Kapitel 10

Grundlagen der Datenflussanalyse und Programmabhängigkeitsgraphen

Kapitel 10: Grundlagen der Datenflussanalyse und Programmabhängigkeitsgraphen

Halbordnungen

Definition

Eine binäre Relation $\leq$ auf einer Menge M heißt Halbordnung, wenn sie reflexiv, transitiv und antisymmetrisch ist.

Es gilt also:

$$\begin{aligned} &\forall x \in M: && x \leq x \\ &\forall x, y, z \in M: && x \leq y \wedge y \leq z \implies x \leq z \\ &\forall x, y \in M: && x \leq y \wedge y \leq x \implies x = y \end{aligned}$$

Schreibweise:

$$\begin{aligned} x < y &\iff x \leq y \wedge x \neq y \\ x \geq y &\iff y \leq x \end{aligned}$$

Beispiele

- Beispiel 1:

$$(M, \leq) \text{ mit } M = \{a, b, c, d, o, u\}$$

$$u < c, u < d, c < a, c < b, d < a, d < b, a < o, b < o$$

- Beispiel 2: Potenzmenge

$$(\mathcal{P}(M), \subseteq)$$

- Beispiel 3:

$$(\mathbb{N}, \leq)$$

Hasse-Diagramme

- Hasse-Diagramme sind graphische Darstellungen von Halbordnungen:
- eine Aufwärts-Kante $x \to y$ bedeutet $x < y$, wegen Transitivität impliziert ein Aufwärts-Pfad $x \to^* y$ ebenso $x \leq y$
- Halbordnungen sind stets (Aufwärts)-zyklenfrei wegen Antisymmetrie, deshalb kann man die Pfeile an den Kanten weglassen.
- Beweis: Wenn $x \to^* y$ und $y \to^* x$, so $x \leq y$ und $y \leq x$, also $x = y$

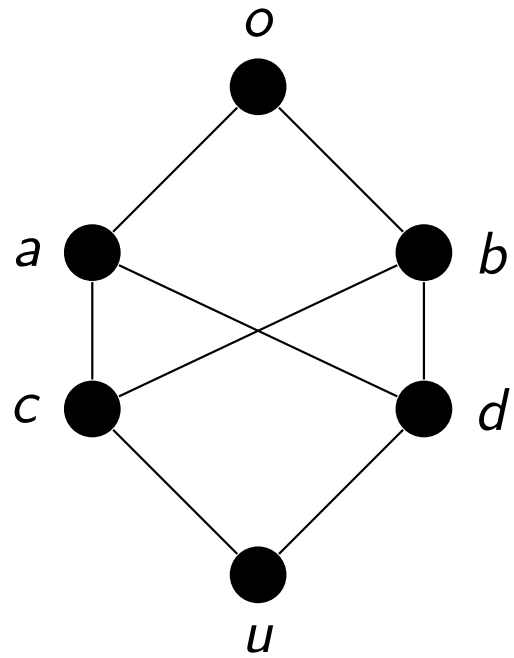


Abbildung: Beispiel 1

Schranken

- Eine Halbordnung heißt total, wenn alle Elemente vergleichbar sind:

$$\forall x, y \in M : x \leq y \vee y \leq x$$

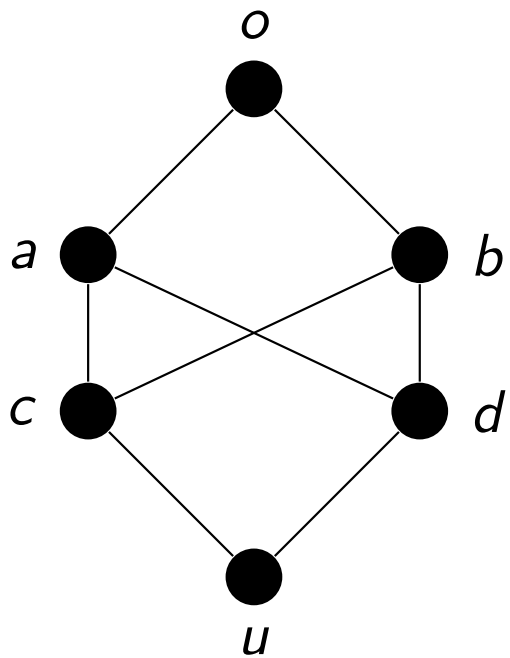
- z heißt obere Schranke zu x, y, wenn $x \leq z \wedge y \leq z$
- z heißt kleinste obere Schranke oder Supremum, wenn jede andere obere Schranke größer ist:

$$\forall o \in M : x \leq o \wedge y \leq o \implies z \leq o$$

- z heißt untere Schranke, wenn $z \leq x \wedge z \leq y$
- z heißt größte untere Schranke oder Infimum, wenn jede andere kleiner ist:

$$\forall o \in M : o \leq x \wedge o \leq y \implies o \leq z$$

Schranken



In Beispiel 1:

- a und b sind obere Schranken zu c, d jedoch keine kleinste obere Schranke, da a, b unvergleichbar sind!
- Obere bzw. untere Schranken muss es nicht immer geben! Bsp: entferne o
- Wenn es eine größte untere Schranke bzw kleinste obere Schranke gibt so sind diese eindeutig.
- Beweis: wenn $x \leq z$, $y \leq z$ und $x \leq z'$, $y \leq z'$, so $z \leq z'$ und $z' \leq z$, also $z = z'$.
- Schreibweise: $x \sqcup y$ (Supremum), $x \sqcap y$ (Infimum).

Verbände

Definition

Eine Halbordnung $(M, \leq)$ heißt Verband, wenn für je zwei Elemente $x, y \in M$ stets $x \sqcup y$ und $x \sqcap y$ existieren. Ferner gibt es ein kleinstes Element $\bot = \bigsqcap M$ und größtes Element $\top = \bigsqcup M$.

Bemerkung: $\bigsqcup M = \bigsqcup_{x \in M} x$

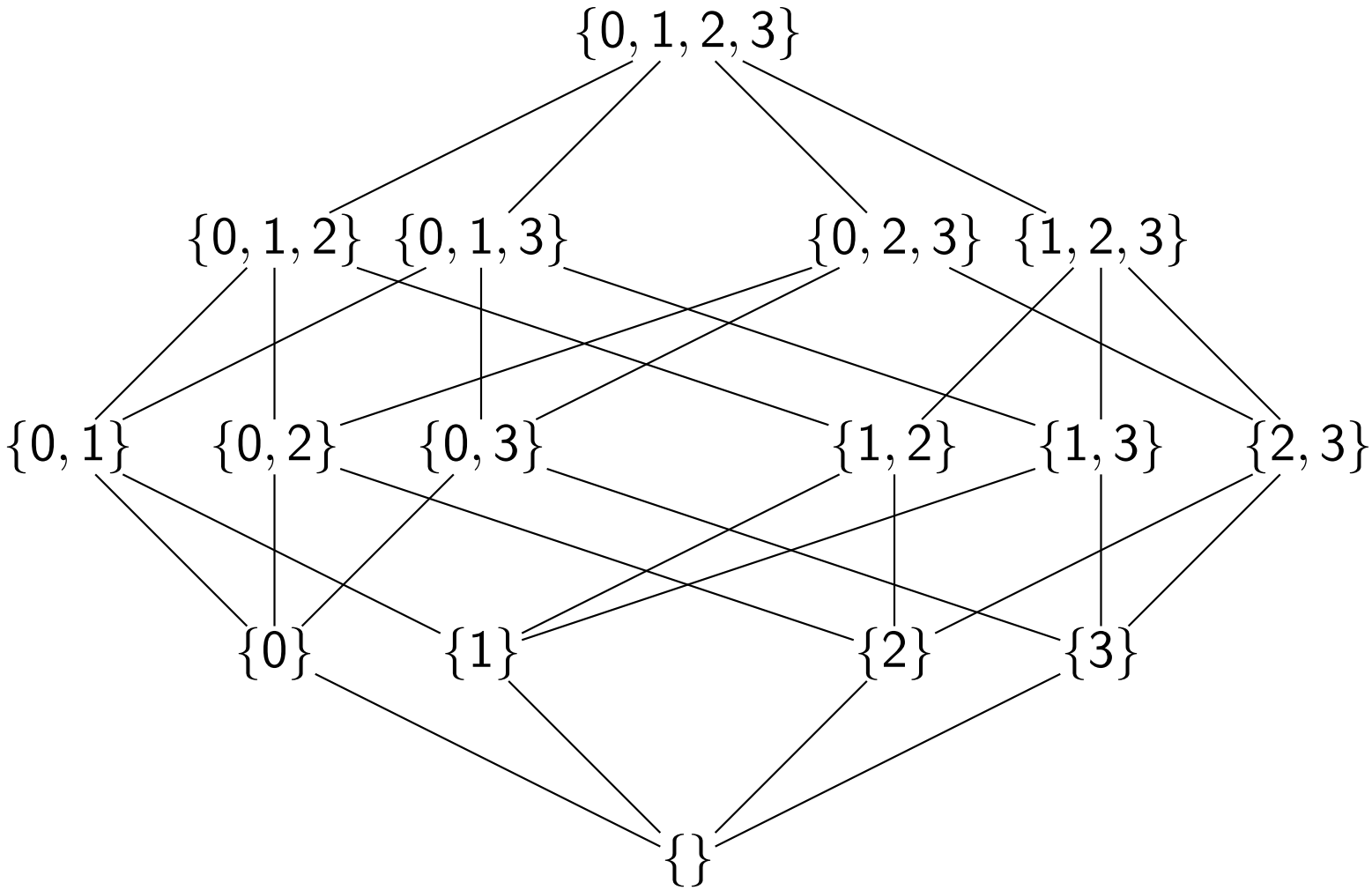
In Verbänden gilt $x \leq y \iff x \sqcup y = y \iff x \sqcap y = x$.

Um die Gleichheit zweier Elemente $x = y$ zu zeigen, wird zumeist $x \leq y$ und $x \geq y$ gezeigt (es sei denn, man kommt mit rein algebraischen Umformungen aus)

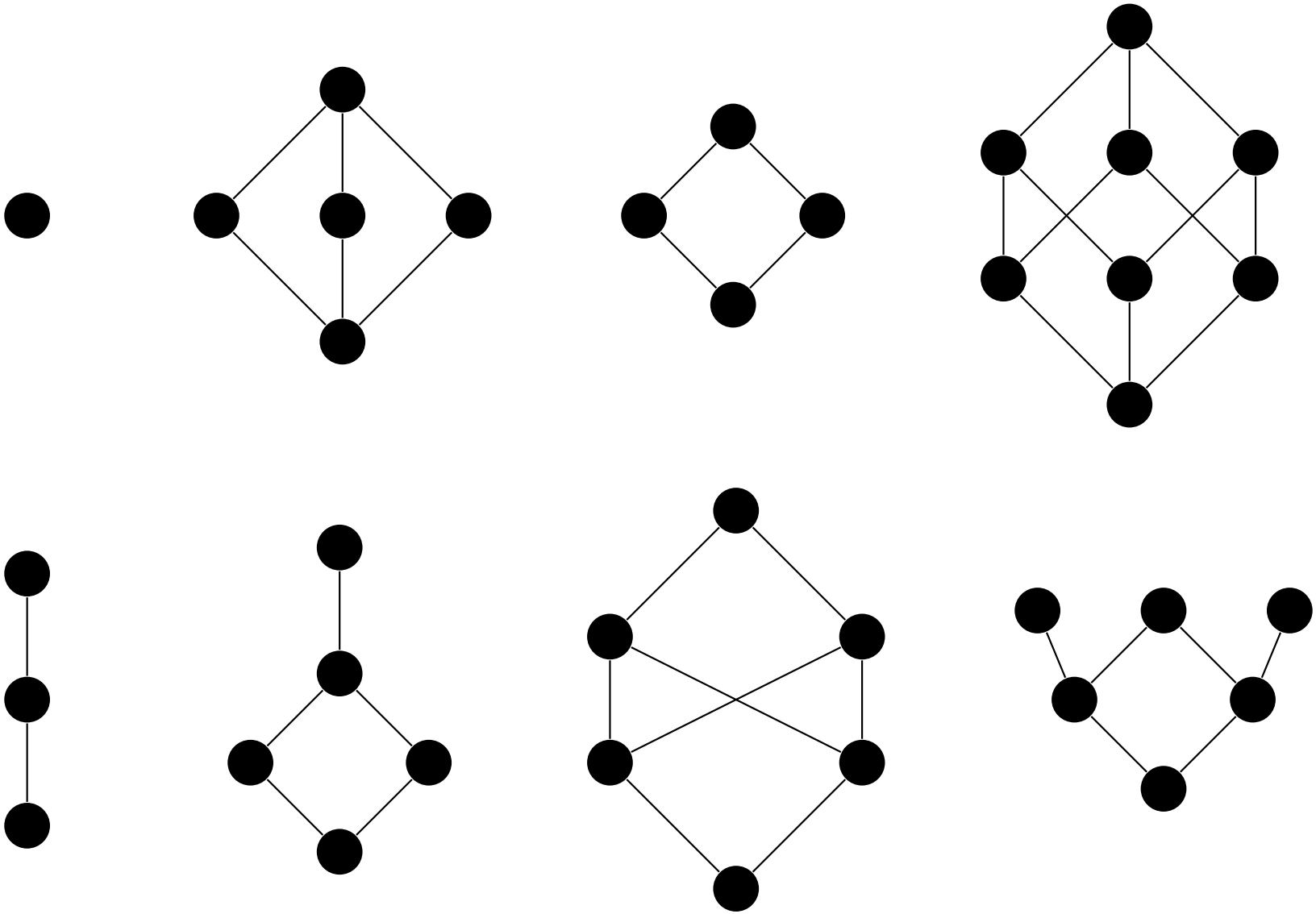
Verbände

- Beispiel 1 ist kein Verband, da a, b, kein Infimum und c, d kein Supremum haben.
- $(\mathcal{P}(M), \subseteq)$ ist Verband mit $\sqcap = \cap, \sqcup = \cup$
- weiteres Beispiel: Begriffsverbände
- Häufig in Beweisen angewandte Eigenschaft: $x \leq z \wedge y \leq z \implies x \sqcup y \leq z$. Analog für $x \sqcap y$.

Beispiel Potenzmengenverband $P(\{0,1,2,3\})$ mit $\subseteq$



Weitere Beispiele: Verband oder Halbordnung?



