(\Sigma, C) - \underline{\text{Algebra}}$

- π_1 und π_2 $\mathfrak{A}$-äquivalent ($\pi_1 \sim_{\mathfrak{A}} \pi_2$): $\leftrightarrow [\![\pi_1]\!]_{\mathfrak{A}} = [\![\pi_2]\!]_{\mathfrak{A}}$
- π_1 und π_2 stark äquivalent ($\pi_1 \approx \pi_2$): $\leftrightarrow [\![\pi_1]\!]_{\mathfrak{A}} = [\![\pi_2]\!]_{\mathfrak{A}}$ für jede (Σ, C)-Algebra $\mathfrak{A}$.

Optimierung von SLC-Programmen Sei $c : SLC \to \mathbb{N}$ eine Kostenfunktion, z.B. die Zahl der Anweisungen, Zahl der Operationen, Summe der Operationsgewichte (+ billiger als *). Für $\pi_1, \pi_2 \in SLC$ mit $\pi_1 \sim_{\mathfrak{A}} \pi_2$ (bzw. $\pi_1 \approx \pi_2$) ist $\pi_1 \leq_c \pi_2$ (c besser): $\leftrightarrow c(\pi_1) \leq c_(\pi_2)$.
π_1 ist c-optimal (bzgl. $\mathfrak{A}$): $\leftrightarrow \pi_1 \leq_c \pi_2$ für alle $\pi \in SLC$ mit $\pi_1 \approx \pi$ (bzw. $\pi_1 \sim_{\mathfrak{A}} \pi$).
Ziel: Programmtransformationen zur SLC-Optimierung

2.1 AE-Analyse und CS-Elimination

Beispiel (Folie) $\pi \approx \pi'$ und $c(\pi') < c(\pi)$ bzgl. Operationsanzahl, aber $c(\pi') > c(\pi)$ bzgl. Codelänge.

2.1.1 AE-Analyse

Bestimmung der für jede Anweisung verfügbaren Ausdrücke (AE = Available Expression). Sei $\pi = (IV, \beta, OV) \in SLC$ mit $\beta = \alpha_1; \ldots; \alpha_n$.

$$\begin{aligned} \underline{\text{OpExp}}_\beta &:= \{e | x \leftarrow e = \alpha_i, e = f(u_1, \ldots, u_i)\} \\ &= \{oe_1, \ldots, oe_t\} \text{ Operationsausdrücke von } \beta \end{aligned}$$

Verfügbarkeit durch Bitvektoren beschreiben: $\mathbb{B}^t := \{(b_1, \ldots, b_t) | b_i \in \{0, 1\}\}$
Bedeutung:

- $b_i = 0$ heißt "oe_i ist verfügbar"
- $b_i = 1$ heißt "oe_i ist nicht verfügbar"
- $AE_i \in \mathbb{B}^t$: Die für die Ausführung von α_i verfügbaren Ausdrücke ($i = 1, \ldots, n$)
- $AE_1 = (1, \ldots, 1)$
- $\alpha \in \underline{\text{Anw}} \mapsto t_\alpha : \mathbb{B}^t \to \mathbb{B}^t$ (information transformer)

t_α beschreibt die Veränderung der Verfügbarkeit von Ausdrücken durch Ausführung von α.

- $t_{x \leftarrow e} := \underline{\text{kill}}_x \circ \underline{\text{gen}}_e$ mit $\underline{\text{gen}}_e, \underline{\text{kill}}_x : \mathbb{B}^t \to \mathbb{B}^t$
- $\underline{\text{gen}}_e(b_1, \ldots, b_t) = (b'_1, \ldots, b'_t)$

$$b'_i = \begin{cases} 1 & \text{falls } e = oe_i \\ b_i & \text{sonst} \end{cases}$$

- $\underline{\text{kill}}_x(b_1, \ldots, b_t) = (b'_1, \ldots, b'_t)$

$$b'_i = \begin{array}{ll} 1 & \text{falls } x \in V_{oe_i} \\ b_i & \text{sonst} \end{array}$$

- $AE_{i+1} = t_{\alpha_i}(AE_i)$ für $i = 1, \ldots, n-1$
- Ergebnis: Analyseinformationen $AE_1, \ldots, AE_n$

[2004/10/20]

- Beobachtung: $\alpha_i = x \leftarrow x$: keine Veränderung von x, aber die Verfügbarkeit von Ausdrücken wird i.a. reduziert

Beispiel (Folie 2.2) $\underline{\text{OpExp}}_\beta = \{oe_1 = x + y, oe_2 = z * y, oe_3 = v - z, oe_4 = w + z\}$

$AE_1 = (1, 1, 1, 1)$
$AE_2 = (0, 1, 1, 1)$
$AE_3 = (0, 1, 1, 1)$
$AE_4 = (1, 1, 1, 1)$
$AE_5 = (0, 1, 1, 1)$
$AE_6 = (0, 1, 0, 1)$

2.1.2 CS-Elimination

- Ziel: Vermeidung der wiederholten Berechnung eines Ausdrucks mit denselben Variablenwerten, weniger durchgeführte Rechenoperationen
- Idee: Ausdruckswerte auf temporären Variablen zwischenspeichern, statt wiederholter Berechnung temporäre Variable verwenden

$T_{CS} : SLC \to SLC$

$T_{CS}(IV, \beta, OV) := (IV, T_{CS}(\beta), OV)$

$T_{CS}(\alpha_1; \ldots, \alpha_n) := T_{CS}(\alpha_1); \ldots, T_{CS}(\alpha_n)$

3 Fälle für $T_{CS}(\alpha_i)$:

1. $\alpha_i = x \leftarrow e, e \notin \underline{\text{OpExp}}_\beta \curvearrowright T_{CS}(\alpha_i) := \alpha_i$
2. $\alpha_i = x \leftarrow oe_j, AE_i^{(j)} = 1 \curvearrowright T_{CS}(\alpha_i) := tv_j \leftarrow oe_j; x \leftarrow tv_j$
3. $\alpha_i = x \leftarrow oe_j, AE_i^{(j)} = 0 \curvearrowright T_{CS}(\alpha_i) := x \leftarrow tv_j$

mit temporären Variablen $tv_1, \ldots, tv_t$ für die Op-Ausdrücke $oe_1, \ldots, oe_t$.

Beispiel (Folie 2.2) Fall (1) tritt nicht auf. Die "relevanten" Verfürbarkeitsbits sind

$AE_1^{(1)} = 1, AE_2^{(2)} = 1, AE_3^{(3)} = 0, AE_4^{(1)} = 1, AE_5^{(3)} = 1, AE_6^{(4)} = 1$

$$\begin{array}{ll} T_{CS}(\beta) = & tv_1 \leftarrow x + y; \\ & z \leftarrow tv_1; \\ & tv_2 \leftarrow z * y; \\ & z \leftarrow tv_2; \\ & x \leftarrow tv_1; \\ & tv_1 \leftarrow x + y; \\ & v \leftarrow tv_1; \\ & tv_3 \leftarrow v - z; \\ & w \leftarrow tv_3; \\ & tv_4 \leftarrow w + v; \\ & w \leftarrow tv_4; \end{array}$$

Korrektheit $\pi \approx T_{CS}(\pi)$ ist unabhängig von der Interpretation. Die Korrektheit ist klar für die Fälle (1) und (2). Im Fall (3) gilt: Der verfügbare Wert von oe_j liegt als Wert von tv_j vor.

Optimierung: Op-Anzahl $(T_{CS}(\pi)) \leq$ Op-Anzahl (π), aber: Code-Länge $(T_{CS}(\pi)) \geq$ Code-Länge (π)

2.2 LV-Analyse und DC-Elimination

2.2.1 LV-Analyse

$\pi = (IV, \beta, OV) \in SLC$.

- Variablenbedingung: $v \in V_\pi$ wird vor ihrer Benutzung definiert
- Möglich: $v \in V_\pi$ wird nach ihrer Definition nicht benutzt

Definition $v \in V_\pi$ heißt **lebendig in** i $(v \in LV_i)$ für $i = 1, \ldots, n$, falls v in $\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_n$ oder OV benutzt wird, ohne vor dieser Benutzung neu definiert zu werden.

Bestimmung der LV_i durch Rückwärtsanalyse:

$LV_n = OV$
$\alpha \in \underline{\text{Anw}} \mapsto t_\alpha : \mathcal{P}(V_\pi) \to \mathcal{P}(V_\pi)$ "Rückwärtstransformation"
$\alpha = x \leftarrow e : t_\alpha := \underline{\text{gen}}_e \circ \underline{\text{kill}}_e$
$\underline{\text{kill}}_x(M) := M \setminus \{x\}, \quad M \in \mathcal{P}(V_\pi)$
$\underline{\text{gen}}_x(M) := M \cup V_e$
$LV_{i-1} := t_{\alpha_i}(LV_i), \quad i = 1, \ldots, n$

Beispiel (Folie 2.3)

$LV_4 = \{u, v\}$
$LV_3 = \{u, x, y\}$
$LV_2 = \{x, y\}$
$LV_1 = \{x, y, u, z\}$
$LV_0 = \{x, y, z\}$

2.2.2 DC-Elimintation

$T_{DC} : SLC \to SLC$

Entferne α_i aus $\beta = \alpha_1; \ldots; \alpha_n$, falls $\alpha_i = x \leftarrow e$ und $x \not\in LV_i (i = 1, \ldots, n)$.

Beispiel (Folie 2.3) Entferne α_2, nicht jedoch α_1! Eine Möglichkeit dazu wäre, die LV-Analyse zu wiederholen. Eine verbesserte Analyse ist jedoch die **NV (needed variable) - Analyse**

NV-Analyse Modifikation: $\alpha = x \leftarrow e$

$\tilde{t}_\alpha(M) := \underline{\text{if}}\ x \in M\ \underline{\text{then}}\ M \setminus \{x\} \cup V_e\ \underline{\text{else}}\ M$
$NV_n = OV$
$NV_{i-1} := \tilde{t}_{\alpha_i}(NV_i)$

Beispiel (Folie 2.4)

$NV_4 = \{u, v\}$
$NV_3 = \{u, x, y\}$
$NV_2 = \{x, y\}$
$NV_1 = \{x, y\}$
$(NV_0 = \{x, y\})$

Korrektheit: $\pi \approx T_{DC}(\pi) \approx \tilde{T}_{DC}(\pi)$

Optimierung: $c(\tilde{T}_{DC}(\pi)) \leq c(T_{DC}(\pi)) \leq c(\pi)$, c: Programmlänge, Op-Anzahl

2.3 RD-Analyse und Konstantenfaltung

Ziel: Partielle Auswertung von $\pi \in SLC$-Programmen unter Kenntnis von Konstanten (Einfluß der Semantik); genauer: Ersetze konstante Variablen und konstante Ausdrücke auf rechten Seiten von Wertzuweisungen durch ihre Werte (Propagation [Variablen] und Faltung [Ausdrücke])

2.3.1 RD-Analyse

$\pi = (IV, \beta, OV) \in SLC$ mit $\beta = \alpha_1; \ldots; \alpha_n$, Interpretation $\mathfrak{A} = \langle A; \varphi \rangle$. Bestimme für $i = 1, \ldots, n$ die dort gültigen, eingabeunabhängigen (konstanten) Variablenwerte:

- $V_\pi = \{v_1, \ldots, v_k\}$
- $D := (A \cup \{\Box\})^k$, $d = (d_1, \ldots, d_k)$
- $d_j \in A$ bedeutet: v_j hat den Wert d_j
- $d_j = \Box$ bedeutet: Wert von v_j unbekannt oder eingabeabhängig für $j = 1, \ldots, k$
- $\alpha = x \leftarrow e \in \underline{\text{Anw}} \mapsto t_\alpha : D \to D$

Erweiterung der Ausdruckssemantik von e:

$[\![e]\!]_{\mathfrak{A}}(d) \in A \cup \{\Box\}$

$[\![e]\!]_{\mathfrak{A}}(d) = \Box :\leftrightarrow \exists v_j \in V_e$ mit $d_j = \Box$

(Strikte Semantik: $0 * \Box = \Box$)

$t_{x \leftarrow e}(d_1, \ldots, d_k) = (d'_1, \ldots, d'_k)$

mit

$$d'_j = \begin{cases} [\![e]\!]_{\mathfrak{A}}(d_1, \ldots, d_k) & \text{falls } x = v_j \\ d_j & \text{sonst} \end{cases}$$

$RD_1 = (\Box, \Box, \ldots, \Box) \in D$

$RD_{i+1} = t_{\alpha_i}(RD_i), \qquad RD_i \in D = A \cup \{\Box\}$

Beispiel (Folie 2.4) $V_\pi = (x, y, z)$

$RD_1 = (\Box, \Box, \Box)$
$RD_2 = (\Box, 10, \Box)$
$RD_3 = (\Box, 10, 2)$
$RD_4 = (\Box, 10, 20)$
$RD_5 = (\Box, \Box, 20)$
$RD_6 = (\Box, \Box, \Box)$

2.3.2 Konstantenfaltung

$T_{CF} : SLC \to SLC$
$T_{CF}(IV, \beta, OV) := (IV, T_{CF}(\beta), OV)$
$T_{CF}(\alpha_1; \ldots; \alpha_n) := T_{CF}(\alpha_1); \ldots; T_{CF}(\alpha_n)$
$T_{CF}(\alpha_i) := x_i \leftarrow T_{CF}(e_i, RD_i), \qquad \alpha_i = x_i \leftarrow e_i$
$T_{CF}(a, d) := a$
$T_{CF}(v_j, d) := \underline{\text{if}}\ d_j \in A\ \underline{\text{then}}\ d_j\ \underline{\text{else}}\ v_j$
$T_{CF}(f(u_1, \ldots, u_r), d) := \underline{\text{if}}\ \exists j \in \{1, \ldots, k\}, T_{CF}(u_j, d) \in V_\pi$
$\quad \underline{\text{then}}\ f(T_{CF}(u_0, d), \ldots, T_{CF}(u_r, d))$
$\quad \underline{\text{else}}\ f_{\mathfrak{A}}(T_{CF}(u_1, d), \ldots, T_{CF}(u_r, d))$

Beispiel (Folie 2.4)

$$\begin{aligned} T_{CF}(\alpha_3) &= z \leftarrow T_{CF}(y * z, (\square, 10, 2)) \\ &= z \leftarrow 20 \end{aligned}$$

$$\begin{aligned} T_{CF}(\alpha_5) &= z \leftarrow T_{CF}(y * z, (\square, \square, 20)) \\ &= z \leftarrow T_{CF}(y, (\square, \square, 20)) * T_{CF}(z, (\square, \square, 20)) \\ &= z \leftarrow y * 20 \end{aligned}$$

Korrektheit von T_{CF}: Offensichtlich gilt: $\pi \sim_{\mathfrak{A}} T_{CF}(\pi)$

Optimierung: Programmlänge unverändert, die Op-Anzahl kann sinken. Ein besonderer Seiteneffekt ist die Entstehung von Dead-Code (im Beispiel (Folie 2.4): $\alpha_1, \alpha_2, \alpha_3$).

2.4 DAG-Darstellung von SLC-Programmen

- DAG (directed acyclic graph): Nützliche Datenstruktur zur Implementierung von Transformationen auf SLC-Programmen
- Idee: Termdarstellung der Werte einer Berechnung

Definition (gerichteter (Σ, V, A)-Graph): $D = \langle K, \underline{\text{lab}}, \underline{\text{suc}} \rangle$ heißt gerichteter (Σ, V, A)-Graph, wenn

- K nichtleere, endliche Menge von Knoten, $k \in K$,
- $\underline{\text{lab}} : K \to (\Sigma \cup V \cup A)$ eine **Markierungsfunktion**
- $\underline{\text{suc}} : K \times \mathbb{N} \dashrightarrow K$ eine **Nachfolgerfunktion** mit der Eigenschaft: $\underline{\text{suc}}(k, i)$ ist definiert $\leftrightarrow \underline{\text{lab}}(k) = f \in \Sigma_{(r)}$ und $1 \leq i \leq r$ ist.

Definition ((Σ, V, A)-Graph) Ein gerichteter azyklischer (Σ, V, A)-Graph heißt (Σ, V, A)-DAG.
Beachte: Jeder Knoten k eines solchen Graphen repräsentiert einen Term $t_k \in T_\Sigma(V \cup A)$.