Rechenregeln

- $x \sqcup x = x$
- $x \sqcup y = y \sqcup x$
- $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$
- $x \sqcup (x \sqcap y) = x$
- $x \sqcap x = x$
- $x \sqcap y = y \sqcap x$
- $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$
- $x \sqcap (x \sqcup y) = x$

Beispielbeweis: Es ist $\forall z : x \sqcup z \geq x$ und deshalb $x \sqcup (x \sqcap y) \geq x$.
Andererseits ist $x \sqcap y \leq x$ und $x \leq x$, deshalb $x \sqcup (x \sqcap y) \leq x$.
Ergo gilt Gleichheit.
In distributiven Verbänden gilt:

- $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
- $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

Definitionen

- Ist $(M; \leq, \sqcap, \sqcup, \top, \bot)$ ein Verband, so ist $(M; \geq, \sqcup, \sqcap, \bot, \top)$ der duale (auf den Kopf gestellte) Verband
- Ein Verband $(M; \leq, \sqcap, \sqcup, \top, \bot)$ heißt vollständig, wenn für beliebige $X \subseteq M$ sowohl $\bigsqcap X$ als auch $\bigsqcup X$ existieren.
- Endliche Verbände sind automatisch vollständig.
- Ein Verband hat endliche Höhe, wenn jeder aufsteigende Pfad $x_1 \leq x_2 \leq \ldots \leq x_i \leq \ldots$ endlich ist.

Beispiel: Für eine beliebige Menge M betrachte

$$(M \cup \{\top, \bot\}, \leq) \text{ wobei } x \leq y \iff x = \bot \vee y = \top \vee x = y$$

Dieser Verband ist der sogenannte flache Verband über M.

Kombinieren von Verbänden

Seien $V_1, \ldots, V_n$ Verbände

Kombination als Produkt

$$V_1 \times ... \times V_n \quad = \quad \{(x_1, .., x_n) \mid x_i \in V_i\}$$

- $\sqcup$, $\sqcap$ komponentenweise: in $V_1 \times V_2$ ist $(x, y) \leq (u, v) \iff x \leq u$ und $y \leq v$
- Höhe$(V_1 \times ... \times V_n)$ = Höhe(V_1) + .. + Höhe(V_n)
- Häufiger Fall in der Programmanalyse: $V^n = V \times ... \times V$ (n-Mal)

Kombination als Summe

$$V_1 + ... + V_n \quad = \quad \{(i, x_i) \mid x_i \in V_i \setminus \{\top, \bot\}\} \cup \{\top, \bot\}$$

- $(i, x) \sqsubseteq (j, y) \iff i = j \wedge x \sqsubseteq y$
- Höhe$(V_1 + ... + V_n) = max($Höhe$(V_1),$ .. , Höhe$(V_n))$

Fixpunkte

Eine Funktion $f : M \to M$ heißt monoton, wenn

$$x \leq y \implies f(x) \leq f(y)$$

x heißt Fixpunkt von f, wenn $f(x) = x$.

Der kleinste Fixpunkt von f wird mit $fix(f)$ bezeichnet.
$fix(f)$ kann explizit berechnet werden:

Satz

In einem Verband endlicher Höhe M hat jede monotone Funktion f einen kleinsten Fixpunkt, und es existiert $k \in N_0$ mit $f^k(\bot) = f^{k+1}(\bot)$ und

$$fix(f) = \bigsqcup_{i \in N_0} f^i(\bot) = f^k(\bot)$$

Beweis Fixpunktsatz

- Es ist $\bot \leq f(\bot)$, wegen Monotonie auch $f(\bot) \leq f(f(\bot))$. Mit Induktion folgt:

$$\bot \leq f(\bot) \leq f^2(\bot) \leq \ldots \leq f^i(\bot) \leq f^{i+1}(\bot) \leq \ldots \leq \top$$

- Verband endlicher Höhe $\Rightarrow$ $f^i(\bot)$ nach Schubfachprinzip nicht alle verschieden.
- Daher gibt es k, j mit $f^k(\bot) = f^{k+j}(\bot)$, sogar $f^k(\bot) = f^{k+1}(\bot) = f(f^k(\bot))$, da alle Elemente auf dem Pfad von k bis $k + j$ gleich.

  $\Rightarrow f^k(\bot)$ ist Fixpunkt von f.

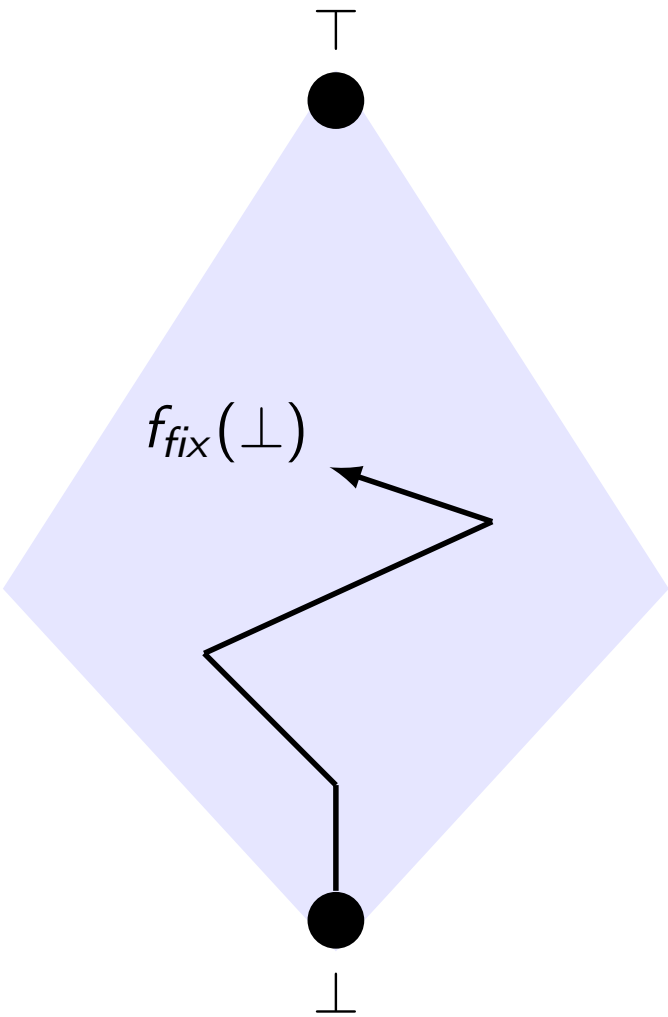
Beweis Fixpunktsatz

- Angenommen $f(z) = z$, also z weiterer Fixpunkt. Dann ist

$$\begin{array}{lll} \bot & & \leq z \\ f(\bot) & \leq f(z) & = z \\ f(f(\bot)) & \leq f(z) & = z \\ & \vdots & \\ f^i(\bot) & & \leq z \\ \Longrightarrow\ f^k(\bot) & & \leq z \end{array}$$

$\Rightarrow f^k(\bot)$ ist kleinster Fixpunkt von f.

Fixpunktberechnung



Komplexität hängt von 3 Faktoren ab:

- Höhe des Verbandes $\rightarrow$ Obere Schranke *#Iterationen*
- Kosten der Berechnung der Funktionswerte von f
- Kosten des Vergleichs auf Gleichheit $f^k(\bot) = f^{k+1}(\bot)$

Anwendung: Lösen von Gleichungssystemen

Sei V ein Verband endlicher Höhe, x_i Variablen und $f_i : V^n \to V$ monotone Funktionen. Gesucht: Lösung des Gleichungssystems

$$\begin{aligned} x_1 &= f_1(x_1, .., x_n) \\ &... \\ x_n &= f_n(x_1, .., x_n) \end{aligned}$$

Gleichungssysteme dieser Art haben eine eindeutige kleinste Lösung

Berechenbar als Fixpunkt der Funktion $F : V^n \to V^n$

$$\begin{aligned} F(x_1, .., x_n) &= (f_1(x_1, .., x_n), .., f_n(x_1, .., x_n)) \\ &= (x_1, .., x_n) \end{aligned}$$

Fixpunkt Algorithmen

Beispiel: Kontrollflussgraph

Seien die Knoten des CFG $N = \{v_1, \ldots, v_n\}$ und der Verband V gegeben. $[\![]\!] : N \to V$ weist CFG-Knoten ein Verbandselement zu. Je Knoten v_i eine Datenflussgleichung: $[\![v_i]\!] = f_i([\![v_1]\!], \ldots, [\![v_n]\!])$

Es gilt den Fixpunkt folgender Funktion zu bestimmen ($x_i = [\![v_i]\!]$):

$$\begin{aligned} F : V^n &\to V^n \\ F(x_1, \ldots, x_n) &= (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) \end{aligned}$$

Naiver Ansatz

```
x = (⊥, ..., ⊥);
do {
    t = x;
    x = F(x);
} while (x ≠ t);
```

Chaotic Iteration

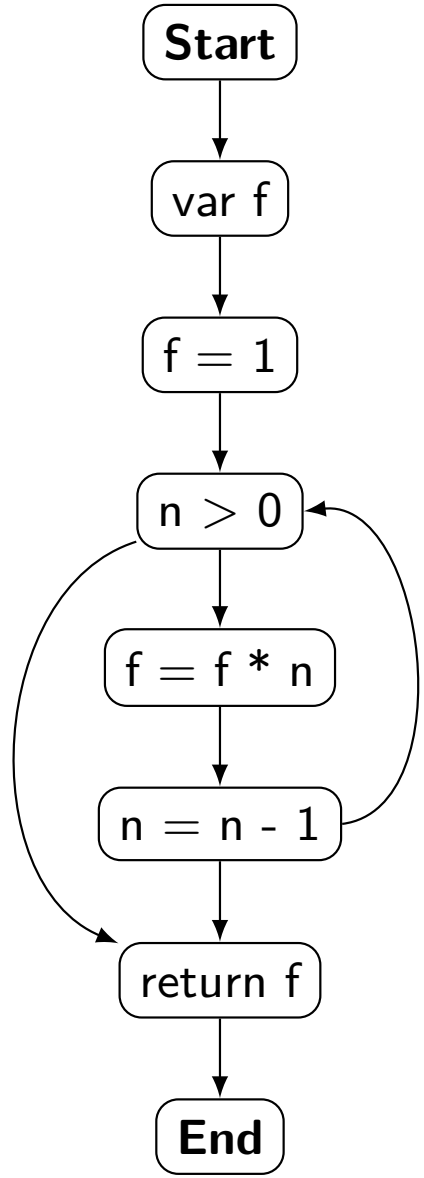
```
x_1 = ⊥; ... x_n = ⊥;
do {
    t_1 = x_1; ... t_n = x_n;
    x_1 = f_1(x_1, ..., x_n);
    ...
    x_n = f_n(x_1, ..., x_n);
} while (x_1 ≠ t_1 ∨ .... ∨ x_n ≠ t_n);
```

Fixpunkt Algorithmen

Naiver Ansatz und *Chaotic Iteration* berechnen alle Knoten in jeder Iteration $\rightarrow$ Geht es besser?

Worklist Algorithmus

$x_1 = \bot; \ldots x_n = \bot;$
q $= [v_1, \ldots, v_n];$
while $(q \neq [])$ {
    assume q $= [v_i, \ldots];$
    y $= f_i(x_1, \ldots, x_n);$
    q $=$ q.tail();
    if $(y \neq x_i)$ {
        for $(v \in succs(v_i))$ { q.append(v);}
        $x_i =$ y;
    }
}



Galois-Verbindungen

wichtig für Theorie der Programmanalyse

$(P, \leq)$ und $(Q, \leq)$ seien Halbordnungen, $\alpha : P \to Q$, $\gamma : Q \to P$ Funktionen
(α, γ) heißt Galois-Verbindung, wenn

1. $x \leq y \implies \alpha(x) \geq \alpha(y)$
2. $u \leq v \implies \gamma(u) \geq \gamma(v)$
3. $x \leq \gamma(\alpha(x))$ und $u \leq \alpha(\gamma(u))$

α und γ sind also beide antimonoton. P und Q sind oft Potenzmengenverbände oder Verbände endlicher Höhe (s.o.)

Galois-Verbindungen

Lemma. (α, γ) sind Galois-Verbindung genau dann wenn

$$4. \quad x \leq \gamma(u) \iff u \leq \alpha(x)$$

Beweis. " $\Longrightarrow$ ": a. Sei $x \leq \gamma(u)$. Dann ist nach 1. und 3. $\alpha(x) \geq \alpha(\gamma(u)) \geq u$. b. analog
"$\Longleftarrow$": Es ist $\alpha(x) \leq \alpha(x)$ und nach 4. $x \leq \gamma(\alpha(x))$, also 3a. 3b analog. Sei nun $x \leq y$. Da $y \leq \gamma(\alpha(y))$, ist $x \leq \gamma(\alpha(y))$ und nach 4. $\alpha(y) \leq \alpha(x)$, also 1. 2. analog.

Lemma.

$$\alpha(x) = \alpha(\gamma(\alpha(x))) \text{ und } \gamma(u) = \gamma(\alpha(\gamma(u)))$$

Beweis. "$\leq$": Sei $v = \alpha(x)$. Dann ist nach 3b. $v \leq \alpha(\gamma(v))$.
"$\geq$": Es ist $x \leq \gamma(\alpha(x))$ und mit 1. $\alpha(x) \geq \alpha(\gamma(\alpha(x))$. Teil b analog.

Korollar. $\alpha \circ \gamma$ ist ein Hüllenoperator, also $x \leq \alpha(\gamma(x))$, $\alpha(\gamma(x)) = \alpha(\gamma(\alpha(\gamma(x))))$, und $x \leq y \implies \alpha(\gamma(x)) \leq \alpha(\gamma(y))$.
$\gamma \circ \alpha$ ist auch ein Hüllenoperator.