Der DAG eines SLC-Programmes Sei $\pi = (IV, \beta, OV) \in SLC$ mit $\beta = \alpha_1; \ldots; \alpha_n$. Der DAG von π wird durch "symbolische Programmausführung mit Konstantenfaltung" bestimmt. Wir definieren den DAG zunächst ohne Bezug auf OV : $D(IV, \beta) = \langle K; \underline{\text{lab}}, \underline{\text{suc}} \rangle$ zusammen mit einer Bewertungsfunktion

$$\underline{\text{val}} : IV \cup V_\beta \to K,$$

bei der $\underline{\text{val}}(x) = k$, falls k den Wert von x nach Durchführung von β darstellt.
Idee:

- verschiedene Knoten für verschiedene Werte (sharing) $\rightsquigarrow$ CS-Elimination
- ersetzen konstanter Ausdrücke $\rightsquigarrow$ CF-Transformation

Anschließende Betrachtung von OV:

- OV bestimmt Teilgraphen $\rightsquigarrow D(\pi) \rightsquigarrow$ DC-Elimination

Konstruktion von $D(IV, \beta)$ und $\underline{\text{val}}$ durch Induktion über die Codelänge n:

1. Induktionsanfang: $n = 0$
   $D(IV, \varepsilon) := \langle K; \underline{\text{lab}}, \underline{\text{suc}} \rangle$ mit $K := IV, \underline{\text{lab}}(x) := x$ für alle $x \in IV$. $\underline{\text{Def}}(\underline{\text{suc}}) := \varnothing, \underline{\text{val}}(x) := x$.
2. Induktion: $n \rightsquigarrow n + 1$
   (IV, β_{n+1}) mit $\beta_{n+1} = \alpha_1; \ldots; \alpha_n; \alpha_{n+1}$. Nach der Induktionsvoraussetzung ist für $\beta_n = \alpha_1; \ldots; \alpha_n$ bekannt: $D(IV, \beta_n) = \langle K; \underline{\text{lab}}, \underline{\text{suc}} \rangle$ und $\underline{\text{val}}(x) \in K$ für alle $x \in IV \cup V_{\beta_n}$.

Konstruktion von $D(IV, \beta_{n+1}) = \langle K'; \underline{\text{lab}}', \underline{\text{suc}}' \rangle$ und $\underline{\text{val}}' : IV \cup \beta_{n+1} \to K'$ ist bestimmt durch $\alpha_{n+1} = x \leftarrow e$.

1. Fall: $e = y$
   Nach Induktionsvoraussetzung (und Variablenbedingung) ist $\underline{\text{val}}(y) \in K$. $D(IV, \beta_{n+1}) := D(IV, \beta_n)$, $\underline{\text{val}}'(y) := \underline{\text{val}}(y)$ ("sharing") und $\underline{\text{val}}'(x') := \underline{\text{val}}(x')$ für $x' \neq x$.

2. Fall: $e = a \in A$

   a) $a \in K$: $D(IV, \beta_{n+1}) := D(IV, \beta_n)$ und $\underline{\text{val}}'(x) = a$ und $\underline{\text{val}}'(x') = \underline{\text{val}}(x')$ für $x' \neq x$.

   b) $a \notin K$: $K' := K \cup \{a\}, \underline{\text{lab}}'(a) = a$ und $\underline{\text{suc}}'(a, i)$ ist nicht definiert für alle i, $\underline{\text{suc}}'(k, i) = \underline{\text{suc}}(k, i)$ sonst. $\underline{\text{val}}'(x) := a \in K'$ und $\underline{\text{val}}'(x') = \underline{\text{val}}(x')$ für $x' \neq x$.

3. Fall: $e = f(u_1, \ldots, u_r)$ mit $u_j \in V \cup A$

   a) Für alle $j = 1, \ldots, r$ gilt: $u_j \in A$ (falls dabei $u_j \neq K$, neuen Knoten wie in 2b) oder $u_j \in V$ mit $\underline{\text{val}}(u_j) \in A$. Dann folgt: $f_{\mathfrak{A}}(u'_1, \ldots, u'_r) := b \in A$ mit

   $$u'_j = \begin{cases} u_j & \text{falls } u_j \in A \\ \underline{\text{val}}(u_j) & \text{falls } u_j \in V \end{cases}$$

      i. $b \in K$: $D(IV, \beta_{n+1}) := D(IV, \beta_n)$, $\underline{\text{val}}'(x) := b$ und $\underline{\text{val}}'(x') = \underline{\text{val}}(x')$ für $x' \neq x$.

      ii. $b \notin K$: Erweiterung von $D(IV, \beta_n)$ um neuen Knoten für b. Bewertung von $\underline{\text{val}}'$ wie in 3(a)i

   b) Es gibt $j \in \{1, \ldots, r\}$ mit $u_j \in V$ und $\underline{\text{val}}(u_j) \notin A$.

      i. Es gibt $k \in K$ mit $\underline{\text{lab}}(k) = f$ und

      $$\underline{\text{suc}}(k, j) = \begin{cases} a & \text{falls } u_j = a \in A \\ \underline{\text{val}}(u_j) & \text{falls } u_j \in V \end{cases}$$

      Dann ist $D(IV, \beta_{n+1}) := D(IV, \beta_n)$ und $\underline{\text{val}}'(x) := k$ und $\underline{\text{val}}'(x') = \underline{\text{val}}(x')$ für $x' \neq x$.

      ii. Es gibt kein $k \in K$, welcher nach 3(b)i den Wert e repräsentiert. $K' := K \dot{\cup} \{k'\}$ (neuer Knoten k'). $\underline{\text{lab}}'(k') := f$,

      $$\underline{\text{suc}}'(k', j) := \begin{cases} a & \text{falls } u_j = a \in A \\ \underline{\text{val}}(u_j) & \text{falls } u_j \in V \end{cases}$$

      $\underline{\text{val}}'(x) := k'$, sonst unverändert.

Eigenschaften von $D(IV, \beta)$

1. Verschiedene Knoten repräsentieren verschiedene Terme: $k_1 \neq k_2 \curvearrowright t_{k_1} \neq t_{k_2}$
2. Kein Knoten repräsentiert einen konstanten Operationsterm $t_k \in T_\Sigma(A) \curvearrowright t_k \in A$

Folgerung

1. unterstützt CS-Elimination
2. unterstützt CF-Transformation

Beispiel (Folie 2.5:DAG-Konstruktion)

- Gemeinsame Teilausdrücke: $\underline{\text{val}}(t) = \underline{\text{val}}(u)$
- Konstantenfaltung: $\underline{\text{val}}(z) = 4, \underline{\text{val}}(v) = \underline{\text{val}}(t) * 4$

Beobachtung: $D(IV, \beta)$ unterstützt mit Hilfe von OV auch die DC-Elimination. $k \in K$ heißt **ausgaberelevant**, wenn $y \in OV$ existiert, sodaß k von $\underline{\text{val}}(y)$ erreichbar ist.
Nicht ausgaberelevante Knoten repräsentieren Dead Code (im Beispiel: 2-+-Knoten und 2). Durch Entfernen dieser Knoten und Einschränkung von $\underline{\text{val}}$ auf OV erhalten wir:

1. den DAG von π: $D(\pi)$
2. die Bewertungsfunktion $\underline{\text{val}} : OV \to K$ (s. Folie 2.6)

Programmoptimierung mit $D(\pi)$ und $\underline{\text{val}} : OV \to K$: Konstruktion eines DAG-optimierten Programmes $T_D(\pi) := \pi_D := (IV, \beta_D, OV)$:

1. Fall: $\underline{\text{val}}$ injektiv, d.h. verschiedene Ausgabevariablen zeigen auf verschiedene Knoten.

   a) $\underline{\text{val}}(OV) \cap (IV \cap A) = \varnothing$, d.h. Ausgabevariablen zeigen nur auf Op-Knoten:
   - Idee: Op-Knoten als Hilfsvariablen verwenden und falls $\underline{\text{val}}(y) = k$ y statt k verwenden.
   - Jeder Op-Knoten k bestimmt mit $\underline{\text{lab}}(k) = f \in \Sigma^{(r)}$ und den Nachfolgerknoten eine Anweisung: $\alpha_k := k \leftarrow f(\underline{\text{suc}}(k,1), \ldots, \underline{\text{suc}}(k,r))$
   - Beachte: $\underline{\text{suc}}(k,j) \in IV \cup A \cup K \cup OV$
   - Wähle eine Bottom-Up-Numerierung der Operationsknoten $k_1, k_2, \ldots, k_s$ (z.B. schichtenweise von links nach rechts), sodaß $k_j \leftarrow f(\ldots k_i \ldots) \curvearrowright i < j$. Dann gilt: $\pi_D := (IV, \beta_D, OV) \in SLC$ mit $\beta_D := \alpha_{k_1}; \ldots; \alpha_{k_s}$.
   - Beachte: $OV \subseteq \{k_1, \ldots, k_s$ und π_D erfüllt die Variablenbedingung. Außerdem besitzt π_D die "single assignment"-Eigenschaft.

   b) $\underline{\text{val}}(y) \in IV \cap A$: π_D von 1a um $y \leftarrow \underline{\text{val}}(y)$, es seie denn, daß $y = \underline{\text{val}}(y)$.

2. Fall: $\underline{\text{val}}(y_1) = \underline{\text{val}}(y_2) = \ldots = \underline{\text{val}}(y_n)$: π_D-Konstruktion von 1a bzgl. y_1 durchführen und nun um $y_2 \leftarrow y_1; \ldots; y_n \leftarrow y_1$ ergänzen.

Zusammenhang zwischen DAG-Optimierung und CS-, DC-, CF- und CP-Optimierung

- Beispiel: $T_{DC}(T_{CP}(T_{CS}(T_{CF}(\pi)))) = T_D(\pi)$

$T_{CF}(\pi) = \pi_1$	$T_{CF}(\pi_1) = \pi_2$	$T_{CP}(\pi_2) = \pi_3$	$T_{DC}(\pi_3) = \pi_4$
$u \leftarrow 2;$	$u \leftarrow 2;$	$u \leftarrow 2;$	
$w \leftarrow x + y;$	$w \leftarrow x + y;$	$w \leftarrow x + y;$	
$z \leftarrow 4;$	$z \leftarrow 4;$	$z \leftarrow 4;$	
$u \leftarrow x * w;$	$u' \leftarrow x * w;$	$u' \leftarrow x * w;$	$w \leftarrow x + y;$
$v \leftarrow 4 + w;$	$u \leftarrow u';$	$u \leftarrow u';$	$u' \leftarrow x * w;$
$v \leftarrow u + 4;$	$v \leftarrow 4 + w;$	$v \leftarrow 4 + w;$	$v \leftarrow u' * 4;$
$t \leftarrow x * w;$	$v \leftarrow u + 4;$	$v \leftarrow u' + 4;$	
$v \leftarrow t * 4;$	$t \leftarrow u';$	$t \leftarrow u';$	$OV : u', v;$
	$v \leftarrow t * 4;$	$v \leftarrow u' * 4;$	
$OV : u, v;$	$OV : u, v;$	$OV : u', v;$	

- Allgemein gilt:

  1. Die DAG-Transformation ist korrekt: $\pi \approx \pi_D$
     Beachte: Die Konstantenfaltung kann bereits ohne Interpretation durchgeführt werden (bzw. auf Termen; Grundterme als Konstantensymbole zulassen).
  2. $T_D(\pi)$ ist optimal bzgl. $T_{DC}, T_{CS}, T_{CF}, T_{CP}$, d.h. keine dieser Transformationen kann $T_D\pi$ weiter optimieren.
  3. $T_D\pi$ optimal bzgl. $\approx$ und Codelänge. (?)
  4. Vermutung: Für jedes $\pi \in SLC$ gibt es ein $n \in \mathbb{N}$ mit $T_D(\pi) = [T_{DC} \circ (T_{CP} \circ T_{CS})^n \circ T_{CF}](\pi)$

     [2004/11/03]

  5. T_D nicht auf iterative Programme übertragbar, wohl aber die anderen Transformationen $T_{CS}, T_{CF}, T_{CP}, T_{DC}$.

2.5 Weitere Optimierungen von SLC-Programmen

Berücksichtigung der Interpretation $\mathfrak{A} = \langle A; \varphi \rangle$ als (Σ, C)-Algebra.

- Algebraische Eigenschaften: Gleichungen

- Beispiel: $\langle \mathbb{Z}; +, -, *, / \rangle$
  - neutrale Elemente $0 + a = a$, $1 * a = a$, $a/1 = a$
  - kommutative Operationen $a + b = b + a$, $a * b = b * a$
  - assoziative Operationen $(a + b) + c = a + (b + c)$
  - distributive Operationen $a * (b + c) = a * b + a * c$
- Beispiel: $\langle \mathbb{B}, \underline{\text{ and }}, \underline{\text{ or }}, \neg, \underline{\text{true}}, \underline{\text{false}} \rangle$
  - $\underline{\text{true or }} b = \underline{\text{true}}$
  - $\underline{\text{false and }} b = \underline{\text{false}}$
  - $\neg\neg b = b$
- Ziel algebraischer Transformationen:
  - ersetze teure Anweisungen durch billigere
  - lokale, kontextunabhängige Ersetzung
  - keine Gesamtanalyse von π erforderlich

Peephole-Optimierung:

- Odee: Schiebe ein kleines Fenster über das Programm und optimiere innerhalb des Fensters
- Beispiel:
  - $y \leftarrow x + a; z \leftarrow y - a; \rightarrow y \leftarrow x + a; z \leftarrow x;$
  - $x \leftarrow x + 0;$ weglassen
  - $\begin{array}{l} u \leftarrow x + y; \\ v \leftarrow y + z; \\ w \leftarrow v + x; \end{array} \Big| \begin{array}{l} u \leftarrow x + y; \\ w \leftarrow u + z; \end{array}$

    $w = (y + z) + x = (x + y) + z$, dadurch wird v nicht mehr benötigt.

Reduction in Strength Ersetze $x \leftarrow y * *2$ durch $x \leftarrow y * y;$ und $x \leftarrow 2 * y$ durch $x \leftarrow y + y;$.

Optimalität von SLC-Programmen

- Spezialfall: $\mathfrak{A} = \langle \mathbb{Z}; +, * \rangle$, keine Konstanten
- Kostenmaß: Programmlänge

Dann gilt für ein $\pi \in SLC$: Es existiert ein optimales Programm π_{opt} für π, d.h. $\pi_{opt} \sim_{\mathfrak{A}} \pi$ und $c(\pi_{opt}) \leq c(\pi')$ für alle π' mit $\pi' \sim_{\mathfrak{A}} \pi$.

- Konstruktion von π_{opt}: NP-vollständig (?)
- optimaler Code für arithmetische Ausdrücke
- Polynomberechnung (Folie 2.7, Optimierung von Polynomen: Horner-Regel), Spezialfall: $g(x) = ax^{16} + bx^8 + cx^4 + dx^2 + ex + f$
  - DAG-optimiertes π mit 36 Anweisungen, 31 Multiplikationen
  - Möglich: 14 Anweisungen, 9 Multiplikationen

3 Ordnungstheoretische Grundlagen

Semantik iterativer und rekursiver Programme ebenso wie Datenflußanalyse, beschreibbar als Lösung von Gleichungssystemen; dazu:

- vollständige Halbordnungen
- vollständige Verbände
- monotone und stetige Funktionen
- Fixpunktberechnungen

Definition (Halbordnung): Sei D eine Menge und $\leq \subseteq D \times D$. Dann heißt $\mathcal{D} = \langle D; \leq \rangle$ eine Halbordnung mit der Halbordnungsrelation $\leq$, falls für alle $a, b, c \in D$ gilt:

- $a \leq a$ (reflexiv)
- $a \leq b, b \leq c \curvearrowright a \leq c$ (transitiv)
- $a \leq b, b \leq a \leftrightarrow a = b$ (antisymmetrie)

Sei $a \in D$ und $T \subseteq D$. Dann heißt

- a **obere Schranke** von T, falls $T \leq a$
- a **kleinstes Element** von T, falls $a \in T$ und $a \leq T$
- a **kleinste obere Schranke (Supremum)** von T, falls $T \leq A$ und $T \leq a' \curvearrowright a \leq a'$; Bezeichnung: $a = \sqcup T$ ("join", "lub" (least upper bound))
  Beachte: Kleinste Elemente, insbesondere $\sqcup T$, sind eindeutig bestimmt
- Besitzt $\mathcal{D}$ ein kleinstes Element, so wird es durch $\bot$ ("bottom") bezeichnet.
- $\gamma : \mathbb{N} \to D$ heißt **Kette**, falls $\gamma(i) \leq \gamma(i+1)$ für alle $i \in \mathbb{N}$ (aufsteigende ω-Kette); Bezeichnung: $(\gamma(i) | i \in \mathbb{N}) = \gamma$
- Eine Halbordnung $\mathcal{D}$ heißt **vollständige Halbordnung**, falls $\mathcal{D}$ ein kleinstes Element hat und jede Kette γ eine kleinste obere Schranke besitzt: $\sqcup \gamma := \sqcup \{\gamma(i) | i \in \mathbb{N}\}$
- $\mathcal{D}$ heißt **vollständiger Verband**, falls jede Teilmenge $T \subseteq D$ eine kleinste obere Schranke $\sqcup T$ besitzt (Beachte: $\bot = \sqcup \varnothing$).
  Schreibweise: $a \sqcup b := \sqcup \{a, b\}$

Beispiel:

1. Seien A und B Mengen. Die Menge $PF(A, B) := \{f | f : A \dashrightarrow B\}$ der **partiellen Funktionen** von A nach B ist halbgeordnet durch $\leq$ mit $f \leq g :\leftrightarrow f(a) \in B \curvearrowright f(a) = g(a) \forall a \in A$, oder: $\underline{\text{graph}}(f) \subseteq \underline{\text{graph}}(g)$.
   Die "leere" Funktion $f_\varnothing$ mit $\underline{\text{Def}}(f_\varnothing) = \varnothing$ ist kleinstes Element von $\langle PF(A, B); \leq \rangle$. Jede Kette $f_0 \leq f_1 \leq \ldots$ besitzt eine kleinste obere Schranke $f = \sqcup \{f_i | i \in \mathbb{N}\}$: $f(a) = b \in \mathbb{B} \leftrightarrow \exists i \in \mathbb{N} : f_i(a) = b$.
   Also ist $\langle PF(A, B); \leq \rangle$ eine vollständige Halbordnung (cpo = complete partial order).
   - Spezialfall: $PF(A) := PF(A, A)$
   - Anwendung: Semantik iterativer Programme
2. Sei A eine Menge. Die Potenzmenge $\mathcal{P}(A) := \{T | T \subseteq A\}$ ist halbgeordnet durch $\subseteq$ und bildet mit $\subseteq$ einen vollständigen Verband: Für $\mathfrak{T} \subseteq \mathcal{P}(A)$ gilt $\sqcup \mathfrak{T} = \bigcup \{T | T \in \mathfrak{T}\}$.
   - Anwendung: Datenflußanalyse (DFA)
   - Beachte: Ist $\mathcal{D} = \langle D, \leq \rangle$ eine Halbordnung, so auch $\mathcal{D}' := \langle D, \geq \rangle$, die duale Halbordnung von D.

Definition und Lemma (Produkt von Halbordnungen) Sind $\mathcal{D}_i = \langle D_i; \leq_i \rangle$ für $i = 1, 2$ Halbordnungen, so auch

$$\mathcal{D}_1 \times \mathcal{D}_2 := \langle D_1 \times D_2, \leq \rangle$$

mit

$$(a_1, a_2) \leq (b_1, b_2) :\leftrightarrow a_i \leq_i b_i \text{ für } i = 1, 2.$$

Definition und Lemma (Funktionenraum): Sei A eine Menge. Ist $\mathcal{D} = \langle D, \leq \rangle$ eine Halbordnung, so auch $F(A, \mathcal{D}) := \langle F(A, D); \leq \rangle$ mit $F(A, D) := \{f | f : A \to D\}$ und $f \leq g :\leftrightarrow f(a) \leq g(a) \forall a \in A$.

- Folgerung: Sind $\mathcal{D}$, $\mathcal{D}_1$ und $\mathcal{D}_2$ vollständige Halbordnungen bzw. vollständige Verbände, so auch $\mathcal{D}_1 \times \mathcal{D}_2$ und $F(A, \mathcal{D})$.
- Insbesondere gilt für Ketten:
  - $\sqcup\{a_1^{(i)}, a_2^{(i)} | i \in \mathbb{N}\} = (\sqcup\{a_1^{(i)} | i \in \mathbb{N}\}, \sqcup\{a_2^{(i)} | i \in \mathbb{N}\})$
  - $(\sqcup\{f^{(i)} | i \in \mathbb{N}\})(a) = \sqcup\{f^{(i)}(a) | i \in \mathbb{N}\}$

  und für beliebige Teilmengen
  - $\sqcup T = (\sqcup \underline{\text{proj}}_1(T), \sqcup \underline{\text{proj}}_2(T))$ mit $T \subseteq D_1 \times D_2$.
  - $(\sqcup T)(a) = \sqcup T(a)$ mit $T \subseteq F(A, D)$.

Definition (monotone und stetige Funktionen) Seien $\mathcal{D}_i = \langle D_i; \leq_i \rangle$ für $i = 1, 2$ Halbordnungen und $f : D_1 \to D_2$:

- f heißt **monoton**, falls $a \leq_1 b \curvearrowright f(a) \leq_2 f(b)$ für alle $a, b \in D_1$;
- f heißt **stetig**, falls $\mathcal{D}_1$ und $\mathcal{D}_2$ vollständig sind, f monoton ist und für jede Kette $(a_i | i \in \mathbb{N})$ von $\mathcal{D}_1$ gilt: $f(\sqcup\{a_i | i \in \mathbb{N}\}) = \sqcup\{f(a_i) | i \in \mathbb{N}\}$
  - Beachte: f monoton $\curvearrowright (f(a_i) | i \in \mathbb{N})$ Kette

Definition (ACC): $\mathcal{D}$ besitzt die ACC-Eigenschaft (ACC = ascending chain condition), wenn gilt: Für jede Kette $(a_i | i \in \mathbb{N})$ ist $\{a_i | i \in \mathbb{N}\}$ endlich.

- Folgerung:
  1. Eine Halbordnung mit ACC ist vollständig.
  2. Monotone Abbildungen auf Halbordnungen mit ACC sind stetig.

Definition (Distributivität): Sind $\mathcal{D}_1$ und $\mathcal{D}_2$ vollständige Verbände, so heißt $f : D_1 \to D_2$ distributiv, falls $f(a \sqcup b) = f(a) \sqcup f(b)$ für alle $a, b \in D_1$ gilt.

Lemma: Seien $\mathcal{D}_1$ und $\mathcal{D}_2$ vollständige Verbände. Dann gilt: $f : D_1 \to D_2$ distributiv $\curvearrowright f$ monoton.

Beweis: $a \leq b \curvearrowright b = a \sqcup b \curvearrowright f(b) = f(a \sqcup b) = f(a) \sqcup f(b) \curvearrowright f(a) \leq f(b)$

- Folgerung: Besitzen $\mathcal{D}_1$ und $\mathcal{D}_2$ auch ACC, so gilt: $f : D_1 \to D_2$ distributiv $\curvearrowright f$ monoton $\curvearrowright f$ stetig

Fixpunktsatz: Sei $\mathcal{D} = \langle D; \leq \rangle$ eine vollständige Halbordnung und $f : D \to D$ stetig. Dann gilt:

1. $(f^i(\bot) | i \in \mathbb{N})$ ist eine Kette von $\mathcal{D}$.
2. $a := \sqcup\{f^i(\bot) | i \in \mathbb{N}\}$ ist Fixpunkt von f, d.h. $a = f(a)$.
3. a ist kleinster Fixpunkt von f, Bezeichnung: $\underline{\text{fix}}(f)$.

Beweis:

1. $\bot \leq f(\bot) \leq f^2(\bot) \leq \ldots$ (da f monoton ist)
2. $f(a) = f(\sqcup\{f^i(\bot)|i \in \mathbb{N}\}) = \sqcup\{f^{i+1}(\bot)|i \in \mathbb{N}\} = \sqcup\{f^i(\bot)|i \in \mathbb{N}\} = a$
3. $b \in f(b)$. Aus $\bot \leq b$ folgt $f(\bot) \leq b$, $f^2(\bot) \leq b \ldots \curvearrowright b$ ist obere Schranke von $\{f^i(\bot)|i \in \mathbb{N}\} \curvearrowright a \leq b$.

- Folgerung: Ist $\mathcal{D}$ eine Halbordnung mit ACC und $f : D \to D$ monoton. Dann gilt: f besitzt einen kleinsten Fixpunkt $\underline{\text{fix}}(f)$ und es existiert $m \in \mathbb{N}$, sodaß $\underline{\text{fix}}(f) = f^m(\bot)$.

4 Datenflußanalyse und Optimierung iterativer Programme

[2004/11/10]

Zwischencode für eine abstrakte Maschine:

- SLC + Verzweigung und Iteration
- Bedingte und unbedingte Sprünge als einzige Kontrollstrukturen

Inhalt:

- Syntax und Semantik von IC (Iterativer Code)
- Basisblock-Darstellung von IC-Programmen
- Abstrakte Interpretation und Datenflußanalyse (DFA)
- Verallgemeinerung der Analyse und Optimierung von SLC-Programmen auf IC-Programmen

4.1 Syntax von IC

Erweiterung von SLC

$V = \{x, y, \ldots\}$	Variablen
$C = \{c, d, \ldots\}$	Konstantensymbole
$\Sigma = \bigcup_{r \geq 1} \Sigma^{(r)}$	Operationssymbole (evtl. $C = \Sigma^{(0)}$)
$\Pi = \bigcup_{r \geq 0} \Pi^{(r)}$	Prädikatsymbole
$L = \{l_i \mid i \in \mathbb{N}\}$	Sprungmarken ("Label")

Σ und Π endlich.

Anw Anweisungen:

- Wertzuweisungen: $v \leftarrow e$ mit $v \in V$ und $e \in V \cup C \cup \{f(u_1, \ldots, u_r) | f \in \Sigma^{(r)}, u_i \in V \cup C\}$
- Sprunganweisungen: goto l mit $l \in L$, if $p(u_1, \ldots, u_r)$ goto l mit $p \in \Pi^{(r)}$, $u_i \in V \cup C, l \in L$
- Markierte Anweisungen: $l : \alpha$ mit $l \in \alpha$ und α unmarkierte Anweisung
- Beispiel: Folie 4.1: "IC-Beispielprogramm: Fakultätsberechnung"

IC-Programme:

$\pi = (\text{Vlist}, \text{Alist}) \in IC :\leftrightarrow \text{Vlist in } x_1, \ldots, x_n; \text{ out } y_1, \ldots, y_m; \text{ loc } z_1, \ldots, z_p;$

mit

- $IV := \{x_1, \ldots, x_n\}$ Eingabevariablen ($n \geq 0$)
- $OV := \{y_1, \ldots, y_m\}$ Ausgabevariablen ($m \geq 1$)
- $LV := \{z_1, \ldots, z_p\}$ lokale Variablen ($p \geq 0$)
- $V_\pi = IV \cup OV \cup LV$ Programmvariablen

- Alist $= \alpha_1; \ldots; \alpha_q$ (Anweisungsliste) mit
  - $\alpha_i \in \underline{\text{Anw}}$, $V_{\alpha_i} \subseteq V_\pi$, $q \geq 0$
  - eindeutige Sprungziele: $\alpha_i = l : \ldots, \alpha_j = l' : \ldots, i \neq j \curvearrowright l \neq l'$
- Beispiel: Folie 4.2: "Beispiel: Flußdiagramm und -graph"

π erfüllt die Variablenbedingung: Jede Programmvariable wird vor ihrer Benutzung definiert (Beachte: Variablen in bedingten Sprunganweisungen werden dort benutzt).
π heißt **standardmarkiert**, falls $\alpha_i = i : \alpha_i'$ für $i = 1, \ldots, q$ und $L_\pi \subseteq \{1, \ldots, q+1\}$ Marken von π

Beispiel (Folie 4.1): π_f berechnet die Fakultät

4.2 Semantik von IC

Sei $\pi = (\text{Vlist}, \text{Alist}) \in IC$, o.B.d.A. sei π standardmarkiert.

Das Flußdiagramm von π (Semantik der Kontrolle)

Alist $= 1 : \alpha_1; \ldots; q : \alpha_q$

$i :$ $\underline{\text{goto}}$ $j \mapsto i \to j$

$$i : \underline{\text{if}}\ be\ \underline{\text{goto}}\ j \mapsto \begin{array}{ll} \overset{1}{\to} \cdot & j \\ \overset{0}{\to} \cdot & i+1 \end{array}$$

Das Flußdiagramm von π ergibt sich durch Verkleben der entsprechenden Teilgraphen (lokalen Variablen ergeben sich indirekt ...).

Fixpunktsammlung Sei $\mathfrak{A} = \langle A; \varphi \rangle$ eine Interpretation für π, d.h.

- $\varphi(f^{(n)}) = f_{\mathfrak{A}} : A^n \to A$,
- $\varphi(p^{(n)}) = P_{\mathfrak{A}} : A^n \to \{0, 1\}$,
- $V_\pi := \{v_1, \ldots, v_s\}$.
- Zustandsraum $Z := A^s \bar{a} = (a_1, \ldots, a_s)$ als Wert des Variablenvektors $\bar{v} = (v_1, \ldots, v_s)$.

Für die **Fortsetzungsfunktionen** ("Continuations") $\varphi_i : Z \dashrightarrow Z (i = 1, \ldots, q+1)$, die von Programmarke i bis zum Programmende berechneten Zustandstransformationen, gelten folgende Gleichungen:

- $\alpha = v_j \leftarrow e \curvearrowright \varphi_i(\sigma) = \varphi_{i+1}(\sigma[j/[\![e]\!]\sigma])$
- $\alpha_i =$ $\underline{\text{goto}}$ $l \curvearrowright \varphi_i(\sigma) = \varphi_l(\sigma)$
- $\alpha_i = \underline{\text{if}}\ be\ \underline{\text{goto}}\ l \curvearrowright \varphi_i(\sigma) = \underline{\text{if}}\ [\![be]\!]\sigma\ \underline{\text{then}}\ \varphi_l(\sigma)\ \underline{\text{else}}\ \varphi_{i+1}(\sigma)$

[2004/11/12]

Fixpunktsemantik für $\pi \in IC$

- $\pi = (\text{Vlist}, \text{Alist})$, $\mathfrak{A} = \langle A; \varphi \rangle$
- $V_\pi = \{v_1, \ldots, v_s\}$, $Z := A^s$
- $\bar{a} = (a_1, \ldots, a_s) \in Z$
- Alist $= 1 : \alpha_1; \ldots; q : \alpha_q$

- Fortsetzungsfunktionen: $1 \leq i \leq q$
  - $\alpha_i = v_j \leftarrow e \qquad \varphi_i(\bar{a}) = \varphi_{i+1}(\bar{a}[j/[\![e]\!]\bar{a}])$
  - $\alpha_i = \underline{\text{ goto } l} \qquad \varphi_i = \varphi_e(\bar{a})$
  - $\alpha_i = \underline{\text{if } be \text{ goto } l}$
    $\varphi_i(\bar{a}) = \underline{\text{if } [\![be]\!]\bar{a} \text{ then } \varphi_l(\bar{a}) \text{ else } \varphi_{i+1}(\bar{a})}$ und $\varphi_{q+1}(\bar{a}) = \bar{a}$. Die $\varphi_1, \ldots, \varphi_{q+1} : A^s \dashrightarrow A^s$ bilden daher die Lösung eines Gleichungssystems:
    - $F_i(\bar{v}) = \tau_i (i = 1, \ldots, q+1)$ mit

$$\tau_i = \begin{cases} F_{i+1}(\bar{v}[v_j/e]) & \text{falls } \alpha_i = v_j \leftarrow e \\ F_l(\bar{v}) & \text{falls } \alpha_i = \underline{\text{ goto } l} \\ \underline{\text{if } be \text{ then }} F_l(\bar{v}) \underline{\text{ else }} F_{i+1}(\bar{v}) & \text{falls } \alpha_i = \underline{\text{if } be \text{ goto } l} \end{cases}$$

      $\tau_{q+1} = \bar{v}$.

Berücksichtigung von Ein/Ausgabe: Sei Vlist = $\underline{\text{ in }} x_1, \ldots, x_n; \underline{\text{ out }} y_1, \ldots, y_n; \underline{\text{ loc }} z_1, \ldots, z_p$,

- $\bar{x} = (x_1, \ldots, x_n)$,
- $(\bar{x}, \bar{a}_0) = (x_1, \ldots, x_n, a_0, \ldots, a_0)$
- $\bar{y} = (y_1, \ldots, y_m)$

Das Gleichungssystem (E_π) ist dann definiert durch:

$$(E_\pi) = \begin{cases} F(\bar{x}) = F_1(\bar{x}, \bar{a}_0) & a_0 \in A \text{ beliebig} \\ F_i(\bar{v}) = \tau_i & (i = 1, \ldots, q) \\ F_{q+1}(\bar{v}) = \bar{y} \end{cases}$$

Lösungstyp:

$$f : A^n \dashrightarrow A^m, \quad f_i : A^s \dashrightarrow A^m, \quad i = 1, \ldots, q+1$$

Beispiel: Fakultätsprogramm

$$(E_\pi): \begin{array}{ll} F(x) & = F_1(x, 12, 12) \\ F_1(x,y,z) & = F_2(x,y,0) \\ F_2(x,y,z) & = F_3(x,1,z) \\ F_3(x,y,z) & = \underline{\text{if } x = z \text{ then }} F_7(x,y,z) \underline{\text{ else }} F_4(x,y,z) \\ F_4(x,y,z) & = F_5(x,y,z+1) \\ F_5(x,y,z) & = F_6(x, y*z, z) \\ F_6(x,y,z) & = F_3(x,y,z) \\ F_7(x,y,z) & = y \end{array}$$

Vereinfachung:

$$\begin{array}{ll} F(x) & = F_3(x,1,0) \\ F_3(x,y,z) & = \underline{\text{if } x = z \text{ then } y \text{ else }} F_3(x, y*(z+1), z+1) \end{array}$$

$\rightsquigarrow$ Endrekursive Funktionsgleichungen ("tail recursive"): Funktionsaufrufe nur in Termspitze, keine Rücksprünge

1. Reduktionssemantik (operationale Semantik)

   $$F(3) \to F_3(3,1,0) \to F_3(3,1,1) \to F_3(3,2,2) \to F_3(3,6,3)) \to 6$$

2. Fixpunktsemantik: Der Funktionenraum (Lösungsraum)

   $$FR_\pi := PF(A^n, A^m) \times PF(A^s, A^m)^{q+1}$$

   ist eine vollständige Halbordnung. Die **Gleichungstransformation** ("Einsetzungsfunktional")

   $$T_\pi : FR_\pi \to FR_\pi$$

ist wie folgt definiert: Für $\bar{\varphi} = (\varphi, \varphi_1, \ldots, \varphi_{q+1}) \in FR_\pi$ und $\bar{F} = (F, F_1, \ldots, F_{q+1})$ bezeichne $\tau_i[\bar{F}/\bar{\varphi}]$ die Ersetzung der Funktionsvariablen aus $\bar{F}$ durch die entsprechenden Funktionen aus $\bar{\varphi}$. Damit

$$T_\pi(\bar{\varphi}) := (\lambda\bar{x}.\varphi_1(\bar{x}, \bar{a}_0), \lambda\bar{v}.\tau_1[\bar{F}/\bar{\varphi}], \ldots \lambda\bar{v}.\tau_q[\bar{F}/\bar{\varphi}], \lambda\bar{v}.\bar{y})$$

Beachte: $\bar{\varphi}$ ist Lösung von E_π gdw. $\bar{\varphi} = T_\pi(\bar{\varphi})$ T_π ist stetig und besitzt daher einen kleinsten Fixpunkt $\underline{\text{fix}}(T_\pi)$. Seine erste Komponente $[\![\pi]\!]_{\underline{\text{fix}}} := \underline{\text{proj}}_1(\underline{\text{fix}}(T_\pi)) : A^n \dashrightarrow A^m = \underline{\text{proj}}(\sqcup\{T_\pi^i(\bot) | i \in \mathbb{N}\})$ mit $\bot = (F_\varnothing, f_{1\varnothing}, \ldots, f_{q+1\varnothing})$ heißt Fixpunktsemantik von π.