Begriffsverbände

Mathematische Begriffsanalyse (Ganter & Wille):

- analysiert Relation R zwischen "Objekten" und "Attributen"
- visualisiert versteckte Struktur in dieser Relation
- transformiert Relation in einen Begriffsverband

$\Rightarrow$ hierarchische Gruppierung von Objekten und Attributen

$\Rightarrow$ Abhängigkeiten, Interferenzen, Zerlegbarkeiten

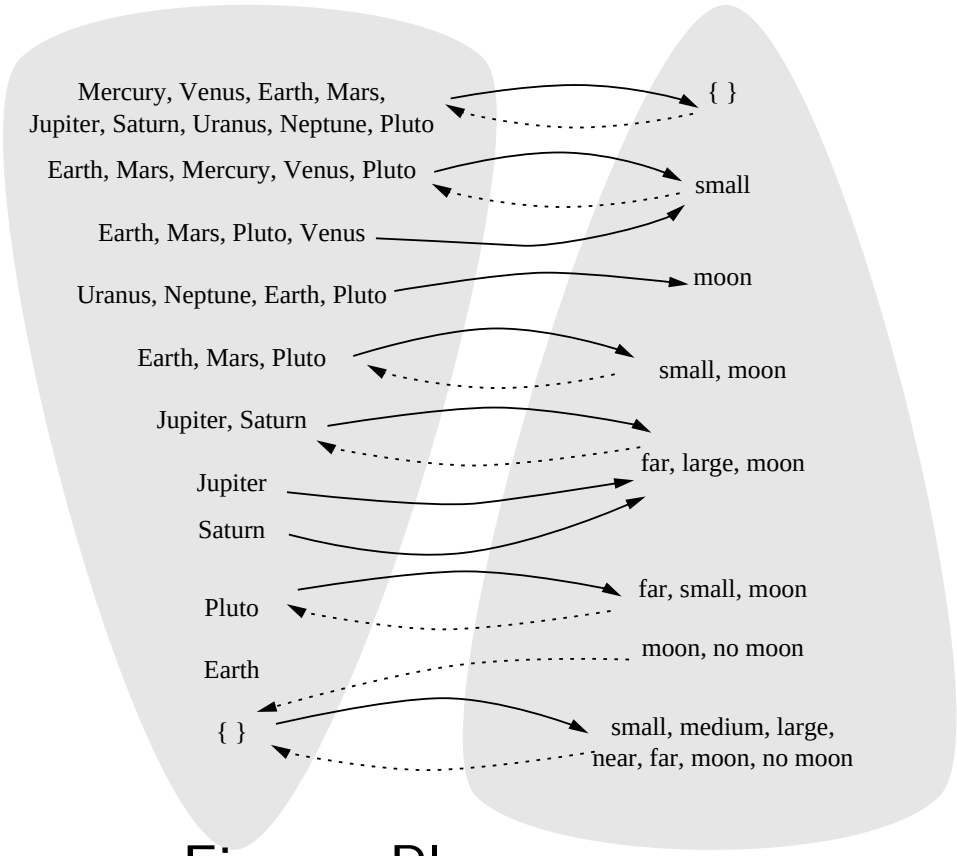
Beispiel

Relation $R \subseteq \text{Pla} \times \text{Eig}$ zwischen Planeten und Eigenschaften:

	small	medium	large	near	far	moon	no moon
Mercury	×			×			×
Venus	×			×			×
Earth	×			×		×	
Mars	×			×		×	
Jupiter			×		×	×	
Saturn			×		×	×	
Uranus		×			×	×	
Neptune		×			×	×	
Pluto	×				×	×	

Beispiel/2

Gemeinsame Attribute / Gemeinsame Objekte:



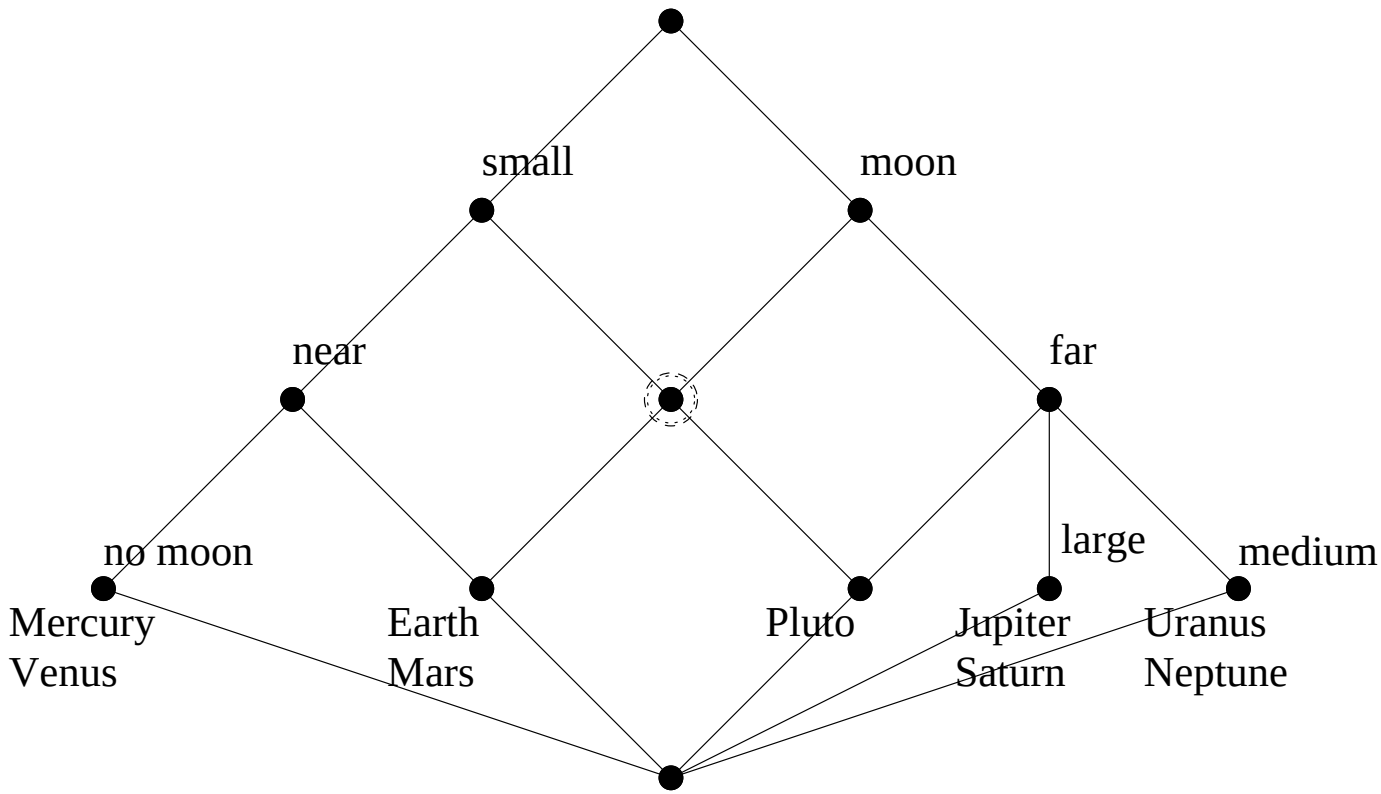
$\alpha : 2^{\text{Pla}} \to 2^{\text{Eig}}$, $\gamma : 2^{\text{Eig}} \to 2^{\text{Pla}}$

$\alpha(P) = \{a \in \text{Eig} \mid \forall p \in P : (p, a) \in R\}$

γ analog. Dies ist eine Galoisverbindung! Beweis: Übung

Begriffsverband

maximale Rechtecke in der Tabelle werden zu Verbandselementen
(Zeilen-/Spaltenpermutationen irrelevant)
maximale Rechtecke: (P, E) mit $E = \alpha(P), P = \gamma(E)$
Details / Algorithmus nichttrivial!



Eigenschaften des Verbandes

- Transformation der Originaltabelle: $\nu(o)$ ist das mit o markierte Verbandselement, $\mu(a)$ das mit a markierte

$$(o, a) \in T \iff \nu(o) \leq \mu(a)$$

- Suprema faktorisieren gemeinsame Eigenschaften heraus: *"Mars und Venus sind beide nah"*
- Infima faktorisieren gemeinsame Objekte heraus: *"Pluto ist sowohl klein als auch weit"*
- Aufwärtskanten sind Implikationen: *"Planeten ohne Mond sind auch nah und klein"*

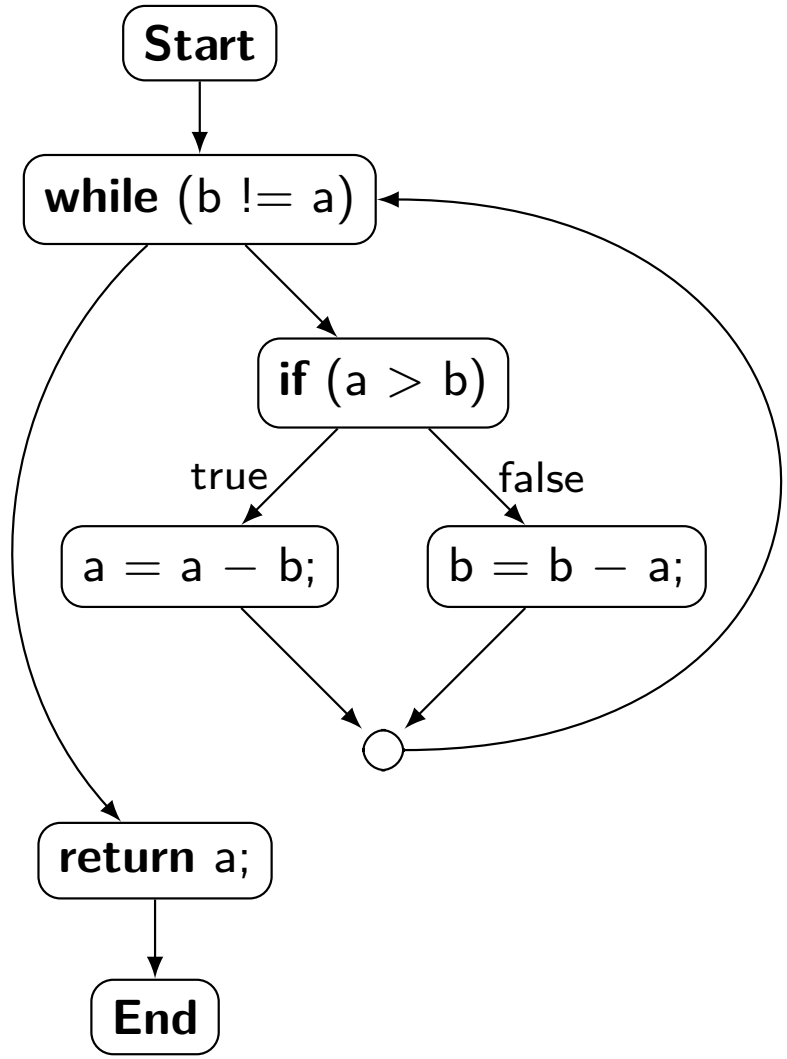
Kapitel 10: Grundlagen der Datenflussanalyse und Programmabhängigkeitsgraphen

Wiederholung: Kontrollflußgraph (CFG)

- Jeder Grundblock besitzt einen entsprechenden Knoten im Kontrollflußgraph.
- Die Kanten zwischen den Knoten repräsentieren mögliche Sprünge zwischen den Grundblöcken.
- Es existieren 2 zusätzliche Knoten **Start** und **End** (auch Startblock und Endblock genannt).
- Es gibt eine Kante von **Start** zu jedem Grundblock mit dem das Programm betreten werden kann.
- Es gibt eine Kante von jedem Grundblock mit dem das Programm verlassen werden kann zu **End**.

Beispiel Kontrollflußgraph

```
int gcd(int a, int b)
{
    while(b != a) {
        if(a > b)
            a = a − b;
        else
            b = b − a;
    }
    return a;
}
```



Korrektheitsbedingung Kontrollflußgraph

$\forall$ Eingabe z: Programm durchläuft bei Eingabe z dynamisch die Folge von Grundblöcken

$$W = (\textbf{Start}, b_1, b_2, \ldots, b_n, \textbf{End})$$

$$\Longrightarrow \; W \text{ ist ein Pfad im CFG.}$$

Beachte: Die umgekehrte Implikation muss nicht gelten. Dh CFG darf zuviele Pfade enthalten (zB toter Code), aber niemals zuwenig! CFG ist *konservative Approximation*. Natürlich will man, dass CFG möglichst wenig nicht dynamisch realisierbare Pfade enthält (*Präzision*). CFGs für strukturierten Steuerfluss (nur 1 Ein-Ausgang je Konstrukt, zB IF, WHILE, ...) sind i.a. präzise.

Dominanz

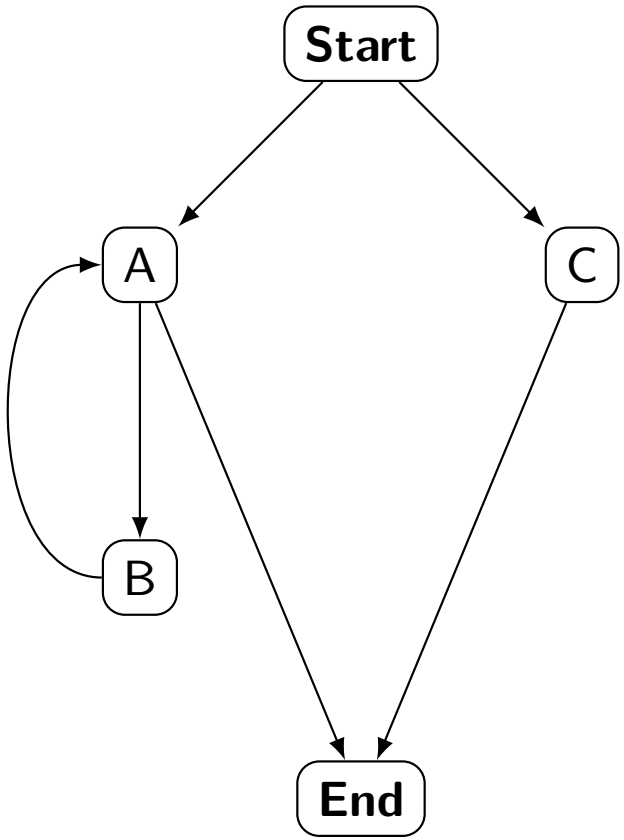
Definition Dominanz

X dominiert Y, wenn jeder Pfad im Kontrollflußgraph von **Start** zu Y auch X enthält. Wir schreiben $X \preceq Y$ (oft auch X dom Y)

Definition Postdominanz

Y postdominiert X, wenn jeder Pfad im Kontrollflußgraph von X zu **End** auch Y enthält. Wir schreiben Y postdom X.

Dominanz Beispiel



- **Start** $\preceq B$, $A \preceq B$
- $\neg(A \preceq C)$, $\neg(A \preceq$ **End**$)$
- **End** postdom B,
  End postdom C,
  A postdom B

Dominanz Anwendungen

- Schleifeneintrittspunkt $\preceq$ Schleifenrumpf
- Schleifenaustrittspunkt postdom Schleifenrumpf
- $X \preceq Y \wedge Y$ postdom $X \implies X, Y$ werden stets zusammen ausgeführt
- Steuerflussregion: Bereich zwischen X und Y
- strukturierte Programme: Konstrukte (z.B. Schleifen) haben genau einen Eintritts- und einen Austrittspunkt
- Dominatorbaum: liefert Schachtelung von Schleifen, If-Anweisungen, u.ä. $\leftrightarrow$ "*Intervallanalyse*" [Tarjan]

Dominanz Eigenschaften

- Dominanz ist reflexiv: $X \preceq X$
- Dominanz ist transitiv: $X \preceq Y \wedge Y \preceq Z \Rightarrow X \preceq Z$.
- strikte Dominanz: $X \prec Y := X \preceq Y \wedge X \neq Y$.
- direkte Dominanz (englisch immediate dominator):

$$X = \text{idom}(Y) := X \prec Y \wedge \neg\exists Z : X \prec Z \prec Y$$

- Jeder Block außer dem Startblock hat genau einen direkten Dominator $\Rightarrow$ direkte Dominatoren bilden einen Baum, den Dominatorbaum.
  Dominatorbaum ist spezieller aufspannender Baum.
- Postdominanz ist ebenfalls reflexiv und transitiv. Definition von strikter Postdominanz, direkter Postdominanz und dem Postdominanzbaum analog.