[2004/11/17]

Beispiel (Fakultätsprogramm):

$F(x) = G(x, 1, 0)$
$G(x, y, z) = \underline{\text{if}}\ x = z\ \underline{\text{then}}\ y\ \underline{\text{else}}\ G(x, x * (z + 1), z + 1)$

$$\begin{array}{lcl}
FR_{\pi_f} & = & PF(\mathbb{Z}, \mathbb{Z}) \times PF(\mathbb{Z}^3, \mathbb{Z}) \\
\bot & = & (f_\varnothing = \lambda x., g_\varnothing = \lambda(x, y, z).) \\
T(\bot) & = & (-, \lambda(x, y, z).\underline{\text{if}}\ x = z\ \underline{\text{then}}\ y) \\
T^2(\bot) & = & (\lambda x.\underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 1, \\
 & & \lambda(x, y, z).\underline{\text{if}}\ x = z\ \underline{\text{then}}\ y \\
 & & \underline{\text{if}}\ x = z + 1\ \underline{\text{then}}\ y * (z + 1)) \\
T^3(\bot) & = & (\lambda x.\underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 1 \\
 & & \underline{\text{if}}\ x = 1\ \underline{\text{then}}\ 1, \\
 & & \lambda(x, y, z).\underline{\text{if}}\ x = z\ \underline{\text{then}}\ y \\
 & & \underline{\text{if}}\ x = z + 1\ \underline{\text{then}}\ y * (z + 1) \\
 & & \underline{\text{if}}\ x = z + 2\ \underline{\text{then}}\ y * (z + 1) * (z + 2)) \\
 & \vdots & \\
T^{n+1}(\bot) & = & (\lambda x.\underline{\text{if}}\ x \leq n - 1\ \underline{\text{then}}\ x!, \\
 & & \lambda(x, y, z).\underline{\text{if}}\ x = z\ \underline{\text{then}}\ y \\
 & & \underline{\text{if}}\ x = z + 1\ \underline{\text{then}}\ y * (z + 1) \\
 & & \underline{\text{if}}\ x = z + n\ \underline{\text{then}}\ y * (z + 1) * (z + 1) * \ldots * (z + n))
\end{array}$$

Also: $[\![\pi]\!]_{\underline{\text{fix}}}(x) = \underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ x!$

Folgerungen der Semantik von IC-Programmen

1. Eine Anweisung kann wiederholt werden, evtl. unendlich oft (Programmschleifen).
2. Ein **Programmpfad** (ein eingabeunabhängiger Pfad im Flußdiagramm vom Eingabe- zum Ausgabeknoten) muß kein **Berechnungspfad** (eingabeabhängiger Pfad einer Berechnung) sein.
3. Turingmaschinen sind als IC-Programme auffaßbar.
4. Die Programmäquivalenz ist nicht entscheidbar. $\rightsquigarrow$

Programmanalyse: Approximation von Programmeigenschaften, Prädikate ignorieren, Programmpfade betrachten

- Die Menge der Programmpfade von $\pi \in IC$ ist regulär.
- Die Menge der Berechnungspfade is i.a. rekursiv aufzählbar, aber nicht entscheidbar.

$\Rightarrow$ sichere, aber evtl. unvollständige Optimierungsinformationen

4.3 Flußgraphen von IC-Programmen

Idee: Die deterministische Verzweigung wird durch nichtdeterministische Auswahl ersetzt; Programmpfade statt Berechnungspfade.

Sei $\pi = \langle \text{Vlist}, \text{Alist} \rangle \in IC$ und $\text{Alist} = 1 : \alpha_1; \ldots; q : \alpha_q$.

1. SI-Graph (SI = single instruction)
   - $G_\pi^{SI} = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$
   - $\underline{\text{Kno}} := \{\alpha_i | \alpha_i = x \leftarrow e, 1 \leq i \leq q\} \cup \{be | \alpha_i = \underline{\text{if}}\ be\ \underline{\text{goto}}\ j, 1 \leq i, j \leq q\}$
   - $s = \alpha_1$ (o.B.d.A. α_1 Wertzuweisung)
   - $\underline{\text{Kan}}$ wie im Flußdiagram
2. BB-Graph (BB = basic block)
   - Ziel: Vereinfachung der Programmanalyse
   - Idee: Zerlegung von π in maximale SLC-Blöcke
   - Konstruktion von $G_\pi^{BB} = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$:
     - α_i heißt **Blockanfang**, falls eine der folgenden Bedgingungen gilt:
       * $i = 1$
       * i ist Sprungziel von π (kommt als $\underline{\text{goto}}\ i$ in π vor)
       * α_{i-1} ist Sprunganweisung

       Ein Blockanfang α_i bestimmt $\alpha_i; \alpha_{i+1}; \ldots; \alpha_j$ als längste Anweisungsfolge ohne weiteren Blockanfang. Weglassen von $\underline{\text{goto}}\ k$ und Ersetzen von $\underline{\text{if}}\ be\ \underline{\text{goto}}\ k$ durch be ergibt den Basisblock $\alpha'_{i_1}; \ldots; \alpha'_{i_k}$

       Beispiel: π_f (Fakultätsprogramm) Blockanfänge: $\alpha_1, \alpha_3, \alpha_4$
       $\beta_1 = \alpha_1; \alpha_2$
       $\beta_2 = \alpha_3;$
       $\beta_3 = \alpha_4; \alpha_5$

Die nachfolgende Korrektur ist notwendig, da in der obigen Weise gleiche Anweisungen α_i nicht im Graphen unterschieden werden könnten.

Flußgraphen von IC-Programmen (Korrektur)

1. SI-Graphen
   - $G_\pi^{SI} = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ und $\underline{\text{lab}} : \underline{\text{Kno}} \rightarrow$ Wertzuweisungen $\cup$ Boolean Expressions $\cup\ \underline{\text{START}}$
   - $\underline{\text{Kno}} := \{i | 0 \leq i \leq q, \alpha_i \neq \underline{\text{goto}}\ j\}$
   - $\underline{\text{lab}}(0) = \underline{\text{START}}$
   - $\underline{\text{lab}}(i) = \begin{cases} \alpha_i & \text{falls } \alpha_i = x \leftarrow e \\ be & \text{falls } \alpha_i = \underline{\text{if}}\ be\ \underline{\text{goto}}\ j \end{cases} \qquad 1 \leq i \leq q, s := 0$
   - $\underline{\text{Kan}}$ folgen aus Flußdiagramm von π
2. BB-Graph
   - $G_\pi^{BB} = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$
   - s letzte Stunde
   - $\underline{\text{Kno}}$ = Basisblöcke + Startblock
3. Flußgraphen für die Rückwärtsanalyse
   - $G_{\overleftarrow{\pi}}^{SI}, G_{\overleftarrow{\pi}}^{BB}$ Kanten invertieren, STOP-Knoten des Flußdiagrammes als Startknoten

4.4 Datenflußanalyse-Systeme

2 Komponenten:

- ein **Flußgraph** als Abstraktion eines Flußdiagrammes (Programmpfade)
- eine **abstrakte Interpretation**: Analyseinformation als Element eines vollständigen Verbandes mit ACC, Informationstransformationen als monotone Abbildungen

Definition (Flußgraph): Sei $\underline{\text{Kno}}$ eine nicht-leere endliche Menge von **Knoten**, $\underline{\text{Kan}} \subseteq \underline{\text{Kno}} \times \underline{\text{Kno}}$ eine Menge von **Kanten** und $s \in \underline{\text{Kno}}$ ein **Startknoten**.
Für $k \in \underline{\text{Kno}}$ bezeichne

$$\underline{\text{Vor}}(k) := \{k' \in \underline{\text{Kno}} | (k', k) \in \underline{\text{Kan}}\}$$

die Menge der **(direkten) Vorgängerknoten**. Für $k, k' \in \underline{\text{Kno}}$ bezeichne

$$\underline{\text{Pfad}}[k, k') := \{(k_1, \ldots, k_r) | k_1 = k, k_r \in \underline{\text{Vor}}(k') \wedge (k_i, k_{i+1}) \in \underline{\text{Kan}} \text{ für } i = 1, \ldots, r-1\}$$

die **Menge der Pfade** von k zu einem direkten Vorgängerknoten von k'.
$G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ heißt **Flußgraph**, falls $\underline{\text{Vor}}(s) = \varnothing$.
Bemerkung: Für $\pi \in IC$ sind $G_\pi^{SI}, G_\pi^{BB}, G_{\overleftarrow{\pi}}^{SI}, G_{\overleftarrow{\pi}}^{BB}$ Flußgraphen.

Definition (Abstrakte Interpretation eines Flußgraphen): Sei $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ ein Flußgraph. Sei $\mathcal{D} = \langle D; \leq \rangle$ ein vollständiger Verband mit ACC, $d_s \in D$ eine **Startinformation** und

$$\varphi : \underline{\text{Kno}} \to \{f | f : D \to D, f \text{ monoton}\}$$

mit $\varphi(s) = \text{id}_D$, Schreibweise: $\varphi_k(d) = \varphi(k)(d)$
Dann heißt $\mathfrak{I} = \langle \mathcal{D}, \varphi, d_s \rangle$ eine **abstrakte Interpretation von** G.

Definition (DFA-System): Ist G ein Flußgraph und $\mathfrak{I}$ eine abstrakte Interpretation von G, so heißt $\Delta = \langle G, \mathfrak{I} \rangle$ ein **DFA-System** (monotone framework).
Ein DFA-System bestimmt in natürlicher Weise eine MOP-Lösung (meet over all paths) sowie eine MFP-Lösung (maximal fixed point).

Definition (Die MOP-Lösung eines DFA-Systems): Sei $\Delta = \langle G, \mathfrak{I} \rangle$ ein DFA-System. Für $k \in \underline{\text{Kno}}$ und $p = (k_1, \ldots, k_r) \in \underline{\text{Pfad}}[s, k)$ definieren wir die **Analyseinformation** von k bzgl p

$$AI_k^p := \varphi_{k_r}(\varphi_{k_{r-1}} \ldots \varphi_{k_1}(d_s) \ldots)$$

Dann ist die **MOP-Lösung für** k die folgende **Analyseinformation**

$$AI_k^{MOP} := \sqcup \{AI_k^p | p \in \underline{\text{Pfad}}[s, k)\} \text{ für } k \neq s$$

und $AI_s^{MOP} := d_s$. $AI^{MOP} := (AI_k^{MOP} | k \in \underline{\text{Kno}})$ heißt **MOP-Lösung von** Δ.
Bemerkung: Die MOP-Lösung ist i.a. nicht berechenbar.

Wiederholung: $\Delta = \langle G, \mathfrak{I} \rangle$ DFA-System

- $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ Flußgraph,
  $\underline{\text{Vor}}(k)$ direkte Vorgängerknoten
  $\underline{\text{Vor}}(s) = \varnothing$
  $\underline{\text{Pfad}}[k, k')$
- $\mathfrak{I} = \langle \mathcal{D}, \varphi, d_s \rangle$ abstrakte Interpretation von G
  $\mathcal{D} = \langle D; \leq \rangle$ vollständiger Verband mit ACC
  $d_s \in D$ Startinformation
  $\varphi : \underline{\text{Kno}} \to \{f | f : D \to D, f \text{ monoton}\}$
  $\varphi_s = \text{id}_D \quad \varphi_k := \varphi(k)$

Definition (Die MFP-Lösung eines DFA-Systems): Sei $\Delta = \langle G, \mathfrak{I} \rangle$ ein DFA-System. Δ bestimmt ein System E_Δ von Datenflußgleichungen:

$$(E_\Delta) \begin{cases} X_s = d_s \\ X_k = \sqcup\{\varphi_{k'}(X_{k'}) | k' \in \underline{\text{Vor}}(k)\}, \quad k \in \underline{\text{Kno}} \setminus \{s\} \end{cases}$$

[2004/11/24]

E_Δ bestimmt eine **Gleichungstransformation**:

$$T_\Delta : D^n \to D^n$$
$$T_\Delta(a_1, \ldots, a_n) := (d_\Delta, \sqcup\{\varphi_i(a_i) | i \in \underline{\text{Vor}}(2)\}, \ldots, \sqcup\{\varphi_i(a_i) | i \in \underline{\text{Vor}}(n)\})$$

und die **MFP-Lösung von Δ**:

$$AI^{MFP} := \underline{\text{fix}}(T_\Delta) \in D^n$$

Beachte: $\mathcal{D} = \langle D, \leq \rangle$ vollständiger Verband mit ACC, $\varphi_i : D \to D$ monoton $\curvearrowright \varphi_i$ stetig $\curvearrowright T_\Delta$ stetig und wegen ACC existiert ein $m \in \mathbb{N}$ mit

$$\underline{\text{fix}}(T_\Delta) = T_\Delta^m(\bot) \in D^n.$$

Also: AI^{MFP} in endlichen vielen Schritten **berechenbar**.

Satz (Correctness-Theorem): In einem DFA-System Δ gilt:

$$AI^{\text{eigentlich}} \leq AI^{MOP} \leq AI^{MFP}$$

Satz: $AI^{MOP} \leq AI^{MFP}$

Beweis: Es genügt zu zeigen, daß für alle $1 < k \leq n, p \in \underline{\text{Pfad}}[1, k)$ und $a = (a_1, \ldots, a_n)$ mit $A = T_\Delta(a)$ gilt:

$$(*) \quad AI_k^p \leq a_k$$

- $(*) \curvearrowright$ Behauptung: $A_1^{MOP} = d_s = AI_1^{MFP}$
  $k \neq 1$: $AI_k^p \leq AI_k^{MFP}$ nach $(*)$ wegen Fixpunkteigenschaft von AI^{MFP}, also auch $AI_k^{MOP} \leq AI_k^{MFP}$.
- **Beweis von** $(*)$**:** Aus (E_Δ) und $a = T_\Delta(a)$:

  $$(**) \quad \varphi_i(a_i) \leq a_k \text{ für } 1 < k \leq n \text{ und } i \in \underline{\text{Vor}}(k)$$

- Wir zeigen $(*)$ durch Induktion über die Länge r von p:
  - $r = 1$: Dann muß $p = 1 \in \underline{\text{Vor}}(k)$, sodaß nach $(**)$ $AI_k^p = \varphi_1(a_1) \leq a_k$
  - $r \sim r + 1$: Sei $p \in \underline{\text{Pfad}}[1, k)$ mit Länge $r + 1$. Dann ist $p = p'.i$ mit $i \in \underline{\text{Vor}}(k), i \neq 1$ (wegen $\underline{\text{Vor}}(1) = \varnothing$) und $p' \in \underline{\text{Pfad}}[1, i)$. Nach IV gilt $AI_i^{p'} \leq a_i$ und nach $(**)$ $\varphi_i(a_i) \leq a_k$. Aus beiden Ungleichungen und der Monotonie von φ_i folgt: $AI_k^p = \varphi_i(AI_i^{p'}) \leq \varphi_i(a_i) \leq a_k$

q.e.d.

Satz (Completeness Theorem, Coincidence Theorem): Sind in einem DFA-System Δ die Funktionen $\varphi_k : D \to D$ distributiv $(\varphi_k(a \sqcup b) = \varphi_k(a) \sqcup \varphi_k(b))$, so gilt:

$$AI^{MOP} = AI^{MFP}$$

Beweis: Es bleibt zu zeigen, daß

$$a = (a_1, \ldots, a_n) := (AI_1^{MOP}, \ldots, AI_n^{MOP})$$

ein Fixpunkt von T_Δ ist: $a = T_\Delta(a)$.

$$\begin{aligned}
T_\Delta(a) &= (d_s, \bigsqcup_{i \in \underline{\text{Vor}}(2)} \varphi_i(a_i), \ldots) \\
&= (d_s, \bigsqcup_{i \in \underline{\text{Vor}}(2)} \varphi_i(AI^{MOP}), \ldots) \\
&= (d_s, \bigsqcup_{i \in \underline{\text{Vor}}(2)} \varphi_i(\bigsqcup_{p \in \underline{\text{Pfad}}_{[1,i)}} AI_i^p), \ldots) \\
&= (d_s, \bigsqcup_{i \in \underline{\text{Vor}}(2)} \bigsqcup_{p \in \underline{\text{Pfad}}_{[i,i)}} \varphi_i(AI_i), \ldots \\
&= (d_s, \bigsqcup_{i \in \underline{\text{Vor}}(2), p \in \underline{\text{Pfad}}_{[i,i)}} AI_k^{p.i}) \\
&= (d_s, \bigsqcup_{p^i \in \underline{\text{Pfad}}_{[1,2)}} AI_2^{p'}, \ldots, \bigsqcup_{p \in \underline{\text{Pfad}}_{[1,n)}} AI_n^{p^i}) \\
&= (a_1, \ldots, a_n) \\
&= a
\end{aligned}$$

mit $AI_2^{p.i} = \varphi_i(AI_i^p)$,

$$(E_\Delta) \begin{cases} x_1 = d_s \\ x_2 = \bigsqcup_{i \in \underline{\text{Vor}}(k)} \varphi_i(X_i) \end{cases}$$

[2004/11/26]

Nachtrag (kleinster Fixpunkt): $F(x) = \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ F(x-1)$, Fortsetzungssemantik $\langle \lambda, -, 0 = \rangle$

1. Operationelle Semantik $a \in \mathbb{Z}$
   - $a \geq 0$: $F(a) \to F(a-1) \to \ldots \to F(0) \to 0$
   - $a < 0$: $F(a) \to F(a-1) \to \ldots$

   $[\![\pi]\!]_{op} = \lambda x.\underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ 0$
   $[\![\pi]\!]_{op}(x) = \underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ 0$

2. Fixpunkt-Semantik

   $\lambda x.$
   $\lambda x.\underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0$
   $\lambda x.\underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}$
   $\quad \underline{\text{if}}\ x = 1\ \underline{\text{then}}\ 0$
   $[\![\pi]\!]_{\underline{\text{fix}}} = \lambda x.\underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ 0$

   Weitere Fixpunkte: $\lambda x.\underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ 17$ – *beliebige Konstante*

Nachtrag zu Kapitel 3

Lemma Sei $\mathcal{D} = \langle D; \leq \rangle$ ein vollständiger Verband mit ACC und $f : D \to D$. Dann gilt: f distributiv $(f(a \sqcup b) = f(a) \sqcup f(b)) \curvearrowright f(\sqcup T) = \sqcup f(t) \quad \forall T \subseteq D, T \neq \varnothing$

Beweis: Sei $T \subseteq D, T \neq \varnothing$.

1. f distributiv $\curvearrowright$ f monoton $\curvearrowright$ $f(T) \leq f(\sqcup T) \curvearrowright \sqcup f(T) \leq f(\sqcup T)$
2. Bleibt zu zeigen: $f(\sqcup T) \leq \sqcup f(T)$.
   $(*)$ Es existiert eine endliche Teilmenge $T_e \subseteq T$ mit $\sqcup T \leq \sqcup T_e$.
   Konstruktion einer Kette in $D : a_0 \leq a_1 \leq a_2 \leq \ldots$, $a_0 \in T$ beliebig, $a_n \rightsquigarrow a_{n+1}$:
   a) Fall: $T \leq a_n, \quad a_{n+1} := a_n$, (also $a_{n+k} = a_n$)
   b) Fall: $T \not\leq a_n$. Dann existiert ein $b_n \in T$ mit $b_n \not\leq a_n$, $a_{n+1} := a_n \sqcup b_n \quad (a_n < a_{n+1})$

   ACC $\curvearrowright a_0 < a_1 < \ldots < a_n = a_{n+1} = a_{n+2} = \ldots$, $a_1 = a_0 \sqcup b_0 \sqcup b_1 \sqcup \ldots \sqcup b_n$, wobei $\{a_0, b_0, b_1, \ldots, b_n\} \in T$, aber $a_1 \notin b$ und $T \leq a_n$, somit $\sqcup T \leq a_1 = \sqcup(a_0 \sqcup b_1 \sqcup \ldots \sqcup b_n)$. Damit folgt die Behauptung:

$$\begin{aligned} f(\sqcup T) &\leq f(a_n) \\ &= f(a_0 \sqcup b_0 \sqcup \ldots \sqcup b_n) \\ &= f(a_0) \sqcup f(b_0) \sqcup \ldots \sqcup f(b_n) \\ &= \sqcup\{f(a_0), f(b_0), \ldots, f(b_n)\} \\ &\leq \sqcup f(T) \end{aligned}$$

Lemma (Doppelt induzierte Menge): Sei $\mathcal{D} = \langle D; \leq \rangle$ ein vollständiger Verband. $T = \{a_{ij} | i \in I, j \in J\} \subseteq D$,

$$\bigsqcup_{i \in I; j \in J} a_{ij} = \bigsqcup_{i \in I} \bigsqcup_{j \in J} a_{ij}$$

Beweis: trivial

[2004/12/01]

4.5 AE-Analyse und CS-Elimination für IC-Programme

- Ziel: Vermeidung wiederholter Auswertung von Ausdrücken
- Beachte: Ein Ausdruck ist für eine Anweisung verfügbar, wenn er auf jedem Berechnungspfad zu dieser Anweisung berechnet wird und die Werte seiner Variablen danach nicht verändert werden.
- AE-Analyse
  Zerlegung der Analyse mit Hilfe von Basisblöcken
  1. Globale Analyse für Basisblockgraph (BB-Graph)
  2. Lokale Analyse für Basisblöcke

  Aufgabe für (1): Konstruktion eines geeigneten DFA-Systems $\Delta = \langle G, \mathfrak{I} \rangle$ zu $\pi \in IC$.
  - G = BB-Graph für π für Vorwärtsanalyse
  - $\mathfrak{I} = \langle \mathcal{D}, \varphi, d_s \rangle$: Programmausdrücke:
    $$* \quad \underline{\text{OpExp}}_\pi := \{e | e = f(u_1, \ldots, u_r), \alpha_i = x \leftarrow e, r \geq 1\} \cup \{be | be = p(u_1, \ldots, u_r), \alpha_i = \underline{\text{if}}\ be\ \underline{\text{goto}}\ \ldots, r \geq 1\} = \{e_1, \ldots, e_t\}$$
    $* \ D := \mathbb{B}^t$ zur Darstellung von Teilmengen von $\underline{\text{OpExp}}_\pi$
    $* \ \bar{b} = (b_1, \ldots, b_t)$, $b_i = \begin{cases} 0 & \text{falls } e_i \text{ verfügbar} \\ 1 & \text{sonst} \end{cases}$
    $* \ \bar{b} \leq \bar{b}' :\leftrightarrow b_i \leq b_i', i = 1, \ldots, t (0 < 1)$
    $* \ \bot = (0, \ldots, 0)$ "beste Information"
    $* \ \top = (1, \ldots, 1)$, $0 \sqcup 1 = 1$
    $* \ (b_1, \ldots, b_t) \sqcup (b_1', \ldots, b_t') = (b_1 \sqcup b_1', \ldots, b_t \sqcup b_t')$
    $* \ d_s = (1, \ldots, 1)$

* Konstruktion von $\varphi : \underline{\text{Kno}} \to \{f : \mathbb{B}^t \to \mathbb{B}^t \text{ monoton}\}$ zunächst für einzelne Anweisungen, dann für Blöcke: $\alpha \in \underline{\text{Anw}} \mapsto \varphi_\alpha : \mathbb{B}^t \to \mathbb{B}^t$
  - $\varphi_{x \leftarrow e} := \underline{\text{kill}}_x \circ \underline{\text{gen}}_e$

$$\underline{\text{gen}}_e(b_1, \ldots, b_t) := (b'_1, \ldots, b'_t) \quad b'_i := \begin{cases} 0 & \text{falls } e = e_i \\ b_i & \text{sonst} \end{cases}$$
$$\underline{\text{kill}}_x(b_1, \ldots, b_t) := (b'_1, \ldots, b'_t) \quad b'_i := \begin{cases} 1 & \text{falls } x \in V_{e_i} \\ b_i & \text{sonst} \end{cases}$$

  - $\varphi_{\underline{\text{goto } k}} := \text{id}_{\mathbb{B}^t}$
  - $\varphi_{\underline{\text{if } be \text{ goto } k}} := \underline{\text{gen}}_{be}$

  Beachte: φ_α ist monoton, weil alle gen- und kill-Funktionen monoton sind.
* Konstruktion der **Blocktransformation**
  - $\beta = \alpha_1; \ldots; \alpha_k$ ($k > 0$, weil Sprungbefehle bleiben)
  - $\varphi_\beta := \varphi_{\alpha_k} \circ \ldots \circ \varphi_{\alpha_1}$ monoton

  Für dieses DFA-System $\Delta = \langle G, \mathfrak{I} \rangle$ gilt: $AE_\beta^{MOP} = AE_\beta^{MFP} \quad \forall \beta \in \underline{\text{Kno}}$
  (AE: "available expression" statt AI "Analyseinformation")
  Grund: Alle φ_β sind distributiv:

$$\begin{aligned} \underline{\text{gen}}_{e_i}(b) \sqcup \underline{\text{gen}}_{e_i}(b') &= (b_1, \ldots, \tfrac{0}{i}, \ldots, b_t) \sqcup (b'_1, \ldots, \tfrac{0}{i}, \ldots, b'_t) \\ &= (b_1 \sqcup b'_1, \ldots, \tfrac{0}{i}, \ldots, b_t \sqcup b'_t) \\ &= \underline{\text{gen}}_i(b \sqcup b') \end{aligned}$$

Beispiel (Folie: "Beispiel: Common Subexpression Elimination"):

- $\underline{\text{OpExp}}_\pi = \{x + y, x * 2, t * t, z * 2, i + 1, z > t, i > 10\}$
- $D = \mathbb{B}^7$
- $\varphi_1(b_1, \ldots, b_7) = (b_1, \ldots, b_7)$
- $\varphi_2(b_1, \ldots, b_7) = (0, 0, 0, 1, 1, 1, 1)$

1	2	3	4	5	6	7
0	~~1~~	~~1~~	1	1	1	1
	0	0				

- $\varphi_3(b_1, \ldots, b_7) = (0, 1, b_3, 1, b_5, 0, b_7)$

1	2	3	4	5	6	7
0	1		~~0~~		~~1~~	
			1		0	

- $\varphi_4(b_1, \ldots, b_7) = (b_1, 0, b_3, b_4, b_5, b_6, b_7)$
- $\varphi_5(b_1, \ldots, b_7) = (b_1, b_2, 0, b_4, b_5, b_6, b_7)$
- $\varphi_6(b_1, \ldots, b_7) = (b_1, b_2, b_3, b_4, 1, b_6, 0)$
- (E_Δ) :

$$\begin{aligned} X_1 &= (1, \ldots, 1) \\ X_2 &= X_2 \\ X_3 &= (0, 0, 0, 1, 1, 1, 1) \sqcup \varphi_6(X_6) \\ X_4 &= \varphi_3(X_3) \\ X_5 &= \varphi_3(X_3) \\ X_6 &= \varphi_4(X_4) \sqcup \varphi_5(X_5) \end{aligned}$$

  Vereinfachung:

  - $X_3 = (0, 0, 0, 1, 1, 1, 1) \sqcup \varphi_6([\varphi_4 \sqcup \varphi_5]\varphi_3(X_3)) \quad (*)$

– $\varphi_6([\varphi_4 \sqcup \varphi_5]\varphi_3(b_1, \ldots, b_7)) = (0, 1, b_3, 1, 1, 0, 0)$

Die Fixpunktiteration für $(*)$ beginnt mit dem besten Wert $\bot = (0, \ldots, 0) \in \mathbb{B}^7$:

– $T^1_{(*)}(\bot) = (0,0,0,1,1,1,1) \sqcup (0,1,0,1,1,0,0) = (0,1,0,1,1,1,1)$

– $T^2_{(*)}(\bot) = (0,0,0,1,1,1,1) \sqcup (0,1,0,1,1,0,0) = (0,1,0,1,1,1,1)$

– $\underline{\text{fix}}(T_{(*)}) = (0,1,0,1,1,1,1)$

[2004/12/08]

[Folie: "Beispiel: Common Subexpression Elimination"]

[Folie: "Gleichungssystem Available Expressions Analyse"]

Ergebnis:

$$
\begin{array}{ll}
AE_1 = (1,1,1,1,1,1,1) & \\
AE_2 = (1,1,1,1,1,1,1) & \\
AE_3 = (0,1,0,1,1,1,1) & \\
AE_4 = (0,1,0,1,1,0,1) & \varphi_3(b_1, \ldots, b_7) = (0,1,b_3,1,b_5,0,b_7) \\
AE_5 = (0,1,0,1,1,0,1) & \\
AE_6 = (0,1,0,1,1,0,1) & \varphi_4(b_1, \ldots, b_7) = (b_1,0,b_2,b_3,b_4,b_5,b_6,b_7) \\
 & \varphi_5(b_1, \ldots, b_7) = (b_1,b_2,0,b_3,b_4,b_5,b_6,b_7)
\end{array}
$$

Folgerung:

– $x + y, t * t$ sind verfügbar in den Blöcken 3,4,5,6.

– $z > t$ ist verfügbar in den Blöcken 4,5,6.

Beachte:

– $x \cdot z$ ist bereits für $B3$ nicht verfügbar, weil $B3$ von $B2$ und $B6$ erreichbar (ohne $B5$ wäre $x \cdot z$ für $B3$ verfügbar)

- Lokale Analyse für Basisblöcke
  Unterschied zur AE-Analyse von SLC-Programmen: Neue Startinformation: Nicht mehr $(1, \ldots, 1)$, sondern die durch globale Analyse gewonnene Information AE_i für Block i ($i = 1, \ldots, 6$ im Bsp.).

  – CS-Elimination
    $T_{CS} : SLC \to SLC$ übertragen: Zwischenspeichern auf temporären Variablen

4.6 RD-Analyse und Konstantenfaltung

Ziel: Ersetze konstante Variablen und konstante Ausdrücke auf rechten Seiten von Wertzuweisungen und in Bedingungen durch ihre Werte (Propagation und Faltung).

Definition: Sei $\pi \in SLC$ mit Interpretation $\mathfrak{A} = \langle A; \varphi \rangle$, $v \in V_\pi, i \in L_\pi$ und $a \in A$.
Dann hat v bei i den konstanten Wert a (Bezeichnung:$\underline{\text{const}}(v, i) = a$), falls dies auf jedem **Berechnungspfad** zu i gilt.

Beachte: i kann auf verschiedenen Berechnungspfaden (verschiedene Eingaben) und mehrfach auf einem Berechnungspfad erreicht werden.

Satz: Die Konstanzeigenschaft von Variablen eines iterativen Programmes $\pi \in IC$ ist i.a. nicht entscheidbar (i.a. bedeutet: bei entsprechender Wahl der Interpretation).

Beweis: Angenommen, die Konstanz-Eigenschaft wäre für alle Interpretationen entscheidbar.

- Interpretation: $\mathfrak{A} = \langle \mathbb{N}, +1, 0, 0? \rangle$
- $IC(\mathfrak{A})$ ist universell (jede berechenbare Funktion darstellbar)
- Sei $\pi \in IC(\mathfrak{A})$, $IV_\pi = \{x_1, \ldots, x_n\}$, $OV_\pi = \{y_1, \ldots, y_n\}$.
- Neues Programm: $\pi_{const} \in IC(\mathfrak{A})$:
  Dann gilt: $\underline{\text{const}}(y_1, i) = 0 \leftrightarrow \pi$ divergiert

[2004/12/10]

- $\underline{\text{const}}(y_1, 0) \leftrightarrow \pi$ divergent, d.h. bei keiner Eingabe terminiert. *Bei der Wiederholung wurde diese Variante genannt.*

Entscheidbarkeit der Konstanzeigenschaft $\leftrightarrow$ Entscheidbarkeit der Divergenz von $\pi \leftrightarrow$ Entscheidbarkeit des Halteproblems von π (Modifikation von π durch Zuweisung der Eingabewerte an Eingabevariablen.)