GOTO-Elimination

Satz(Böhm, Jacopini 1964): Jedes Programm kann nur durch While / If-Then-Else / Begin-End dargestellt werden. [1]
$\Rightarrow$ Ersatz von GOTO durch strukturierte Schleifen.

Die so entstehenden Programme sind jedoch softwaretechnisch pervers

[1] Beweisidee: Durchnummerieren der Grundblöcke im CFG, "Program Counter"

Beispiel

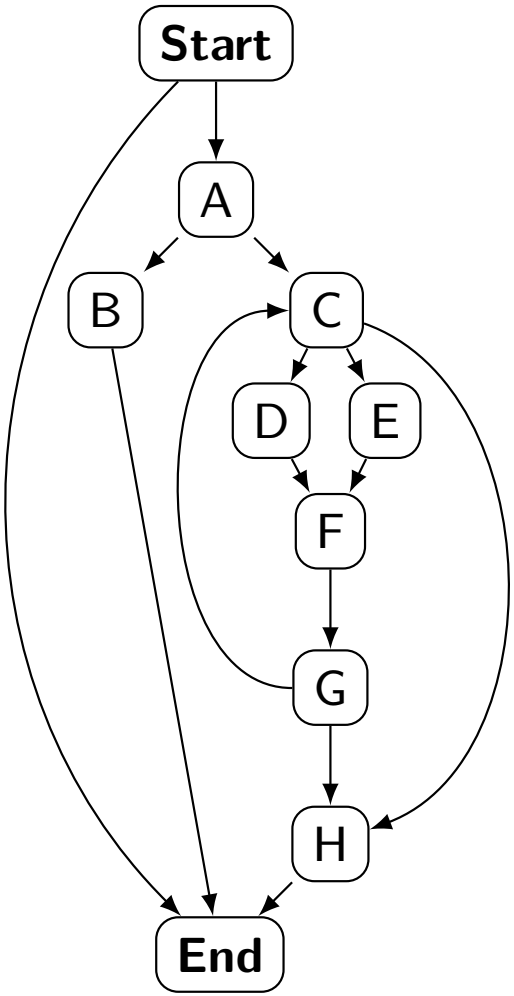


Abbildung: Kontrollflußgraph

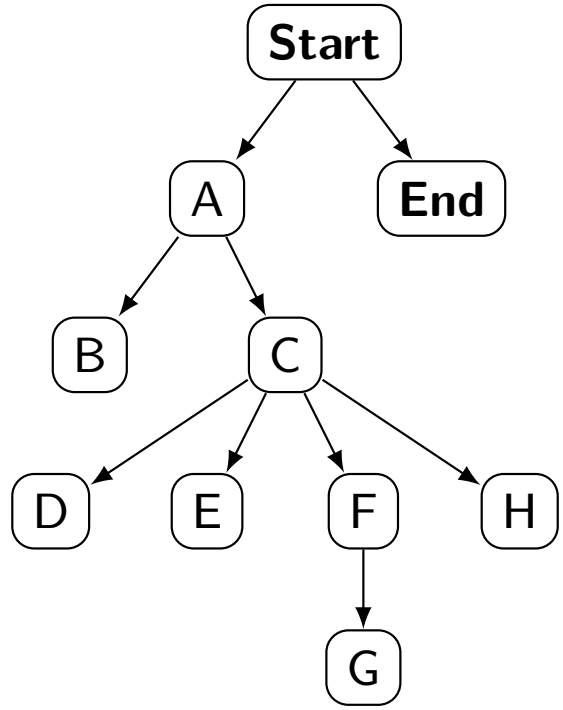


Abbildung: Dominanzbaum

GOTO-Elimination - Identifikation geschachtelter Schleifen

1. Dominatoren, zu denen Rückwärtskanten führen bilden Schleifeneintrittspunkte.
2. Postdominatoren der Eintrittspunkte bilden Schleifenaustrittspunkte.
3. Schleifen werden von innen nach aussen identifiziert (bottom-up Durchlauf im Dominatorbaum).
4. Falls 1./2. nicht möglich so entspricht die Schleife einer starken Zusammenhangskomponente.
5. Kombination von Dominatoren und Zusammenhangskomponenten: finde zuerst strukturierte Schleifen.
6. Evtl. aufbrechen des Graphen und Kopieren von Knoten / Einfügen zusätzlicher Kontrollflags.

GOTO-Eliminiation - Beispiel

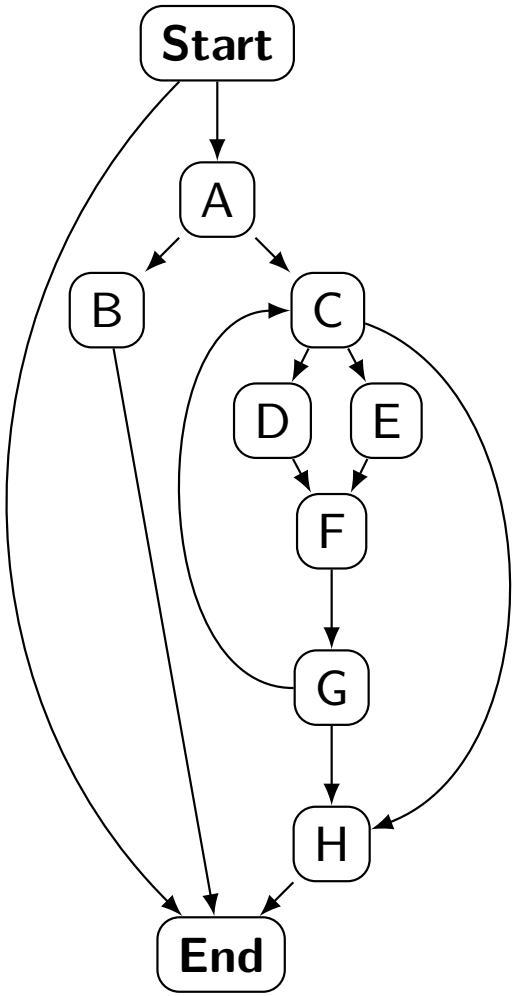


Abbildung: Kontrollflußgraph

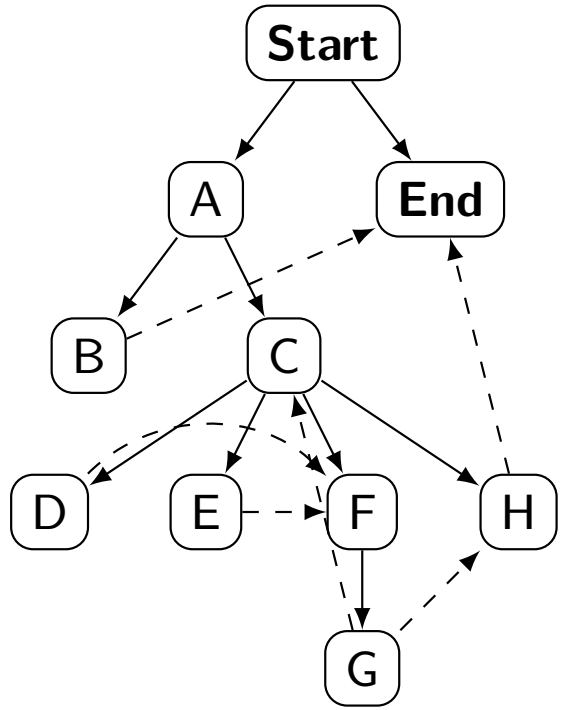
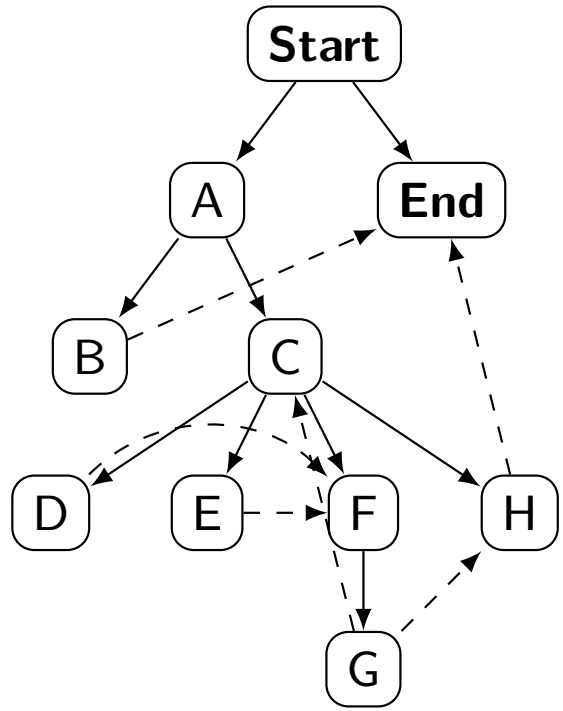


Abbildung: Dominanzbaum mit Ablaufkanten

GOTO-Eliminiation - Beispiel

```
if( A ) {
    B;
} else {
    while( C1 && !G ) {
        if( C2 ) {
            D;
        } else {
            E;
        }
        F;
    }
    H;
}
```



GOTO-Eliminiation - Beispiel

weiteres Beispiel: Einfügen der Kante (B, D)

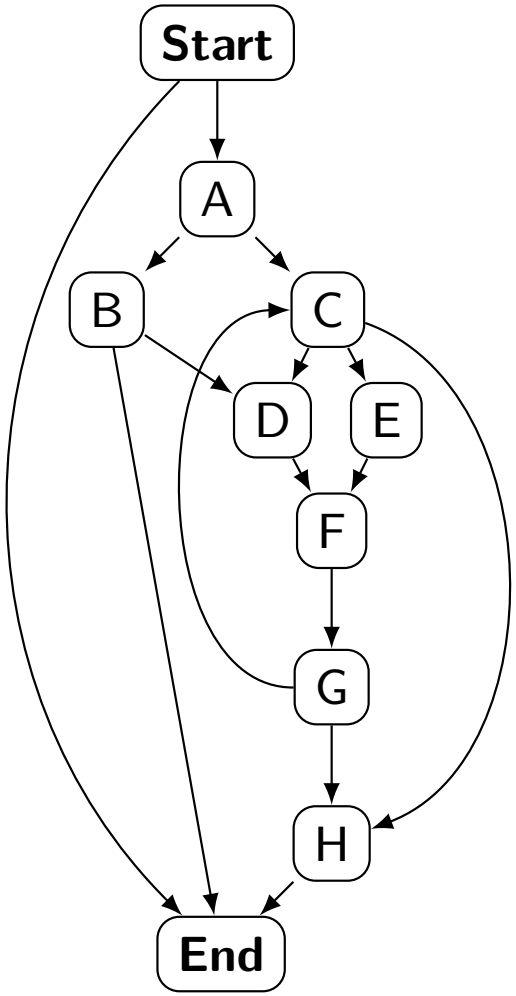


Abbildung: Kontrollflußgraph

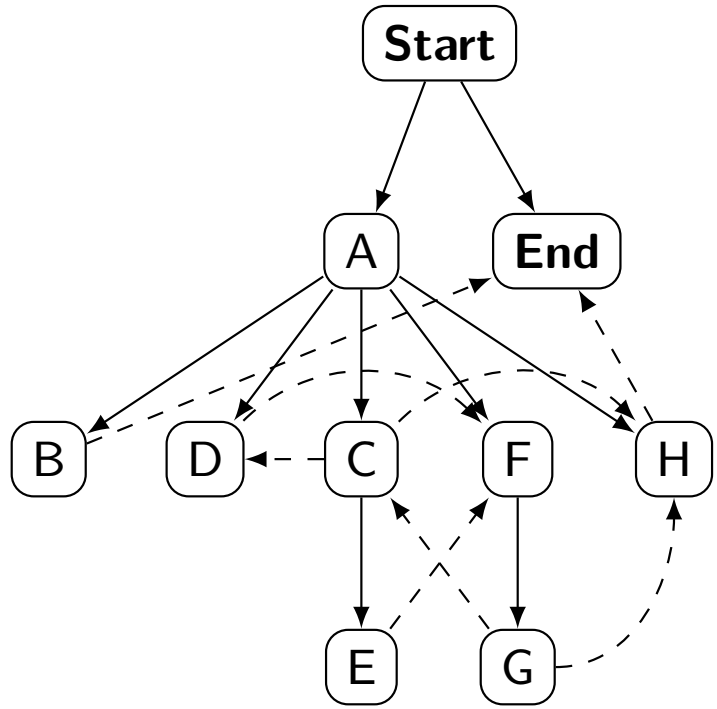
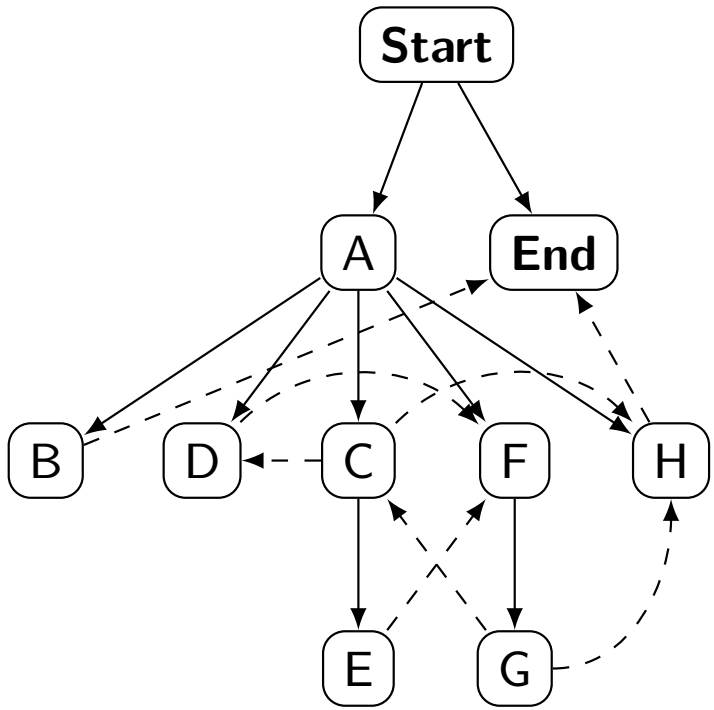


Abbildung: Dominanzbaum mit Ablaufkanten

GOTO-Eliminiation - Beispiel

```
if ( A ) {
      flag1 = true;
      if ( B ) { flag2 = true; }
      else { flag2 = false; }
} else { flag1 = false; }

if ( !flag1 || flag2) {
      while ( C1 && !G || flag2) {
            if ( C2 || flag2 ) {
                  D; flag2 = false;
            } else { E; }
            F;
      }
      H;
}
```



Berechnung mit Fixpunktiteration

Die Menge aller Dominatoren von X lässt sich darstellen als

$$\mathrm{dom}(X) = \{X\} \cup \bigcap_{Y \in \mathrm{pred}(X)} \mathrm{dom}(Y)$$

$\Longrightarrow$ Berechnung mit Fixpunktiteration möglich (vgl Kapitel Datenflussanalyse). Worst-Case Laufzeit quadratisch bis kubisch.