[Folie: "Reaching Definitions Analyse und Konstantenfaltung"]

4.6.1 RD-Analyse

Methode Zweistufige Berechnung

1. Bestimme für jeden Basisblock von $\pi \in IC$ die Menge der am Blockanfang konstanten Variablen mit ihrem Wert (globale Analyse).
2. Bestimme aus dieser Information die entsprechenden Informationen für die Anweisungen der Basisblöcke.

Zu 1: Konstruktion eines DFA-System $\Delta = \langle G, \mathfrak{I} \rangle$. Konstruktion von $\mathfrak{I} = \langle \mathcal{D}, d_s, \varphi \rangle$.

- $V_\pi = \{v_1, \ldots, v_n\}$
- Beispiel:
  $\{ \underset{1}{u}, \underset{2}{v}, \underset{3}{x}, \underset{4}{y}, \underset{5}{z} \}$
- $D := (A \cup \{\bot, \top\})^k = \hat{A}^k$
- $d_j = a$ bedeutet: v_j hat den Wert d_j
- $d_j = \top$ bedeutet: v_j hat mehrere Werte
- $d_j = \bot$ bedeutet: ? (irgendein Wert, unbekannt)
- Beispiel: $D = (\mathbb{Z} \cup \{\bot, \top\})^5$, $(\bot, 4, 5, 6, 7) \sqcup (1, 4, 2, 1, 7) = (1, 4, \top, \top, 7)$
- $d_s = (\underbrace{\top, \top, \ldots, \top}_{\text{Eingabevariablen}} , \underbrace{\bot, \ldots, \bot}_{\text{übrige Variablen}})$ Startinformation
- Konstruktion der montonen **Blocktransformationen** $\varphi_i : \hat{A}^k \to \hat{A}^k$
- Zunächst **Anweisungstransformationen** $\varphi_\alpha : \hat{A}^k \to \hat{A}^k$
- Dazu: Erweiterte Ausdruckssemantik (vgl. SLC)
  - $e \in V \cup C \cup \{f(u_1, \ldots, u_r) | f \in \Sigma^{(r)}, u_i \in V \cup C\}$,
  - $d \in \hat{A}^k$
  - $[\![E]\!]_{\mathfrak{A}}(d) \in \hat{A}$, wobei
  - $f_{\mathfrak{A}}(\ldots, \bot, \ldots) = \bot$, z.B. $3 + \bot = \bot$
  - $f_{\mathfrak{A}}(\ldots, \top, \ldots) = \top$, z.B. $4 * \top = \top$

- $f_{\mathfrak{A}}(\ldots, \bot, \ldots, \top, \ldots) = \top$

- $\varphi_{x \leftarrow e}(d_1, \ldots, d_k) = (d'_1, \ldots, d'_k)$

$$d'_j := \begin{cases} [\![e]\!]_{\mathfrak{A}}(d_1, \ldots, d_k) & \text{falls } x = v_j \\ d_j & \text{sonst} \end{cases}$$

- Sprunganweisungen ändern die Konstanteninformation nicht! $\varphi_\alpha(d) = d$, falls α Sprunganweisung
- Blocktransformation: Komposition der Anweisungstransformationen
- Beispiel (Folie: "Reaching Definitions Analyse und Konstantenfaltung"):

$$\begin{array}{ccccccc} \{ & u, & v, & x, & y, & z & \} \\ & 1 & 2 & 3 & 4 & 5 & \end{array}$$

$$\begin{array}{ll} \varphi_1(d_1, \ldots, d_5) & = (d_1, \ldots, d_5) \\ \varphi_2(d_1, \ldots, d_5) & = (d_1, d_2, 1, 1, 1) \\ \varphi_3(d_1, \ldots, d_5) & = (d_1, \ldots, d_5) \\ \varphi_4(d_1, \ldots, d_5) & = (d_3 + d_4 + d_5, d_2, \ldots, d_5) \\ \varphi_5(d_1, \ldots, d_5) & = (d_1, d_2, d_4 + 2, d_4, d_5) \\ \varphi_6(d_1, \ldots, d_5) & = (d_1, d_3 + d_4, d_3, d_4, d_5) \end{array}$$

  φ_i sind monoton, weil alle erweiterten Grundfunktionen $\hat{f}_{\mathfrak{A}}$ monoton sind.

- Bestimmung der **MFP-Lösung** von $\Delta = \langle G, \mathfrak{I} \rangle$:
  - Datenflußgleichungen: Beispiel

$$(E_\Delta) \quad \begin{array}{ll} X_1 & = d_1 \\ X_2 & = \varphi_1(X_1) \\ X_3 & = \varphi_2(X_2) \sqcup \varphi_6(X_6) \\ X_4 & = \varphi_3(X_3) \\ X_5 & = \varphi_4(X_4) \\ X_6 & = \varphi_4(X_4) \sqcup \varphi_5(X_5) \end{array}$$

  - Vereinfachung von (E_Δ)

$$\begin{array}{ll} X_1 & = (\top, \top, \top, \top, \top) \\ X_2 & = (\top, \top, \top, \top, \top) \\ X_3 & = (\top, \top, 1, 1, 1) \sqcup \varphi_6([\varphi_4 \sqcup \varphi_5 \circ \varphi_4](X_3))(*) \\ X_4 & = X_3 \\ X_5 & = \varphi_4(X_3) \\ X_6 & = [\varphi_4 \sqcup \varphi_5 \circ \varphi_4](X_3) \end{array}$$

$$\begin{array}{ll} [\varphi_5 \circ \varphi_4](d_1, \ldots, d_5) & = (d_3 + d_4 + d_5, d_2, d_4 + 2, d_4, d_5) \\ \varphi_4 \sqcup [\varphi_5 \circ \varphi_4](d_1, \ldots, d_5) & = (d_3 + d_5 + d_5, d_2, d_3 \sqcup (d_4 + 2), d_4, d_5) \\ \varphi_6(\varphi_4 \sqcup [\varphi_5 \circ \varphi_4])(d_1, \ldots, d_5) & = (d_3 + d_4 + d_5, (d_3 \sqcup (d_4 + 2)) + d_4, d_3 \sqcup (d_4 + 2), d_4, d_5) \end{array}$$

[2004/12/15]

Wiederholung

- $V_\pi = \{u, v, x, y, z\}$
- $D = \hat{\mathbb{Z}}$
- Folie: "Reaching Definitions Analyse und Konstantenfaltung"
- Folie: "RD^{MFP}: Transferfunktionen und Gleichungssystem"

- Fixpunkt-Iteration für Gleichung $(*)$ (siehe Folie):

$$T^0_{(*)}(\bot,\ldots,\bot) = (\bot,\ldots,\bot)$$
$$T^1_{(*)}(\bot,\ldots,\bot) = (\top,\top,1,1,1)$$
$$T^2_{(*)}(\bot,\ldots,\bot) = (\top,\top,\top,1,1)$$
$$T^3_{(*)}(\bot,\ldots,\bot) = (\top,\top,\top,1,1)$$

  Also: $\underline{\text{fix}}(T_{(*)}) = (\top,\top,\top,1,1)$

- MFP-Lösung von Δ:

$$RD^{MFP}_1 = (\top,\top,\ldots,\top)$$
$$RD^{MFP}_2 = (\top,\top,\ldots,\top)$$
$$RD^{MFP}_3 = (\top,\top,\top,1,1)$$
$$RD^{MFP}_4 = (\top,\top,\top,1,1)$$
$$RD^{MFP}_5 = (\top,\top,\top,1,1)$$
$$RD^{MFP}_6 = (\top,\top,\top,1,1)$$

  Die Variablen y und z haben bei Eintritt in die Blöcke 3, 4, 5 und 6 den konstanten Wert 1.

Lokale Analyse SLC-Analyse von Block i mit der Startinformation $RD^{MFP}_i (i = 1,\ldots,k)$, hier: 6.

Optimierung Konstantenfaltung wie bei SLC.

Frage: $RD^{MOP} = RD^{MFP}$? Knotentransformationen φ_i distributiv ?

Satz: Es gibt ein $\pi \in IC$ mit $RD^{MOP} \neq RD^{MFP}$.

Beweis:

[**Folie: "4.9** $RD^{MOP} \neq RD^{MFP}$**"**]

- $\begin{Bmatrix} x, & y, & z \\ 1 & 2 & 3 \end{Bmatrix}$, $D = \hat{\mathbb{Z}}^3$

- $\varphi_1(d_1,d_2,d_3) = (d_1,d_2,0)$
  $\varphi_2(d_1,d_2,d_3) = (2,3,d_3)$
  $\varphi_3(d_1,d_2,d_3) = (3,2,d_3)$
  $\varphi_4(d_1,d_2,d_3) = (d_1,d_2,d_1+d_2)$

- $RD^{MOP}_5 = (2,3,5) \sqcup (3,2,5) = (\top,\top,5)$

- RD^{MFP} :

$$\begin{array}{lll} X_1 = (\top,\top,\top) & & \\ X_2 = \varphi_1(X_1) & = (\top,\top,0) & \\ X_3 = \varphi_1(X_1) & = (\top,\top,0) & \\ X_4 = \varphi_2(X_2) \sqcup \varphi_3(X_3) & = (2,3,0) \sqcup (3,2,0) & = (\top,\top,0) \\ X_5 = \varphi_4(X_4) & = (\top,\top,\top) & \end{array}$$

  Also: $RD^{MFP}_5 = (\top,\top,\top) > RD^{MOP}_5$

q.e.d.

Grund: φ_4 nicht distributiv:

$\varphi_4(d \sqcup d') \neq \varphi_4(d) \sqcup \varphi_4(d')$ für $d = (2,3,z), \quad d' = (3,2,z)$.

Insgesamt gilt: Falls Block 2 bei $z = 0$ Sprungziel:

$RD_5 = (2,3,5) < RD^{MOP}_5 < RD^{MFP}_5$

4.6.2 Nicht-Entscheidbarkeit der Konstanteninformationen RD

[Folie: "Nicht-Entscheidbarkeit der Konstanteninformationen RD"]

Satz: Die pfaddefinierten Konstanteninformationen $RD_i^{MOP}(i = 1, \ldots, k)$ sind i.a. nicht entscheidbar.

Beweis: Wären die Mengen RD_i^{MOP} immer entscheidbar, so auch das MPCP ("Modifiziertes Postsches Korrespondenzproblem") und damit das PCP.
MPCP: Sei Σ ein Alphabet, $n \in \mathbb{N}$ und $u_1, v_1, \ldots, u_n, v_n \in \Sigma^*$. Gibt es $i_1, \ldots, i_m \in \{1, \ldots, n\} mm \geq 1$ mit $i_1 = 1$ ("modifiziertes PCP"), sodaß

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

Es gilt (z.B. Schöning, TI): Das MPCP ist nicht entscheidbar.
Betrachte die o.g. Folie mit der Interpretation $\langle A; \varphi \rangle$,

- $A = \Sigma^*$,
- $\varphi(++)(u, v) := uv$,
- $\varphi(=)(u, v) := \begin{cases} \varepsilon & \text{falls } u \neq v \\ a & \text{falls } u = v (a \in \Sigma) \end{cases}$

Dann folgt: Jeder Pfad vom Startknoten nach k entspricht einer Indexfolge $i_1, \ldots, i_m$ mit $i_1 = 1$ und umgekehrt. Also: $RD_k^{MOP}(z) = \varepsilon \curvearrowright MPCP(u_1, v_1, \ldots, u_n, v_n)$ hat keine Lösung. Somit würde die Entscheidbarkeit der RD_i^{MOP} die Entscheidbarkeit von $MPCP(u_1, v_1, \ldots, u_n, v_n)$ implizieren. Widerspruch!

q.e.d.

4.6.3 Zusammenfassung

1. $RD \leq RD^{MOP} \leq RD^{MFP}$
2. $RD < RD^{MOP} < RD^{MFP}$
3. RD und RD^{MOP} sind i.a. nicht entscheidbar.

4.7 Effiziente Fixpunkt-Berechnung

Δ DFA-System, $T_\Delta : D^n \to D^n$. MFP-Lösung von Δ:

$$AI^{MFP} = \underline{\text{fix}}(T_\Delta) \in D^n$$

T_Δ stetig $\curvearrowright \underline{\text{fix}}(T_\Delta) = \bigsqcup_{i=0}^{\infty} T_\Delta^i(\bot)$. Falls $\mathcal{D} = \langle D; \leq \rangle$ mit ACC $\curvearrowright \underline{\text{fix}}(T_\Delta) = \bigsqcup_{i=0}^{m} T_\Delta^i(\bot) = T_\Delta^m(\bot)$ für passendes $m \in \mathbb{N}$.

[2004/12/17]

Statt gleichmäßiger Fixpunktiteration werden wir komponentenweise Iteration betreiben.

Definition (Faire Folge): Eine Folge $I = i_1, i_2, i_3, \ldots$ mit $i_j \in \{1, \ldots, n\}$ heißt **fair**, wenn für jedes $j \in \mathbb{N}$ und jedes $i \in \{1, \ldots, n\}$ ein $m \in \mathbb{N}$ existiert mit $i = i_{j+m}$.
Mit anderen Worten: In einer fairen Folge kommt jeder Index unendlich oft vor.

Beispiel: $(1, \ldots, n, 1, \ldots, n, \ldots)$

Ein Index i bestimmt für $T_\Delta : D^n \to D^n$ die Komponentenfunktion $g_i : D^n \to D^n$ mit $g_i(d_1, \ldots, d_n) = (d'_1, \ldots, d'_n)$, wobei $d'_i := \underline{\text{proj}}_i(T_\Delta(d_1, \ldots, d_n))$ und $d'_j := d_j$ für $j \neq i$. (Berechnung durch i-te Gleichung.)

Satz (Chaotische Fixpunkt-Iteration): Sei $\mathcal{D} = \langle D, \leq \rangle$ ein vollständiger Verband und $T : D^n \to D^n$ stetig. Dann gilt für eine faire Indexfolge $I = (i_1, i_2, \ldots)$ mit $i_j \in \{1, \ldots, n\}$

$$\underline{\text{fix}}(T) = \sqcup\{g_{i_j}(g_{i_{j-1}} \ldots (g_{i_1}(\bot)) \ldots) | j \in \mathbb{N}\}$$

Frage: Ist diese Folge monoton wachsend ?
Aus dem Beweis des Fixpunktsatzes ist bekannt, daß $\bot \leq T(\bot) \leq T^2(\bot) \leq \ldots$; hier kann dies jedoch nicht angewendet werden, da die Relationen nicht bekannt sind: $\bot \leq g_{i_1}(\bot) ? g_{i_2}(g_{i_1}(\bot))$.

Beweisskizze: $g_{i_j}(\ldots g_{i_1}(\bot)\ldots) \leq T^j(\bot) \curvearrowright \sqcup\{g_{i_j}(\ldots g_{i_1}(\bot)\ldots) | j \in \mathbb{N}\} \leq \sqcup\{T^j(\bot) | j \in \mathbb{N}\} = \underline{\text{fix}}(T)$ Wegen der Fairness gilt: Zu jedem $j \in \mathbb{N}$ existiert ein k_j, sodaß in $(1, \ldots, i_{k_j})$ jeder Index mindestens j-mal vorkommt. Damit folgt $T^j(\bot) \leq g_{i_{k_j}}(\ldots(g_{i_1}(\bot)\ldots)$. Somit gilt $\underline{\text{fix}}(T) = \sqcup\{T^j(\bot) | j \in \mathbb{N}\} \leq \sqcup\, g_{i_{k_j}}(\ldots g_{i_1}(\bot)\ldots)$

q.e.d.

Folgerung: Besitzt $\mathcal{D}$ die ACC-Eigenschaft, dann existiert eine Indexfolge $i_1, \ldots, i_k$, sodaß $\underline{\text{fix}}(T) = g_{i_k}(\ldots g_{i_1}(\bot)\ldots)$.

Problem: Bestimmung einer möglichst kurzen Indexfolge.

Worklist-Algorithmus: Komponentenfunktionen $g_i : D^n \to D^n$ weiter zerlegen: $g_i(d_1, \ldots, d_n) = (d'_1, \ldots, d'_n)$ mit $d'_k = d_j$ für $k \neq i$ in

$$\begin{aligned} d'_i &= \underline{\text{proj}}_i(T_\Delta(d_1, \ldots, d_k)) \\ &= \sqcup\{\varphi_j(d_j) | j \in \underline{\text{Vor}}(i)\} \end{aligned}$$

Rechenschritte: $\varphi_j(d_j)$ durch Flußgraphen steuern.