Zugrundeliegender Verband: Potenzmengenverband der Grundblöcke (oder Anweisungsnummern $x_1, x_2, \ldots, x_n$)
Mit $z_i = dom(x_i)$ ist

$$f_i(z_1, z_2, \ldots, z_n) = \{x_i\} \cup \bigcap_{x_j \in \mathrm{pred}(x_i)} z_j$$

wobei f_i nicht von allen z_j abhängt, sondern nur von Vorgängern von x_i
f_i sind monoton (wieso?), $F = (f_1, f_2, \ldots, f_n)$ hat deshalb Fixpunkt

Der Lengauer-Tarjan-Algorithmus

Lengauer, T. und Tarjan, R. E. *A Fast Algorithm for Finding Dominators in a Flowgraph* (1979)

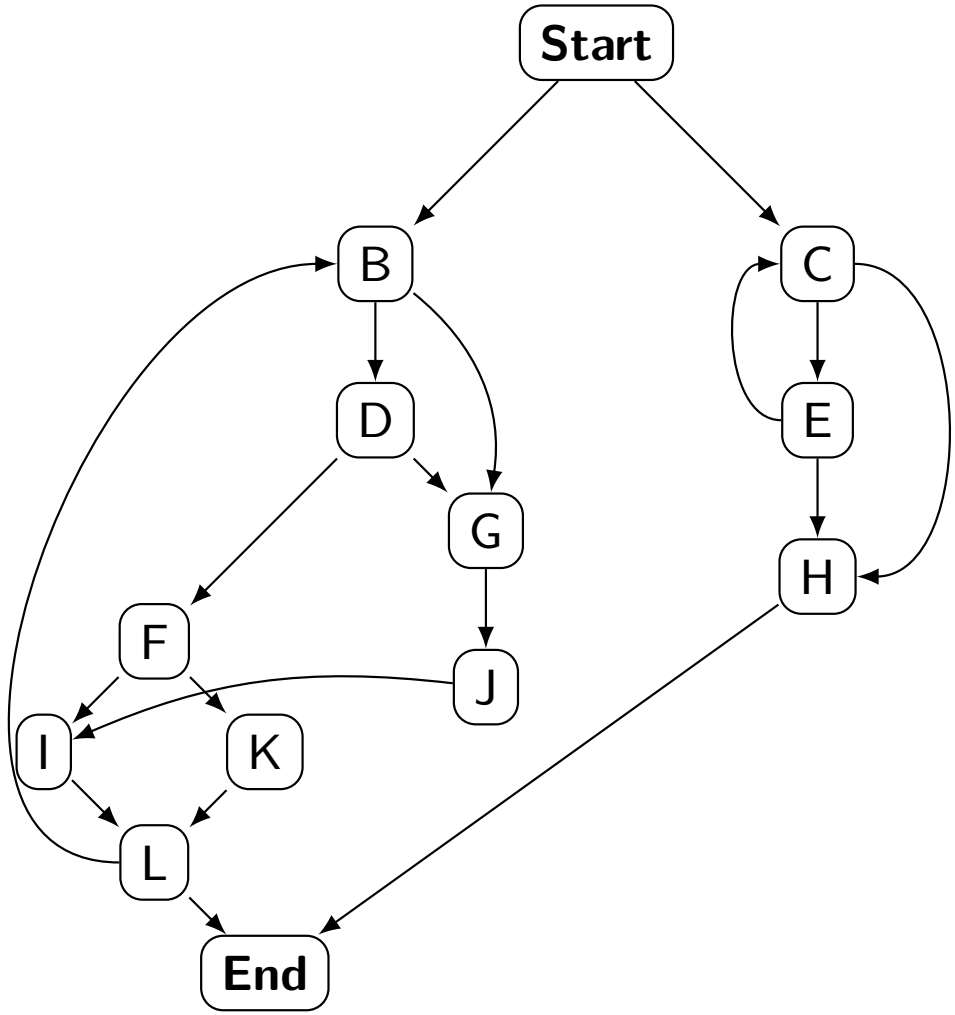
Tiefensuchbaum

Nimmt man die bei einer Tiefensuche besuchten Kanten und Knoten, so entsteht ein Tiefensuchwald. Auf Kontrollflußgraphen ein Tiefensuchbaum.

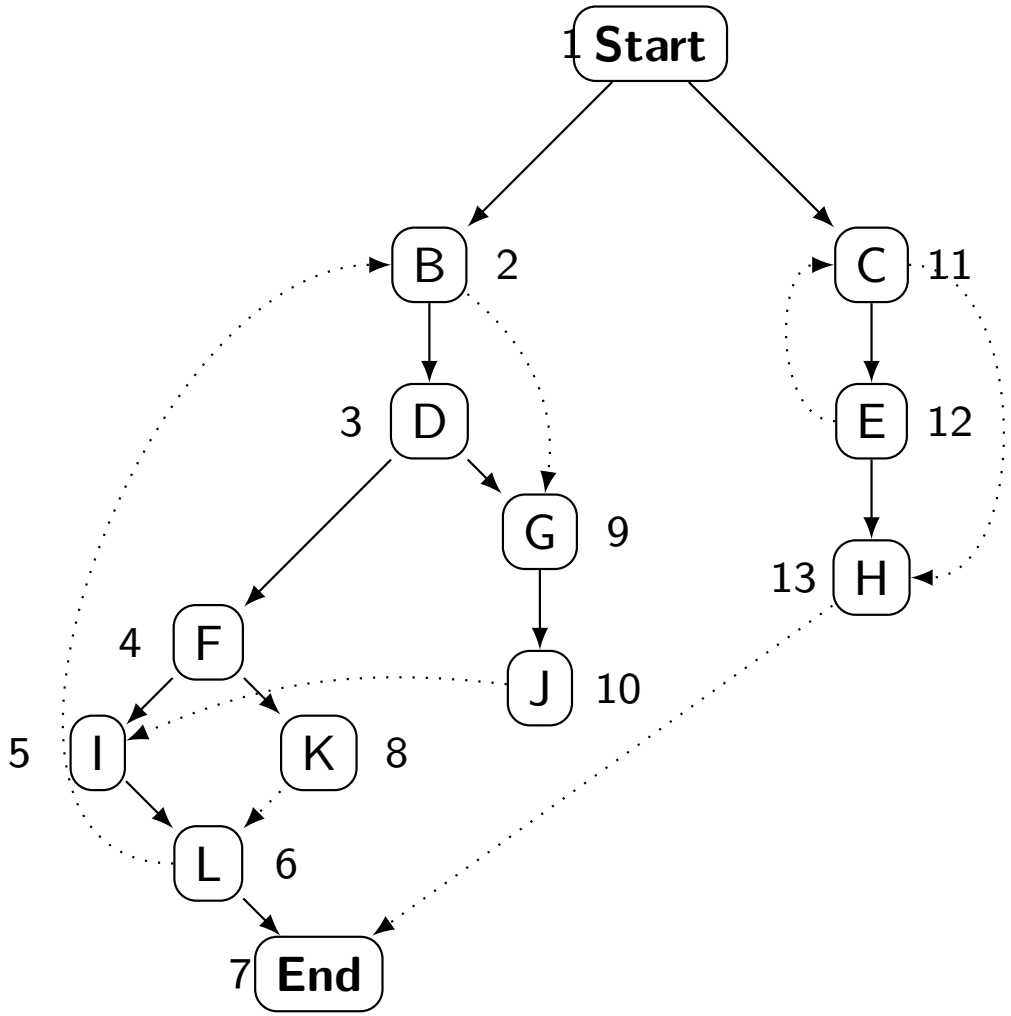
Tiefensuchnummern

Nummerieren der Knoten in Besuchsreihenfolge (pre-order Numbering) legt die Tiefensuchnummern der Knoten fest. Schreibweise: dfnum(x)

Beispiel - Kontrollflußgraph



Beispiel - Tiefensuchbaum mit Tiefensuchnummern



Tiefensuchbäume - Eigenschaften

Betrachte Knoten $n \neq$ **Start** und seinen Pfad im Spannbaum:

- Für alle Knoten k auf dem Pfad gilt: $dfnum(k) \leq dfnum(n)$.
- Alle Dominatoren von n liegen auf dem Pfad (folgt aus Definition des Dominators). Damit gilt: $d \preceq n \Rightarrow dfnum(d) \leq dfnum(n)$

$\Rightarrow$ Berechnung des direkten Dominators von n durch Testen der Vorgänger auf dem Pfad.

- Ein Knoten x des Pfades ist genau dann kein Dominator von n, wenn es alternative Pfade zu n um x herum gibt.
- Sei Pfad $a \rightarrow^* b$ vorhanden. Entweder ist $dfnum(a) < dfnum(b)$, dann liegt a im Tiefensuchbaum oberhalb b (Schreibweise: a anc b). Oder es ist $dfnum(a) > dfnum(b)$, dann wurde a später besucht und liegt „rechts neben“ b
- weiss man also, dass Pfad $a \rightarrow^* b$ vorhanden ist, so kann man mit dfnum-Vergleich einfach testen, ob a anc b

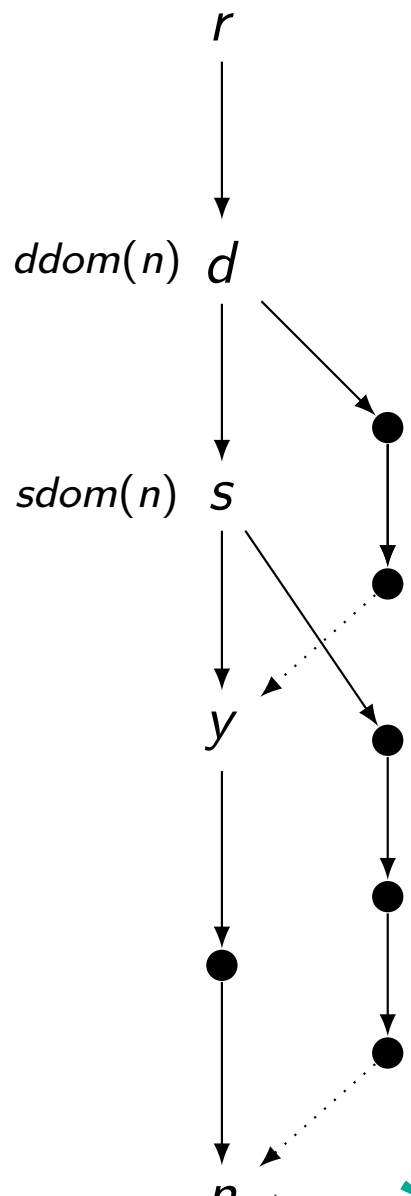
Semidominatoren

Beachte: Im folgenden identifizieren wir Knoten mit ihren Tiefensuchnummern!

Ein Semidominator ist definiert als

$$sdom(n) = min\{m | P = m, v_1, \ldots, v_k, n \text{ ist Pfad mit } v_i > n \text{ für } 1 \leq i \leq k\}$$

- $sdom(n)$ ist kleinster (i.e. oberster) Vorgänger von n auf dem Tiefensuchpfad, von dem Pfad $s \rightarrow^* n$ außerhalb des Tiefensuchpfades („Umleitung“) existiert.
- Der Semidominator muss nicht der direkte Dominator sein. (Beispiel siehe Bild).
- aber unterhalb $sdom(n)$ kann $ddom(n)$ nicht liegen!



Berechnung des Semidominators von n

$sdom(n)$ wird aus Semidominatoren der Vorgänger von n berechnet. Betrachte dazu jeden Vorgänger v von n.
$sdom(n)$ ist die „oberste Umleitung“ nach n über ein v. Für jedes v wird Menge von Kandidaten bestimmt, am Ende alle Kandidatenmengen vereinigt und darin $sdom(n)$ bestimmt

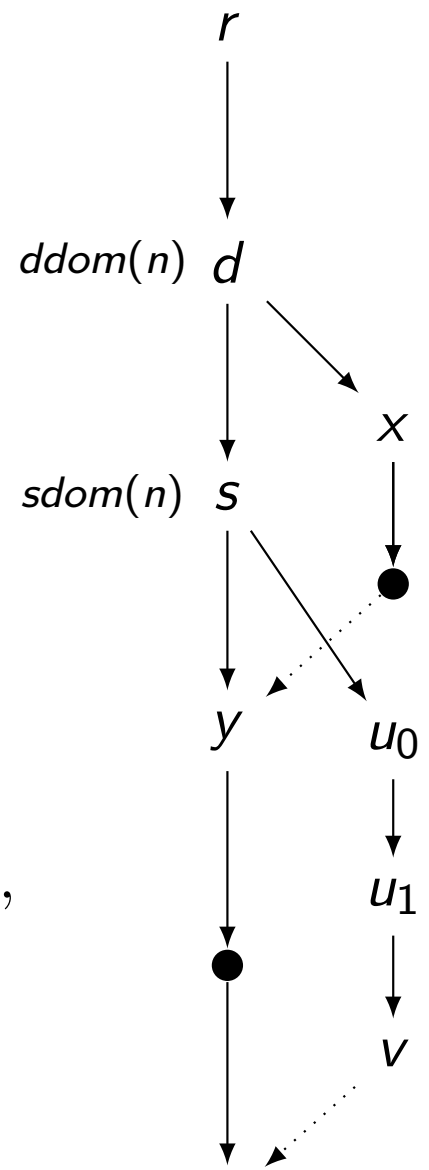
Fall 1:
Liegt v auf dem Tiefensuchpfad nach n (also $\text{dfnum}(v) < \text{dfnum}(n)$) so ist v ein Kandidat (denn v ist evtl der einzige Vorgänger)

Berechnung des Semidominators von $n/2$

Fall 2:
Liegt v nicht auf dem Tiefensuchpfad nach n (also $\text{dfnum}(v) > \text{dfnum}(n)$) so betrachte alle u auf dem Tiefensuchpfad nach v (incl v); für diese gilt $\text{dfnum}(u) \leq \text{dfnum}(v)$.
Alle $sdom(u)$ sind Kandidaten, denn es gibt dann Umleitung $sdom(u) \rightarrow^* u$, also auch Umleitung $sdom(u) \rightarrow^* u \rightarrow^* v \rightarrow n$, dh unterhalb dieser Kandidaten kann $dom(n)$ nicht liegen. Deshalb ist

$$sdom(n) = min\{sdom(u) \mid dfnum(u) \leq dfnum(v), v \rightarrow n \in CFG\}$$

$sdom(n)$ ist der Kandidat mit der kleinsten Tiefensuchnummer.



Implementierung Semidominatorberechnung

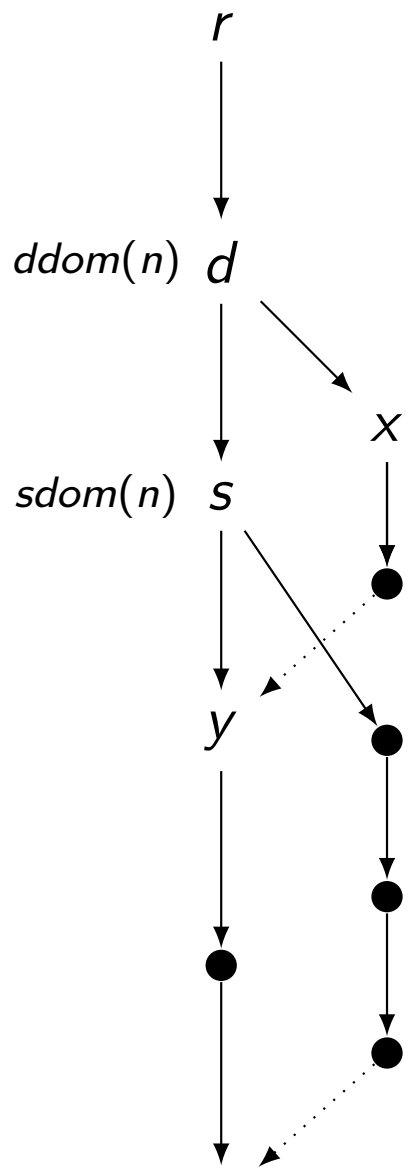
Implementierung: top-down Traversierung des Tiefensuchbaums; wenn man zu n kommt, sind die $sdom(u)$ bereits berechnet; man muss nur noch die Ketten $n \to v \to^* u \ldots$ rückwärts laufen und dabei Minimum der $dfnum(sdom(u))$ bestimmen

Dominator Theorem

Auf dem Tiefensuchpfad zwischen $sdom(n)$ und n einschließlich, sei y derjenige Knoten dessen Semidominator die kleinste Tiefensuchnummer besitzt. Dann ist

$$ddom(n) = \begin{cases} sdom(n) & \text{if } sdom(y) = sdom(n) \\ ddom(y) & \text{if } sdom(y) \neq sdom(n) \end{cases}$$

Also ist $ddom(n) = sdom(y)$ mit $y \in sdom(n) \rightarrow^* n$ und $sdom(y)$ minimal (i.e. möglichst weit oben)
Beweis: LT79



Dominator Theorem/2

Zur Erläuterung:

1. Wenn $sdom(y) = sdom(n)$, so gibt es keinen Umweg $x \rightarrow^* y$ mit x oberhalb $sdom(n)$. Deshalb auch keinen $x \rightarrow^* z$ mit z unterhalb y auf dem Tiefensuchpfad, denn y hat minimalen Semidominator. Also auch keinen $x \rightarrow^* z \rightarrow^* n$. Deshalb ist $sdom(n) = ddom(n)$.
2. Sonst gilt: $ddom(n)$ kann nicht unterhalb $ddom(y)$ auf Tiefensuchpfad liegen, da es ja Umweg $ddom(y) \rightarrow^* y \rightarrow^* n$ gibt. $ddom(n)$ kann nicht oberhalb liegen, da y oberhalb n liegt und dehalb muss der direkte (!) Dominator von y oberhalb des direkten Dominators von n liegen

Implementierung: ähnlich wie Semidominatoren. Das wiederholte Ablaufen der Kette $n \rightarrow y \rightarrow \ldots$ kann Faktor $O(n)$ kosten

$\Longrightarrow$ Pfadkompression (s.u.)

Berechnung der direkten Dominatoren

- Tiefensuche, dfnums
- Konstruktion des Dominanzbaums bottom-up als aufspannender Baum in inverser dfnum-Reihenfolge
- dabei Berechnung der Semidominatoren gemäß Formel
- Pfadkompression (ähnlich wie bei Union-Find) zur Berechnung der Pfade $sdom(n) \rightarrow^* n$ anwenden.
- Komplexität: $O(n \cdot \ln n)$, mit balancierter Pfadkompression: $O(n \cdot \alpha(n))$ ($\alpha =$ inverse Ackermann-Funktion; für alle praktischen Fälle ≤ 7)

Kapitel 10: Grundlagen der Datenflussanalyse und Programmabhängigkeitsgraphen

Wozu Datenflussanalyse?

Datenflussanalyse ist eine Familie von Verfahren zur

- Codeoptimierung:

```
x := 42;
... // x wird hier
... // nicht umdefiniert
IF x<0 THEN
  p(a,b,c);
```

$\Longrightarrow$

```
x := 42;
```

- Parallelisierung:

```
FOR i:= 1 TO 100 DO
  a[i] := a[i]+1;
```

$\Longrightarrow$

```
PARBEGIN
  a[1] := a[1]+1;
  ...
  a[100] := a[100]+1;
PAREND
```

- Programmverstehen: Welche Anweisungen beeinflussen den Wert von x in Zeile 4711?
- Sicherheitsprüfung: . . . siehe Vortrag . . .

Beispiel: Konstantenfaltung

Wert von Variablen zur Übersetzungszeit bekannt $\Rightarrow$
Kontrollkonstrukte statisch auswerten:

```
x := 42;
y := 17;
IF x<0 THEN
   read(y);
   x := y;
END;
IF y>0 THEN
   p(x);
```

$\Longrightarrow$

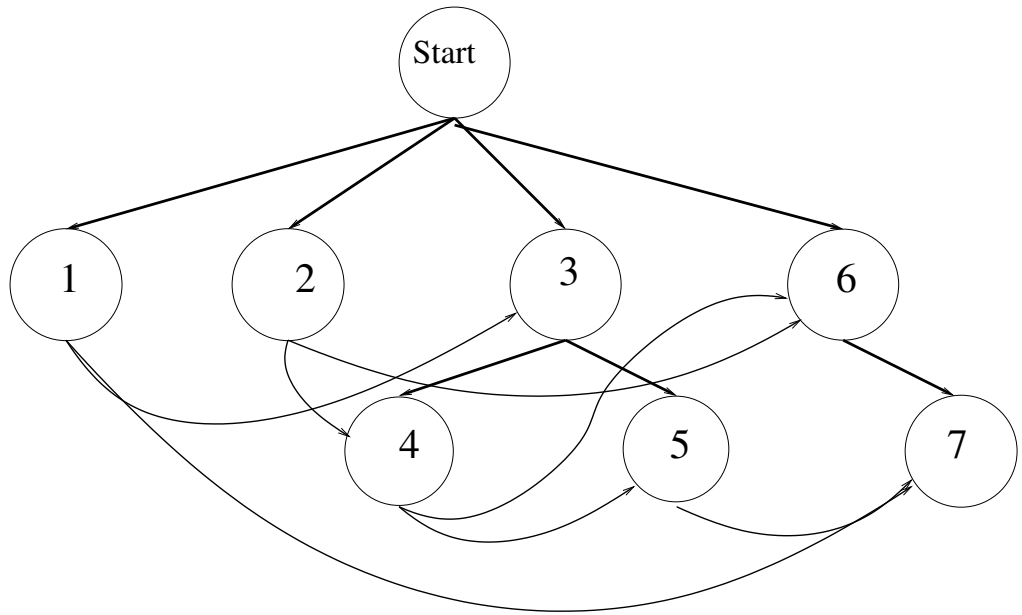
```
x := 42;
y := 17;
p(42);
```

Beispiel: Programmabhängigkeitsgraph

Kanten zwischen Definition / Verwendung einer Variablen bzw. zwischen Kontrollprädikaten / kontrollierten Anweisungen

```
(1) x := 42;
(2) y := 17;
(3) IF x<0 THEN
(4) read(y);
(5) x := y;
      END;
(6) IF y>0 THEN
(7) p(x);
```

$\Longrightarrow$

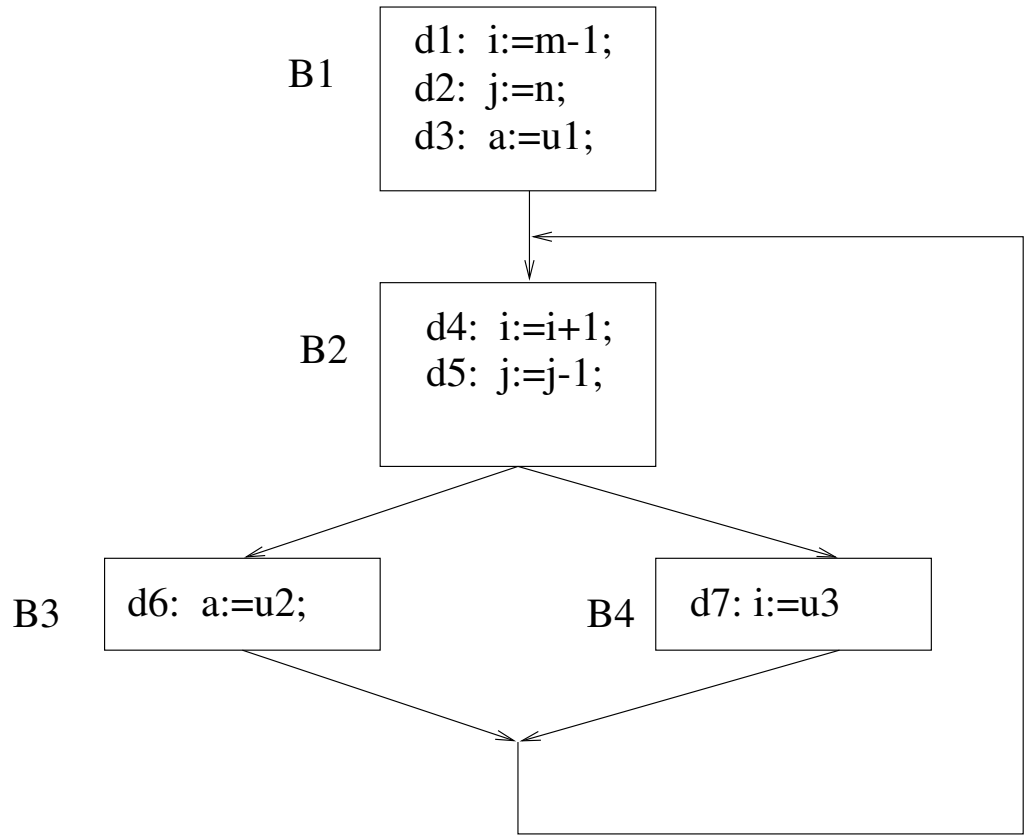


Der Kontrollflussgraph (1/2)

Ausgangspunkt der Analyse ist stets der Kontrollflussgraph[2]:

```
i := m−1;
j := n;
a := u1;
WHILE e2 DO
   i := i+1;
   j := j−1;
   IF e1 THEN
      a := u2
   ELSE
      i := u3
   END;
END;
```

$\Longrightarrow$



[2] engl.: control flow graph, CFG

Der Kontrollflussgraph (2/2)

Aufbau eines Kontrollflussgraphen:

- Knoten: Grundblöcke B_i – elementare Anweisungssequenzen ohne Kontrollflussänderung
- Kante von B_i nach B_j: B_j kann unmittelbar nach B_i ausgeführt werden
- Kann im Allgemeinen beliebige Zyklen enthalten (GOTOs!)

Datenflussanalyse orientiert sich an der Struktur des CFG.

Sichtbare Definitionen (1/3)

Ein bekanntes Datenflussproblem: Welche Zuweisung hat potentiellen Effekt auf eine gegebene Anweisung?