- Eingabe: $\Delta = \langle G, \mathfrak{I} \rangle$ mit $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ und $\mathfrak{I} = \langle \mathcal{D}, \varphi, d_s \rangle$, $\underline{\text{Kno}} = \{s = 1, 2, \ldots, n\}$
- Verfahren:

  $\underline{\text{var}}\ j, l$: $\underline{\text{Kno}}$;
  $\underline{\text{var}}\ AI$: array $[1..n]$ of D;
  $\underline{\text{var}}\ W$: list of $\underline{\text{Kan}}$ % Worklist für Rechenschritt;

  Start:

  $AI[1] := d_s$
  $AI[j] := \bot_D$ für $j = 2, \ldots, n$;
  $W := nil$;
  for all $(j, l) \in \underline{\text{Kan}}$ do
  $W := \text{cons}((j, l), W)$;

  Iteration: % φ_j als Rechenschritt

  $\underline{\text{while}}\ W \neq nil\ \underline{\text{do}}$
  $k := \text{fst}(\text{head}(W)); k' := \text{snd}(\text{head}(w))$;
  $W := \text{tail}(W)$;
  $\underline{\text{if}}\ \varphi_k(AI[k]) \not\leq AI[k']$
  $\underline{\text{then}}\ AI[k'] := AI[k'] \sqcup \varphi_k(AI[k])$;
  for all k'' with $(k', k'' \in \underline{\text{Kan}})$ but not (k', k'') in W
  $W := \text{cons}((k', k''), W)$;

  [1] Ausgabe:

  $AI = (AI[1], \ldots, AI[n])$

[Folie: "Beispiel: Common Subexpression Elimination"]

[1] fst = first; snd = second

Beispiel (AE-Analyse):

$\underline{\text{OpExp}}^{\pi} = \{x+y, x*z, t*t, z*2, i+1\}$
$D = \mathbb{B}^5$
$W = (1,2)(2,3)(3,4)(3,5)(4,6)(5,6)(6,3)$

W	$RD[1]$	$RD[2]$	$RD[3]$	4	5	6
$(1,2)\ldots$	$(1,\ldots,1)$	$(0,\ldots,0)$	$(0,\ldots,0)$	$0,\ldots,0$	$0,\ldots,0$	$0,\ldots,0$
$(2,3)\ldots$		$(1,\ldots,1)$	$(0,0,0,1,1)$			
$(3,4)\ldots$				$0,1,0,1,1$		
$(3,5)\ldots$					$0,1,0,1,1$	
$(4,6)\ldots$						$0,0,0,1,1$
$(5,6)\ldots$						$0,1,0,1,1$
$(6,3)$			$(0,1,0,1,1)$			
$(3,4)$				$(0,1,0,1,1)$		
$(3,5)$					$(0,1,0,1,1)$	

[2004/12/22]

4.8 Live Variable Analyse und Dead Code Elimination für IC-Programme

Definition: Sei $\pi \in IC, x \in V_\pi, i \in L_\pi$. Dann heißt x **lebendig in** i, falls $p \in \underline{\text{Pfad}}[i, q+1]$ existiert, sodaß x auf p benutzt wird (rechte Seite einer Zuweisung oder Test oder Ausgabevariable), ohne vorher neu definiert zu werden.

Ausgabe: Bestimme für jeden Programmpunkt $i \in L_\pi$ die Menge LV_i der lebendigen Variablen.

Hier: Verbesserung (s. SLC): Bestimmung der NV_i (needed variables)

Beachte: Wertzuweiseungen mit toter linker Seite bilden Dead Code.

[Folie: "Beispiel: Dead Code Elimination"]

Konstruktion eines DFA-Systems $\Delta = \langle G, \mathfrak{I} \rangle$:

- Rückwärtsanalyse: Welche Anweisungen sind ausgaberelevant ?
- Datenflußgraph G ("Rückwärts-BB-Graph") mit $\mathcal{D} = \langle \mathcal{P}(V_\pi); \subseteq \rangle$
  - Blocktransformationen $\varphi_i : \mathcal{P}(V_\pi) \to \mathcal{P}(V_\pi)$
  - in Block $i : \alpha_1; \ldots, \alpha_R$: $\varphi_i := \varphi_{\alpha_1} \circ \ldots \circ \varphi_{\alpha_R}$ wegen Rückwärtsanalyse.
- Anweisungstransformationen: $t_\alpha : \mathcal{P}(V_\pi) \to \mathcal{P}(V_\pi)$

$$\begin{array}{ll} \alpha = v \leftarrow e & t_{v \leftarrow e}(M) := \underline{\text{if}}\ v \in M\ \underline{\text{then}}\ M \setminus \{v\} \cup V_e\ \underline{\text{else}}\ M \\ \alpha = \underline{\text{if}}\ be\ \underline{\text{goto}}\ l & t_\alpha(M) := M \cup V_{be} \\ \alpha = \underline{\text{goto}}\ l & t_\alpha(M) = m \end{array}$$

- Zusätzlich für Ausgabeknoten: $\varphi_{out}(M) := OV_\pi$
- Startinformation: $d_s = \varnothing$
- Gleichungssystem:

$$E_\Delta : \begin{array}{l} X_{out} = d_s \\ X_2 = \varphi_{out}(X_{out}) \sqcup \varphi_3(X_3) \\ X_3 = \varphi_2(X_2) \\ X_1 = \varphi_2(X_2) \\ X_{in} = \varphi_1(X_1) \end{array}$$

- *Der Operator "−" bedeutet, daß der "Subtrahent" aus der Menge entfernt wird, analog bedeutet der Operator "+", daß der "Summand" zu der Menge hinzugefügt wird.*

$\varphi_1(M)$	$= M$	$-w$	
		$-u+y$	falls $u \in M$
		$+x$	falls $v \in M$
		$-v+(x,y)$	falls $v \in M$
	$= M$	$-w$	
		$-u+y$	falls $u \in M$
		$-v+(x,y)$	falls $v \in M$
$\varphi_2(M)$	$= M$	$-z+(u,v)$	
$\varphi_3(M)$	$= M$	$-z+(u,v)$	falls $z \in M$
		$-v+w$	falls $v \in M$
$\varphi_{out}(M)$	$= \{z\}$		

- Berechnung von NV^{MFP} nach Worklist-Algorithmus

W	NV_{out}	NV_1	NV_2	NV_3	NV_{in}
$(out,2)(2,3)(3,2)(2,1)(1,in)$	$d_s = \varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$(2,3)(3,2)(2,1)(1,in)$	$d_s = \varnothing$	$\varnothing$	z	u,v	$\varnothing$
$(3,2)(2,1)(1,in)$	$d_s = \varnothing$	$\varnothing$	u,w,z	u,v	$\varnothing$
$(2,3)(2,1)(1,in)$	$d_s = \varnothing$	$\varnothing$	u,w,z	u,v,w	$\varnothing$
$(3,2)(2,1)(1,in)$	$d_s = \varnothing$	$\varnothing$	u,w,z	u,v,w	$\varnothing$
$(2,1)(1,in)$	$d_s = \varnothing$	u,v,w	u,w,z	u,v,w	$\varnothing$
$(1,in)$	$d_s = \varnothing$	u,v,w	u,w,z	u,v,w	x,y

[Folie: "Beispiel: Dead Code Elimination"]

- Ergebnis der globalen NV-Analyse:

$$NV_1^{MFP} = \{u,v,w\}$$
$$NV_2^{MFP} = \{u,w,z\}$$
$$NV_3^{MFP} = \{u,v,w\}$$
$$NV_{in}^{MFP} = \{x,y\}$$

φ_i sind distributiv: $\varphi_i(M \cup M') = \varphi_i(M) \cup \varphi_i(M') \curvearrowright NV^{MOP} = NV^{MFP} \curvearrowright$ Dead Code Optimierung (vgl. SLC)

5 Kontrollflußanalyse und Schleifenoptimierung

[2005/01/12]

Sei $\pi \in IC$. Ausnutzung der Schleifenstruktur des Flußgraphen G_π (SI/vorwärts)

- schnellere Fixpunktberechnung (Schleifen)
- Schleifenoptimierung (großer Einfluß)
  1. Code Motion (Verschieben von schleifeninvarianten Anweisungen vor die Schleife)
  2. Elimination von Induktionsvariablen (gleiche Wertprogression von mehreren Variablen nur auf einer Variablen durchführen)

5.1 Schleifenanalyse von Flußgraphen

Flußdiagramm eines strukturierten Quellprogrammes: Baum mit Rücksprüngen:

- while-Schleifen: 1 Eingang = Ausgang
- repeat-exit i-Schleifen: 1 Eingang - mehrere Ausgänge (Java: break, continue-Anweisungen)

[Folie: "Beispiele für Flußgraphen"]

Flußgraph G_1:

- Schleifen $\{1,2,3\}, \{5,7\}$ (aber nicht $\{1,5,6,8\}$), $\{1,5,6,7,8\}$ und $\{1,2,3,5,6,7,8\}$
- Beachte: Jede Schleife hat genau einen Anfangsknoten, welcher vom Startknoten außerhalb der Schleife erreichbar ist, hier 1 bzw. 5.

Flußgraph G_2:

- Schleifen: Keine, weil $\{2,3\}$ 2 Anfangsknoten besitzt.

Definition: Sei $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ ein Flußgraph ($\underline{\text{Vor}}(s) = \varnothing$ und $\underline{\text{Pfad}}[s,k] \neq \varnothing$ für alle $k \in \underline{\text{Kno}}$). Dann heißt $S \subseteq \underline{\text{Kno}}$ eine **Schleife**, falls gilt:

1. S ist stark zusammenhängend, d.h. für alle $k, k' \in S$ mit $k \neq k'$ existiert $p \in \underline{\text{Pfad}}[k,k']$ mit $\underline{\text{Kno}}(p) \subseteq S$.
2. Es existiert genau ein Knoten $k \in S$, sodaß $\underline{\text{Vor}}(k) \setminus S \neq \varnothing$. k heißt **Anfang** von S.

Beachte: Es muß $p \in \underline{\text{Pfad}}[s,k)$ existieren mit $\underline{\text{Kno}}(p) \cap S = \varnothing$.

Aufgabe: Bestimmung der Schleifen von Flußgraphen.

- Dominatorrelation auf Knoten

Definition (Dominatorrelation): Sei $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ ein Flußgraph und $k, k' \in \underline{\text{Kno}}$.

1. k' dominiert k (Bezeichnung: k' dom k): $\leftrightarrow$ für jedes $p \in \underline{\text{Pfad}}[s,k]$ gilt: $k' \in \underline{\text{Kno}}(p)$
2. k' dominiert k direkt: $\curvearrowright k'$ dom $k, k' \neq k$, für alle k'' gilt: k'' dom $k, k'' \neq k \curvearrowright k''$ dom k'

Beispiel: G_3:

- s dom k, k dom k für all $k \in \underline{\text{Kno}}$
- 2 dom $k \curvearrowright k = 2$
- Dominatoren von 8: $(s), 1, 3, 4, 7, (8)$
- 4 dom 7 direkt

Folgerung: Jeder Knoten $k, k \neq s$, besitzt genau einen direkten Dominator, weil die Dominatoren von k linear geordnet sind. Daher ist eine Baumdarstellung der Dominatorrelation möglich.

Beispiel: Dominatorbaum von G_3: dom -Relation ist reflexiv, transitiv und antisymmetrisch, also eine Halbordnungsrelation.

Definition: $(b, a) \in \underline{\text{Kan}}$ heißt Rückwärtskante, falls a dom b.

Lemma: Sei (b, a) eine Rückwärtskante und $S := \{k \in \underline{\text{Kno}} | \exists p \in \underline{\text{Pfad}}[k, b] \text{ mit } a \notin \underline{\text{Kno}}(p)\}$ Dann ist $S \cup \{a\}$ eine Schleife mit Anfang a.

Beweis: Sei $k \in S$ und $p \in \underline{\text{Pfad}}[k, b]$ mit $a \notin \underline{\text{Kno}}(p)$. Sei $p' \in \underline{\text{Pfad}}[s, k)$. Also $p'p \in \underline{\text{Pfad}}[s, b]$. a dom $b \curvearrowright a \in \underline{\text{Kno}}(p'p) \curvearrowright a \in \underline{\text{Kno}}(p') \curvearrowright \exists p'' \in \underline{\text{Pfad}}(a, k]$ mit $a \notin \underline{\text{Kno}}(p'')$, sodaß $\underline{\text{Kno}}(p'') \subseteq S$. Es folgt:

1. $S \cup \{a\}$ ist stark zusammenhängend: $k, k' \in S \cup \{a\}$: $k \dashrightarrow a \dashrightarrow k'$
2. a ist Anfang von $S \cup \{a\}$: Da $\underline{\text{Vor}}(s) = \varnothing$, muß $a \neq s$ sein. Sei $p \in Pfad[s, a)$ mit $a \notin \underline{\text{Kno}}(p)$. Da a dom k für alle $k \in S$ gilt, folgt: $\underline{\text{Kno}}(p) \cap S = \varnothing$ und auch die Eindeutigkeit von a, weil S unter Vorgängerbildung abgeschlossen ist.

[2004/01/14]

[Folie: "Beispiele für Flußgraphen"]

Bemerkung: $S \cup \{a\}$ heißt **natürliche Schleife** von (b, a).

Beispiel: G_3: Die natürliche Schleife von $(4, 3)$ ist $\{3, 4, 5, 6, 7, 8, 10\}$.

Aufabge: Bestimmung der Rückwärtskanten und ihrer natürlichen Schleifen.

Idee: dom -Relation $\curvearrowright$ Rückwärtskanten; Vorgängerrelation $\curvearrowright$ natürliche Schleife
Flußgraphen von strukturierten Programmen besitzen nur natürliche Schleifen (ungesicherte Aussage, *die Rücksprungknoten können vermutlich nicht zusammenfallen*).

Definition (Reduzierbarer Flußgraph): Ein Flußgraph $G = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s \rangle$ heißt **reduzierbar**, falls $G' := \langle \underline{\text{Kno}}, \underline{\text{Kan}}', s \rangle$ mit $\underline{\text{Kan}}' := \underline{\text{Kan}} \setminus \{(a, b) | (a, b) \text{ ist Rückwärtskante}\}$ azyklisch ist.

Beispiel: G_1 und G_3 sind reduzierbar, G_2 nicht.

5.2 ud- und du-chains

Weitere Programminformation zur Bestimmung schleifeninvarianter Berechnungen.

- use-definition-chain (ud-chain): Menge der Definitionen für die Benutzung einer Variablen
- definition-use-chain (du-chain): Menge der Benutzungen für die Definition einer Variablen

Definition: Sei $\pi \in IC, v \in V_\pi, l, l_1, l_2 \in L_\pi$ und $p \in \underline{\text{Pfad}}_\pi(l_1, l_2)$.

- v wird in l definiert ($\underline{\text{def}}(v,l)$): $\leftrightarrow \alpha_l = l : v \leftarrow e$
- v wird in l benutzt ($\underline{\text{use}}(v,l)$): $\leftrightarrow$ entweder $\alpha_l : v \leftarrow e$ mit $v \in V_e$ oder $\alpha_l : \underline{\text{if}}\ be \ldots$ mit $v \in V_{be}$
- v wird auf p definiert: $\leftrightarrow \underline{\text{def}}(v,l)$ für ein $l \in \underline{\text{Kno}}(p)$
- $\underline{\text{clear}}(v,p) :\leftrightarrow v$ wird nicht auf p definiert
- $\underline{\text{ud}}(v,l) := \{l' | \underline{\text{use}}(v,l), \underline{\text{def}}(v,l'), \exists p \in \underline{\text{Pfad}}(l',l) : \underline{\text{clear}}(v,p)\}$
- $\underline{\text{du}}(v,l) := \{l' | l \in \underline{\text{ud}}(v,l')\}$

Berechnung dieser Information durch Variante der Reaching-Definition-Analyse: DFA-System $\Delta = \langle G, \mathfrak{I}\rangle$ zu $\pi \in IC$, G SI/vorwärts-Graph von π, $\mathfrak{I} = \langle \mathcal{D}, \varphi, d_s\rangle$. Analyseinformation für $l \in \underline{\text{Kno}} = L_\pi$:

$$RD_l = \{(v,l') | \underline{\text{def}}(v,l') \wedge \exists p \in \underline{\text{Pfad}}(l',l) \text{ mit } \underline{\text{clear}}(v,p)\}$$

Es folgt:

$$l' \in \underline{\text{ud}}(v,l) \leftrightarrow (v,l') \in RD_l \text{ und } \underline{\text{use}}(v,l)$$

Abstrakte Interpretation $\mathfrak{I}$:

- $D := \mathcal{P})V_\pi \times (L_\pi \cup \{0\}))$, 0 Definitionsmarke für $v \in IV$
- $d_s := \{(v,0) | v \in IV\}$
- $\varphi_\alpha := D \to D$
  - $\alpha = i : v \leftarrow e$
    $\varphi_\alpha(M) := M \setminus \{(v,l) | l \in L_\pi \cup \{0\}\} \cup \{(v,i)\}$
  - andere α:
    $\varphi_\alpha = id_D$

φ_α distributiv, sodaß $RD_l^{MFP} = RD_l^{MOP}$.