- Eine Variable x wird durch eine Zuweisung, Read-Anweisung, Ausgabe-Parameter usw. *definiert*.
- Definitionen (Zuweisungsstellen) $D = \{d_1, d_2, \ldots\}$
- $D_x \subseteq D$: Menge aller Definitionen von x
- Eine Definition $d \in D_x$ *killt* alle anderen Definitionen von x
- Eine Definition d *erreicht* einen Grundblock B, wenn sie nicht vorher gekillt wird. D.h. es gibt Pfad im CFG $d \rightarrow^* B$, auf dem keine weitere Definition von x liegt

Sichtbare Definitionen (2/3)

Für jeden Grundblock B werden zunächst bestimmt:

1. $gen(B)$: Menge der in B neu erzeugten Definitionen
2. $kill(B)$: Menge der in B gekillten Definitionen

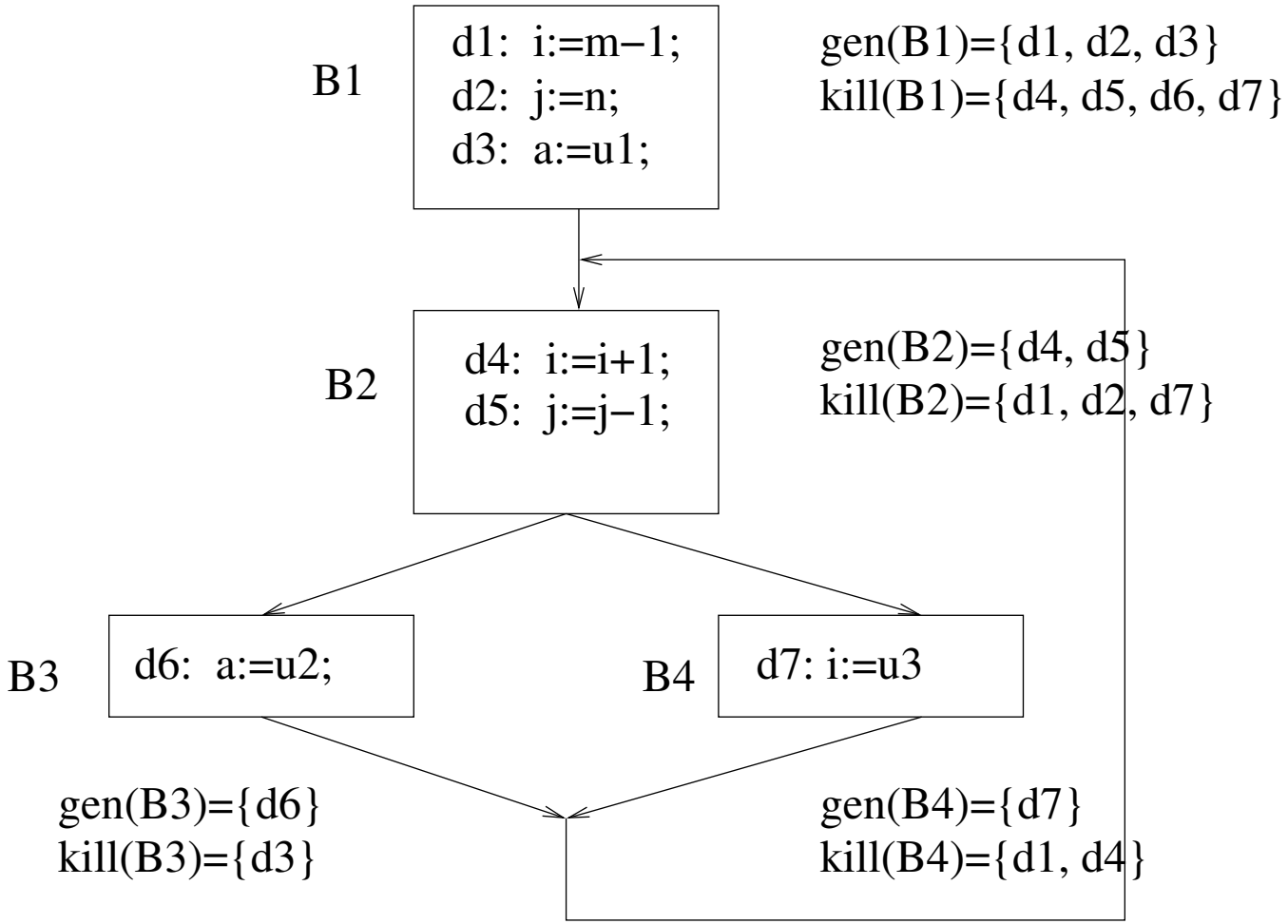
Beispiel: $B \cong$ d: x:=E; (B enthält nur eine Zuweisung)

$$\begin{aligned} gen(B) &= \{d\} \\ kill(B) &= D_x - \{d\} \end{aligned}$$

im Allgemeinen ist

$$kill(B) = \bigcup_{x \in vars(B)} D_x - (gen(B) \cap D_x)$$

Sichtbare Definitionen (3/3)



Transferfunktionen

Der Effekt jedes Grundblocks B wird durch eine *Transferfunktion* beschrieben: $f_B : 2^D \to 2^D$

$$f_B(X) = gen(B) \cup (X - kill(B))$$

„Eine Definitionsmenge X wird in B teilweise gekillt, andererseits kommen neue Definition hinzu“

Gesucht für jeden Grundblock B: $in(B), out(B) \in 2^D$ mit

$$out(B) = f_B(in(B))$$
$$in(B) = \bigcup_{C \in pred(B)} out(C)$$

Die Lösung sagt über jeden Grundblock, welche Definitionen ihn beeinflussen können.

Verband der abstrakten Werte

Die Transferfunktionen für sichtbare Definitionen sind nur ein Beispiel einer allgemeinen Methodik:

- Relevante Information wird in Form von *abstrakten Werten* berechnet
- abstrakte Werte werden für Ein- und Ausgang jedes Grundblocks berechnet
- abstrakte Werte sind stets Elemente eines Verbandes $(L; \leq, \sqcap, \sqcup)$ mit endlicher Höhe

Idee: Wenn man im Verband von oben nach unten geht, wird die durch Verbandselement repräsentierte Information immer genauer. $\top$ steht für Unwissen, $\bot$ steht für Widerspruch (zuviel Information)

Bemerkung: Leider ist die Literatur nicht einheitlich, und manche Verbände stehen in manchen Büchern auf dem Kopf. Mathematisch ist das belanglos (dualer Verband).

Beispiel 1: Verband für sichtbare Definitionen

$$L_S = (L; \leq, \sqcap, \sqcup) = (2^D; \supseteq, \cup, \cap)$$

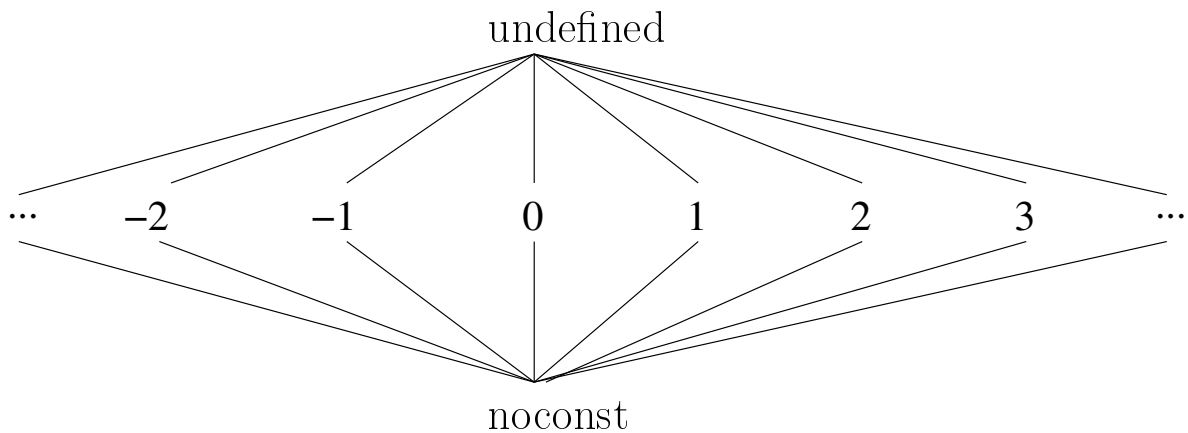
ist endlicher Potenzmengenverband.

Der Verband steht auf dem Kopf: $\emptyset = \bot$ repräsentiert Nichtwissen

Beispiel: Verband L_P für Konstantenpropagation

Es gibt folgende Fälle:

1. Man weiß nichts über den Wert einer Variable an einem Programmpunkt ($\top$ = undefined)
2. Die Variable hat an einem bestimmten Programmpunkt genau einen bekannten Wert n
3. Die Variable hat an einem Punkt 2 oder mehr verschiedene Werte ($\bot$ = noconst)



Der *flache Verband*: unendlich breit, Höhe 3.

Formal: $L_P = \mathbb{N} \cup \{\top, \bot\}, \quad x \leq y \iff x = y \vee x = \bot$

Dieser Verband steht nicht auf dem Kopf :-)

Wieso Verbände?

Gründe:

- Es gibt Supremum und Infimum (wichtig für zusammenlaufende CFG-Kanten)
- Monotone Funktionen in Verbänden endlicher Höhe haben stets Fixpunkte
- Potenzmengenverbände können effizient durch Bitvektoren implementiert werden

Transferfunktionen

Sei $(L; \leq, \sqcap, \sqcup)$ der Verband der abstrakten Werte. Zu jedem Grundblock B wird eine *Transferfunktion*

$$f_B : L \to L \in F = \{f_B \mid B \text{ Grundblock}\}$$

angegeben, die die Wirkung von B beschreibt.

Ferner wird verlangt:

1. $id_L \in F$ (Identität $\rightsquigarrow$ Nullanweisung)
2. $x \leq y \Rightarrow f_B(x) \leq f_B(y)$ (Monotonie $\rightsquigarrow$ Konvergenz)
3. $f_B \circ f_{B'} \in F$ (Abgeschlossenheit)
4. $f_B \sqcap f_{B'} \in F$ (punktweises Infimum)
5. evtl. $f_B(x \sqcap y) = f_B(x) \sqcap f_B(y)$ (Distributivität)

Distributive Analysen sind genauer.

Beispiel: Sichtbare Definitionen

$$L_S = (L; \leq, \sqcap, \sqcup) = (2^D; \supseteq, \cup, \cap)$$
$$f_B : 2^D \to 2^D, f_B(X) = gen(B) \cup (X - kill(B))$$

die f_B sind monoton und distributiv.

Beweis:

1. $X \leq Y \implies Y \subseteq X \implies f_B(Y) = gen(B) \cup (Y - kill(B)) \subseteq gen(B) \cup (X - kill(B)) = f_B(X) \implies f_B(X) \leq f_B(Y)$
2. $f_B(X \sqcap Y) = f_B(X \cup Y) = gen(B) \cup ((X \cup Y) - kill(B)) = (gen(B) \cup (X - kill(B))) \cup (gen(B) \cup (Y - kill(B))) = f_B(X) \sqcap f_B(Y)$

Aufstellen und Lösen der Datenflußgleichungen (1/2)

Gesucht: Stabile Werte $in(B), out(B) \in L$ am Anfang und Ende jedes Grundblocks B mit

$$out(B) = f_B(in(B)).$$

Dies definiert ein Gleichungssystem:

- Hintereinanderausführung von Basic Blocks entspricht Komposition der Transferfunktionen:

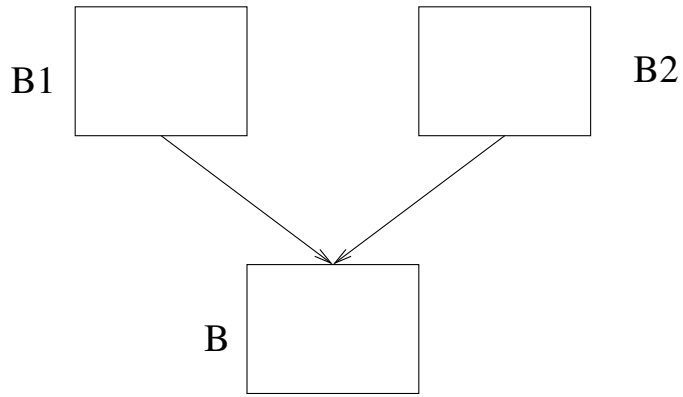
$$f_{B;B'}(D) = f_{B'}(f_B(D)) \qquad in(B') = out(B)$$

- ...

Aufstellen und Lösen der Datenflußgleichungen (2/2)

Dies definiert ein Gleichungssystem:

- ...
- falls zwei Pfade im CFG zusammenlaufen, muss das Infimum ihrer Transferfunktionen berechnet werden:

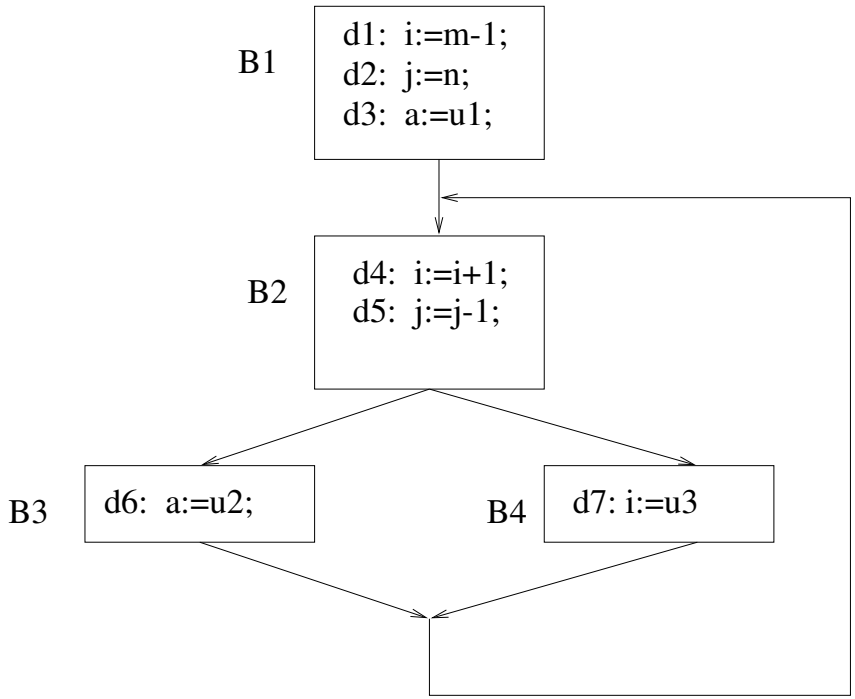


$$f_{(B1||B2)}(X) = (f_{B1} \sqcap f_{B2})(X) = f_{B1}(X) \sqcap f_{B2}(X)$$

Im speziellen Fall „sichtbare Definitionen“ müssen die entsprechenden Definitionsmengen $\in 2^D$ *vereinigt* werden:

$$in(B) = out(B1) \cup out(B2) = f_{B1}(in(B1)) \cup f_{B2}(in(B2))$$

Beispiel: Sichtbare Definitionen (1/2)



Gleichungssystem für alle Werte: $in(B_i), out(B_i)$

$$
\begin{aligned}
in(B1) &= \emptyset \\
in(B2) &= out(B1) \cup out(B3) \cup out(B4) \\
&= f_{B1}(in(B1)) \cup f_{B3}(in(B3)) \cup f_{B4}(in(B4)) \\
in(B3) &= in(B4) = out(B2) = f_{B2}(in(B2))
\end{aligned}
$$

Beispiel: Sichtbare Definitionen (2/2)

Initialisierung: $in(B_i) = \emptyset$

Mengen werden als Bitvektoren dargestellt.

Beispieleinträge:

$$\begin{aligned} in_1(B2) &= out_0(B1) \cup out_0(B3) \cup out_0(B4) \\ &= 1110000 \vee 0000010 \vee 0000001 = 1110011 \\ out_1(B2) &= gen(B2) \cup (in_1(B2) - kill(B2)) \\ &= 0001100 \vee (1110011 \wedge \overline{1100001}) = 0011110 \end{aligned}$$

Fixpunktiteration:

Block B	$in_0(B)$	$out_0(B)$	$in_1(B)$	$out_1(B)$	$in_2(B)$	$out_2(B)$
$B1$	0000000	1110000	0000000	1110000	0000000	1110000
$B2$	0000000	0001100	1110011	0011110	1111111	0011110
$B3$	0000000	0000010	0001100	0001110	0011110	0001110
$B4$	0000000	0000001	0001100	0010111	0011110	0010111

$\Rightarrow$ am Anfang von B4 sind $d3, d4, d5, d6$ sichtbar, insbesondere die beiden Zuweisungen an a!