[Folie: 5.2 "Beispiel: RD-Analyse und ud-chains"]

5.3 Schleifeninvariante Berechnungen und Code Motion

$\pi \in IC$ mit SI / Vorwärts-Graph $G_\pi = \langle \underline{\text{Kno}}, \underline{\text{Kan}}, s\rangle$, Schleife $S \subseteq \underline{\text{Kno}}$ mit Anfang $k \in S$, $\underline{\text{ud}}$-chains für $(v,l) \in V_\pi \times L_\pi$.

5.3.1 Schleifeninvariante Berechnungen

Idee: Rechenausdrücke mit festem Wert in S vor Schleifeneintritt berechnen

Beachte: Mögliche Propagation fester Werte innerhalb von S

Definition: $l \in S$ heißt **S-invariant**, wenn in $\alpha_l = l : x \leftarrow e$ oder $\alpha_l = l : \underline{\text{if}}\ be \ldots$ der Ausdruck e bzw. be während einer Schleifenberechnung stets denselben Wert hat.

Bestimmung S-invarianter Knoten: Es gilt: $l \in S$ ist S-invariant, falls $\alpha_l = l : x \leftarrow e$ oder $\alpha_l = l : \underline{\text{if}}\ be\ \underline{\text{goto}}\ l'$ und für alle $y \in V_e$ bzw. $y \in V_{be}$ gilt:

- $\underline{\text{ud}}(y,l) \cap S = \varnothing$
- $\underline{\text{ud}}(y,l) = \{l'\} \subseteq S$ und l' ist S-invariant

[2004/01/19]

Algorithmus (Analyse): Induktive Markierung S-invarianter Knoten

Optimierung: Berechnung der Ausdrücke S-invarianter Knoten vor Schleifeneintritt auf temporären Variablen (SLC)

5.3.2 Code Motion

Verschiebung einer Wertzuweisung vor den Schleifenanfang. Sei $l \in S$ S-invariant und $\alpha_l = l : x \leftarrow e$. Dann gilt: α_l kann direkt vor Schleifenanfang ausgeführt werden, falls

1. Für jeden Ausgangsknoten $k' \in S$, d.h. es existiert ein Knoten $k'' \notin S$ mit $(k', k'') \in \underline{\text{Kan}}$ gilt: l dom k'
2. Es gibt nur eine Definition von x in S.
3. Wird x in $l' \in S$ benutzt, so folgt: $\underline{\text{ud}}(x, l') = \{l\}$.

Die folgenden Folien erläutern in Beispielen diese Bedingungen.

[Folie: "Beispiel: Illegale Codeverschiebung (Bedingung 1)"]

[Folie: "Bedingungen 2 und 3 für Codeverschiebung"]

Analysealgorithmus:

1. S-invariante Knoten
2. O.g. Eigenschaten mit $\underline{\text{ud}}$- und dom -Information prüfen

Optimierung: $\surd$

5.4 Induktionsvariablen

Induktionsvariablen: Variablen einer Schleife mit konstanter Wertprogression

Ursache:

- $i \leftarrow i + 1$ direkte Induktion
- $j \leftarrow 3 * i$ indirekte Induktion

2 Optimierungen:

1. Strenth reduction (Addition statt Multiplikation), Beispiel: $j \leftarrow j + 3$
2. Elimination von Induktionsvariablen

Entstehung von Induktionsvariablen: z.B. Zählschleifen

[Folie: "Beispiel: Strength reduction für Induktionsvariablen"]

Definition (Induktionsvariablen): Sei $\pi \in IC(\mathbb{Z}, +, *, <, >)$ und S eine Schleife im (SI/Vorwärts)-Flußgraphen von π.

1. $x \in V_s$ heißt **Basisinduktionsvariable** ($x \in \text{BIV}(S)$), falls gilt:
   - Es existiert $i \in S$ mit $\alpha_i = i : x \leftarrow e$.
   - Für alle $i \in S$ mit $\alpha_i = i : x \leftarrow e$ gilt: $e \in \{x + c, c + x | c \in \mathbb{Z}\}$
2. $y \in V_s$ heißt **abhängig Induktionsvariable** ($y \in \text{AIV}(S)$), falls gilt:
   - Es existiert $i \in S$ mit $\alpha_i = i : y \leftarrow e$.
   - Für alle $i \in S$ mit $\alpha_i = i : y \leftarrow e$ gilt: $e \in \{c * x, x * c, c + x, x + c | c \in \mathbb{Z}, x \in \text{BIV}(S)\}$

Bemerkung: Verallgemeinerung durch verschachtelte Abhängigkeiten

Folgerung: Für eine Schleifenberechnung gilt: Konstante Wertprogression von $n \in \text{BIV}(S)$ überträgt sich mit einem Faktor auf die von x abhängigen Variablen $y \in \text{AIV}(S)$.

Beachte: Verschiedene Definitionen mit unterschiedlichen Progressionen

5.4.1 "Strength reduction" für Induktionsvariablen

Idee: Ersetze Multiplikationen durch Additionen (Schleife!)

Sei $y \in \mathrm{AIV}(S)$ und bei $i \in S$ eine Definition von y: $\alpha_i = i : y \leftarrow e$ mit $V_e = \{x\}, x \in \mathrm{BIV}(S)$. Ist e ein Produkt, so ist dies wie folgt in S vermeidbar: Wähle neue Variable t_i und füge hinter jeder Definition von x eine Defininition von t_i. Diese ist bestimmt durch e und die Definition von x: Falls $e = d * x$ und $x \leftarrow x + c$, so ist $t_i \leftarrow t_i + d * c$. Initialisierung von t_i vor Schleifenanfang: $t_i \leftarrow d * x$.

5.4.2 "Elimination von Induktionsvariablen"

Sei $y \in \mathrm{AIV}(S)$ mit genau einer Definition $i : y \leftarrow d * x$, also: $x \in \mathrm{BIV}(S)$. x werde nur benutzt in ihren Definitionen, in der Definition von y und in Bedingungen der Form if *be* goto .... Dann kann x eliminiert werden:

1. Transformation durch strenth reduction
2. Simulation von x in *be* durch t_i.

[Folie: "Beispiel: Elimination von Induktionsvariablen"]

6 Interprozedurale Analyse

- bisher: intra-prozedural (π als Prozedurrumpf)
- jetzt: Erweiterung von IC um rekursive Prozeduren
- Probleme: Abhängigkeit zwischen Aufruf und Rücksprung, Parameterbehandlung
- hier: keine geschachtelten Prozedurdeklarationen, nur Wert- und Ergebnisparameter

[Folie: "ICP-Beispielprogramm: Fibonaccifunktion"]

6.1 Syntax von ICP (Intermediate Code with Prozedures)

$\pi = (\text{Plist}, \text{Vlist}_0, \text{Alist}_0) \in ICP$

Plist	$= \text{Pdecl}_1, \ldots, \text{Pdecl}_r$	Liste von Prozedurdeklarationen
Pdecl_i	$= (\text{pid}_i, \text{Vlist}_i, \text{Alist}_i)$	$i = 1, \ldots, r$
pid_i		Prozedurbezeichner
Vlist_i	$=$ $\underline{\text{in}}\ x_1, \ldots, x_{n_i}$; $\underline{\text{out}}\ y_1, \ldots, y_{m_i}$; $\underline{\text{loc}}\ z_1, \ldots, z_{p_i}$;	Liste der formalen Parameter
Alist		Anweisungsliste
	$\underline{\text{Anw}}(IC) \cup \{\text{pid}_i(e_1, \ldots, e_{n_i}, v_1, \ldots, v_{m_i}) \mid$	
$\underline{\text{Anw}}$	$:=$ e_j einfache Rechenausdrücke als aktuelle Parameter,	
	$v_k \in V$ (paarweise verschieden), $i = 1, \ldots, r\}$	

Prozeduraufrufe in Alist_0 und $\text{Alist}_1, \ldots, \text{Alist}_r \rightsquigarrow$ verschränkte Rekursion.
Variablen müssen deklariert sein:

$$V_{\text{Alist}_i} \subseteq V_{\text{Vlist}_i} \quad (i = 0, \ldots, r)$$

6.2 Semantik von ICP

- Eingabeparameter als Wertparameter (call by value)
- Jeder Prozeduraufruf mit eigenem Zustandsraum
- Werte der formalen Ausgabeparameter in aktuelle Ausgabeparameter übertragen
- Laufzeitkeller mit Aufrufframes
- Fixpunktsemantik als Erweiterung der Fortsetzungssemantik um IC

6.3 MOP-Analyse für ICP-Programme

[Folie: "Flußgraph von π_{fib}"]

1. Problem: Pfade von $\pi \in ICP$

   a) Die reguläre Pfadmenge von G_{fib} (naiver Ansatz) $G_{fib} \mapsto G_{fib}^{\text{iter}}$ entsteht aus G_{fib} durch Hinzunahme von Aufrufkanten: $(1,3)(4,3)(5,3)$ und Rücksprungkanten: $(7,2)(7,5)(7,7)$, Ferner: Weglassen der Kanten hinter Prozeduraufrufen: $(1,2)(4,5)(5,7)$.
   $\underline{\text{Pfad}}[1,2] \ni 13672(\surd), 136753672(?)$
   - Vorteil: DFA-Technik ist direkt übertragbar

- Nachteil: Zu viele Pfade, Bedingungen für Analyse zu scharf
- Grund: Keine Bindung zwischen Rücksprung und Aufruf

b) Die kontextfreie Pfadmenge von G_{fib}: $G = \langle N, \Sigma, P, S\rangle$ mit

- $\Sigma = \{1, 2, \ldots, 7\}$,
- $N = \{P[1,2], P[2,2], P[3,7], P[4,7], \ldots, P[7,7]\}$,
- P :

$$\begin{aligned} P[1,2] &\to 1P[3,7]P[2,2] \\ P[2,2] &\to 2 \\ P[3,7] &\to 3P[4,7] \mid 3P[6,7] \\ P[4,7] &\to 4P[3,7]P[5,7] \\ P[5,7] &\to 5P[3,7]P[7,7] \\ P[6,7] &\to 6P[7,7] \\ P[7,7] &\to 7 \end{aligned}$$

- $P[1,2]$ 136753672 ist hier nicht möglich (nach 1367 muß eine 2 folgen) $\rightsquigarrow \underline{\text{RPfad}}[s,k)$

2. Problem: Knotentransformationen für Aufruf- und Rücksprungknoten, Abstrakte Interpretation von G_π:

- $\mathcal{D} = \langle D; \leq\rangle$ vollständiger Verband mit ACC
- $d_s \in D$ Startinformation (Hauptprogramm)
- Knotentransformationen:
  a) Bei Zuweisungs- und Verzweigungsknoten wie bekannt aus IC: $\varphi_{ij} : D \to D$
  b) Bei Stop- und Goto-Knoten $\varphi_{ij} = \mathrm{id}_D$
  c) Bei Aufruf und Rücksprung ?
     - Idee: Pfadberechnung mit Kellerspeicher (analog Laufzeitkeller)
     - Beachte: Rücksprungadressen unnötig, da Pfad vorgegeben
     - $\mathrm{Stack}_D := D^+$ (leerer Keller unnötig), Spitze rechts
     - $\varphi_{ij} : D \to D$ für Aufrufknoten ij zur Berechnung der Startinformation für den Aufruf aus oberstem Kellerelement (Parameterübergabe)
     - $\varphi_{k(q_k+1)} : D \times D \to D$ für den Rücksprungknoten zur Berechnung der Information beim Rücksprung, bestimmt durch Information vor Aufruf und bei Aufrufende (die beiden obersten Kellereinträge)

  Jedes φ_{ij} bestimmt eine Stacktransformation $\varphi^{st}_{ij} : \mathrm{Stack}_D \to \mathrm{Stack}_D$.

[2005/01/26]

[Folie: "ICP-Beispielprogramm: Fibonaccifunktion"]

[Folie: "Flußdiagramm von π_{fib}"]

Für Fall

- 2a: $\varphi^{st}_{ij}(wd) := w\varphi_{ij}(d)$
- 2b: $\varphi^{st}_{ij}(wd) := wd\varphi_{ij}(d)$ ("PUSH")
- 2c: $\varphi^{st}_{k(q_k+1)}(wdd') := w\varphi_{k(q_k+1)}(dd')$

Für $k \in \underline{\text{Kno}} \setminus \{s\}$, $(s = 01)$ und $p = (k_1, \ldots, k_q) \in \underline{\text{RPfad}}[s,k)$ definieren wir die Analyseinformation für k bzgl. p durch $AI\langle p\rangle := \mathrm{top}(\varphi_{k_q}(\ldots\varphi_{k_1}(d_s)\ldots)) \in D$ mit $\mathrm{top} : \mathrm{Stack}_D \to D \quad (wd \mapsto d)$.

Die MOP-Lösung für k:

- $AI^{MOP}_s := d_s$
- $AI^{MOP}_k := \sqcup\{AI\langle p\rangle | p \in \underline{\text{RPfad}}[s,k)\}$

6.4 MFP-Analyse für ICP-Programme

$\pi = (\text{Plist}, \text{Vlist}_0, \text{Alist}_0) \in ICP$ bestimmt die Flußgraphen $G_0, \ldots, G_r$ und den daraus kombinierten Flußgraphen G_π. Unterschiedliche Kantenstruktur: $\alpha_{ij} = \text{pid}_k(\ldots)$:

1. $(ij, i(j+1)) \in \underline{\text{Kan}}(G_i)$
2. $(ij, k1), (k(q_k+1), i(j+1)) \in \underline{\text{Kan}}(G_\pi)$

MFP-Lösung für iterative Flußgraphen:

$$E \left\{ \begin{array}{l} AI_s = d_s \\ AI_k = \bigsqcup\limits_{l \in \underline{\text{Vor}}(k)} \varphi_l(AI_l) \end{array} \right.$$

Übertragung auf den rekursiven Fall Betrachtung von $G_0, \ldots, G_r$ statt G_π, da die Gleichungen aus E lokale Zusammenhänge beschreiben.

- Aufrufknoten ij mit $\alpha_{ij} = \text{pid}_k(\ldots)$: Volle Transformationen $\Phi_k : D \to D$ von Q_k verwenden. Neben den Gleichungen für die Analyseinformationen zusätzliche Gleichungen für die $\Phi_k (k = 1, \ldots, r)$.
- Dazu: Duale Sicht zur Berechnung der Φ_k: $ij \mapsto$ nicht die von dort bis $i(q_i+1)$, sondern die von $i1$ bis ij bestimmte Transformation

  $$\begin{array}{l} \Phi_{i(q_i+1)} = \Phi_i \\ \Phi_{ij}(d) = \tilde{\varphi}_{il}(\Phi_{il}(d)) \sqcup \tilde{\varphi}_{im}(\Phi_{im}(d)) \end{array}$$

  $\tilde{\varphi}_{il}, \tilde{\varphi}_{im}$ lokale Anweisungstransformation:

  - $\tilde{\varphi}_{il} := \varphi_{il}$ für "iterative' Knoten (Anweisung, Verzweigung, Sprung)
  - il Aufrufknoten mit $\alpha_{ij} = \text{pid}_k(\ldots)$: $\tilde{\varphi}_{ij}(d) := \varphi_{k(q_k+1)}(\Phi_k(\varphi_{il}(d)))$

  $\rightsquigarrow$ rekursive Funktionsgleichungssysteme zur Bestimmung der Transformationen $\Phi_1, \ldots, \Phi_r$

  $$FE \left\{ \begin{array}{lll} F_i(X) & = F_{i(q_i+1)}(X) & \\ F_{i1}(X) & = X & \\ F_{ij}(X) & = \bigsqcup\limits_{il \in \underline{\text{Vor}}(ij) \text{ in } G_i} \tilde{\varphi}_{il}(F_{il}(X)) & \end{array} \right.$$

  wobei

  $$\tilde{\varphi}_{il}(X) = \left\{ \begin{array}{ll} \varphi_{il}(X) & \text{bei "iterativen" Knoten} \\ \varphi_{k(q_k+1)}(X, F_k(\varphi_{il}(X)) & \end{array} \right.$$

  Die eigentlichen Gleichungssysteme $E_0, \ldots, E_r$ zur Berechnung der Analyseinformationen AI_{ij}:

  $$\begin{array}{l} AI_{01} = d_s \\ AI_{i1} = \bigsqcup\limits_{kj \text{ mit } \alpha_{kj} = \text{pid}_i(\ldots)} \varphi_{kj}(AI_{kj}) \\ AI_{ij} = \bigsqcup\limits_{il \in \underline{\text{Vor}}(ij) \text{ in } G_i} \tilde{\varphi}_{il}(AI_{il}) \end{array}$$

$$E_\pi := \bigcup_{i=0}^{r} E_i \cup \bigcup_{i=1}^{r} FE_i$$

Fixpunkt-Technik bestimmt die MFP-Lösung $AI^{MFP} := (AI_{ij}^{MFP} | i = 0, \ldots, r, j = 1, \ldots, q_i)$

Satz (Knoop, Steffen):

1. Korrektheit $AI_{ij}^{MOP} \leq AI_{ij}^{MFP}$
2. Volltändigkeit $AI_{ij}^{MOP} \leq AI_{ij}^{MFP}$ falls alle φ_{ij} distributiv