Beispiel: Konstantenpropagation (1/3)

Jeder B_i ist aus elementaren Zuweisungen zusammengesetzt. Es ist

$$L = \{(x, c) \mid x \in \mathit{Vars}(P), c \in L_C\}$$

Transferfunktion für Zuweisung `x := E`: $f_B : 2^L \to 2^L$

$$Z = \{(\mathtt{v1}, c_1), (\mathtt{v2}, c_2), \ldots\}$$

$$\mathtt{x} := \mathtt{E}$$

$$f_B(Z) = \mathit{gen}(B) \cup (Z \setminus \mathit{kill}(B)) = \{(\mathtt{x}, \mathit{eval}(\mathtt{E}))\} \cup (Z \setminus \{(\mathtt{x}, c_x)\})$$

mit

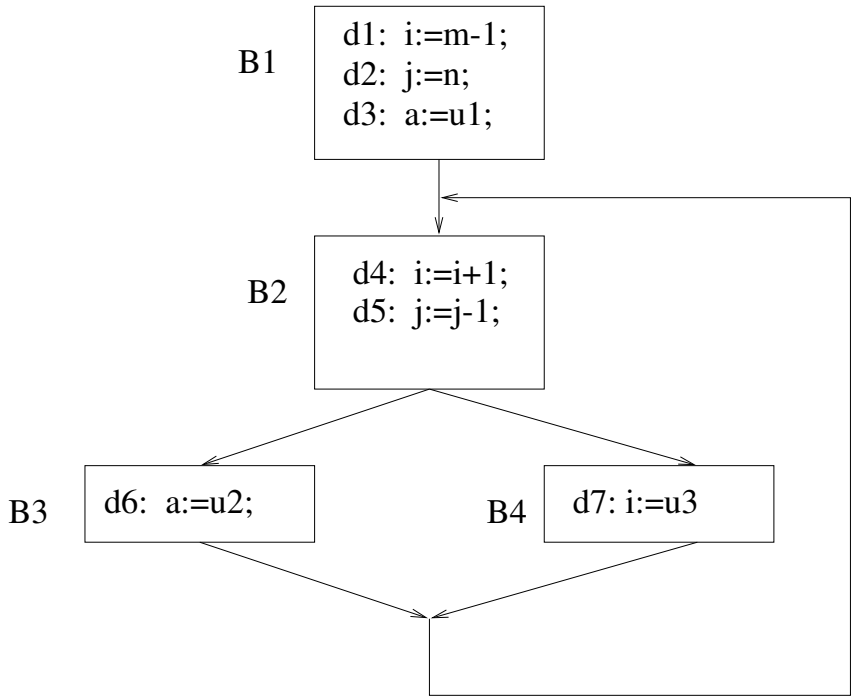
$$\mathit{eval}(E) = \begin{cases} \mathit{value}(E) & \forall v \in \mathit{Vars}(E) : (v, c_v) \in Z \\ & \wedge c_v \notin \{\top, \bot\} \\ \top & \exists v \in \mathit{Vars}(E) : (v, \top) \in Z \\ \bot & \exists v \in \mathit{Vars}(E) : (v, \bot) \in Z \end{cases}$$

wobei $\top = \mathit{undefined}$, $\bot = \mathit{noconst}$.

$\bot$ bedeutet: die Variable ist garantiert keine Konstante.

f_B ist monoton, aber nicht distributiv! (Übung)

Beispiel: Konstantenpropagation (2/3)



Betrachtet man $m, n, u1, u2, u3$ als Konstanten, i, j, a als Variablen, so ergibt sich das Gleichungssystem

$$\begin{aligned} in(B1) &= \{(i, \top), (j, \top), (a, \top)\} \\ in(B2) &= out(B1) \sqcap out(B2) \sqcap out(B3) \\ &= f_{B1}(in(B1)) \sqcap f_{B3}(in(B3)) \sqcap f_{B4}(in(B4)) \\ in(B3) = in(B4) &= out(B2) = f_{B2}(in(B2)) \end{aligned}$$

Beispiel: Konstantenpropagation (3/3)

Das Infimum ist definiert durch

$$\begin{aligned} X \sqcap Y &= \{(v,r) \mid (v,a) \in X, (v,b) \in Y, r = a \sqcap b \in L_C\} \\ &\quad \cup \{(v,a) \in X \mid (v,b) \notin Y\} \cup \{(v,b) \in Y \mid (v,a) \notin X\} \end{aligned}$$

Bsp: $\{(i,42),(j,\top)\} \sqcap \{(j,17),(i,13),(a,1)\} = \{(i,\bot),(j,17),(a,1)\}$

Fixpunktiteration (Übung!) ergibt u.a.:

$$\begin{aligned} in(B2) &= \{(i,(m-1) \sqcap (m-1+1) \sqcap u3),(j,n \sqcap (n+1)),(a,u1 \sqcap u2)\} \\ &= \{(i,\bot),(j,\bot),(a,\bot)\} \end{aligned}$$

$\Longrightarrow$ Am Anfang von $B2$ sind die Variablen i, j, a garantiert keine Konstanten, sondern haben zur Laufzeit mindestens zwei verschiedene Werte

Falls IF/WHILE-Ausdrücke konstant werden, so verschwinden Pfade, d.h. *das Gleichungssystem ändert sich*! Man muss dann die Iteration mit dem neuen System wiederholen.

Korrektheit der Fixpunktiteration

Satz. (Kam/Ullman 77) Seien $P_1, P_2, \ldots$ alle Pfade von **Start** zu Programmpunkt s; seien die Transferfunktionen $f_B \in F$ distributiv; sei $fix(s) \in L$ der durch Fixpunktiteration für $out(s)$ berechnete Wert. Dann ist

$$fix(s) = \bigsqcap_i f_{P_i}(\top).$$

Bemerkungen:

- Dies ist deswegen bemerkenswert, weil es im Allgemeinen unendlich viele Pfade gibt.
- Falls f nicht distributiv, gilt nur $\geq$ statt $=$, d.h. der Fixpunkt stellt eine konservative Approximation dar.
- Für manche nicht-distributive Probleme ist die exakte Lösung nicht berechenbar z.B. Konstantenpropagation
- Es gibt Verallgemeinerungen des Satzes für interprozedurale Analyse (Knoop et al. 95)

Programmabhängigkeitsgraph (1/2)

Seien x, y Anweisungen bzw. Grundblöcke; $gen(x), in(x)$ wie vorher, $var(d)$ sei die in einer Definition definierte Variable; $uses(x)$ seien die in x benutzten Variablen.

Definition *(Datenabhängigkeit)*

$$x \to y \iff \exists d \in gen(x) : d \in in(y) \wedge var(d) \in uses(y)$$

Definition *(Dominanz)*
x dominiert y, wenn jeder CFG-Pfad von *Start* zu y über x führt.

Definition *(Postdominanz)*
y postdominiert x, wenn jeder CFG-Pfad von x zu *Stop* über y führt.

Programmabhängigkeitsgraph (2/2)

Definition *(Kontrollabhängigkeit)*
$x \to y \iff \exists$ Pfad P von x nach y im CFG $\forall z \in P, z \neq x, z \neq y$: y postdominiert $z \ \wedge \ y$ postdominiert nicht x

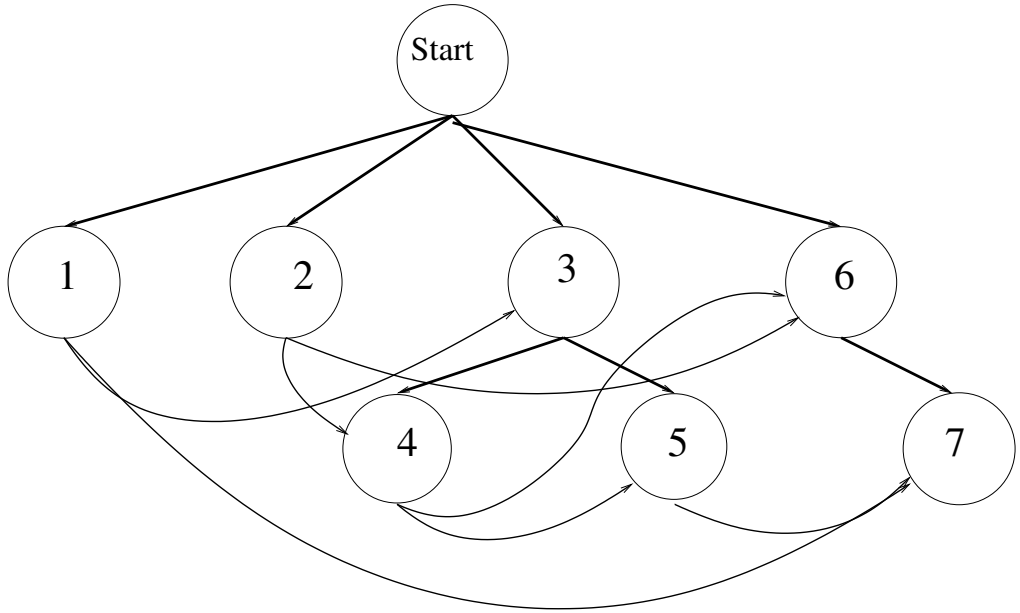
Bemerkung: In strukturierten Programmen sind kontrollabhängige Anweisungen Y gerade jene im Rumpf eines `if`, `while` usw; sie hängen von der regierenden Bedingung x ab

Definition Der PDG $(S, \to)$ hat alle Anweisungen als Knoten und Daten-/Kontrollabhängigkeiten als Kanten.

Programmabhängigkeitsgraph – Beispiel

```
(1) x := 42;
(2) y := 17;
(3) IF x<0 THEN
(4) read(y);
(5) x := y;
      END;
(6) IF y>0 THEN
(7) p(x);
```

$\Longrightarrow$



Bemerkung: Falls es Prozeduren, Adressen (Aliasing) oder komplexe Datenstrukturen gibt, wird der PDG wesentlich komplizierter

Beispiel: Programm für elektronische Waage ($\rightsquigarrow$ PTB)

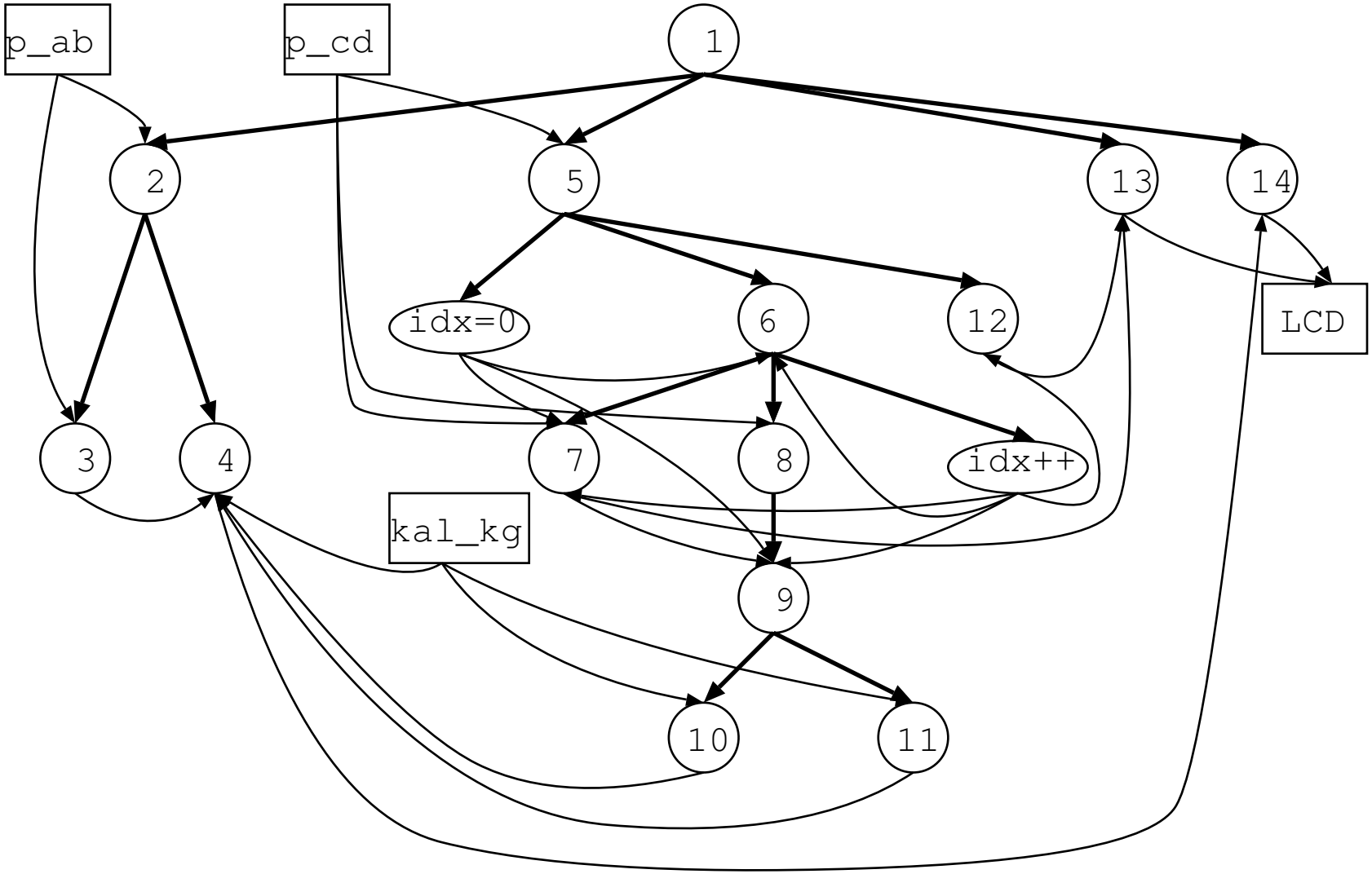
```
    void main() {
        int p_ab[2] = {0, 1};
        int p_cd[1] = {0};
        char e_puf[8];
        int u; int idx;
        float u_kg; float kal_kg = 1.0;

(1)     while(TRUE) {
(2)         if ((p_ab[CTRL2] & 0x10)==0) {
(3)             u = ((p_ab[PB] & 0x0f) << 8) + (unsigned int)p_ab[PA];
(4)             u_kg = u * kal_kg;
            }
(5)         if ((p_cd[CTRL2] & 0x01) != 0) {
(6)             for (idx=0;idx<7;idx++) {
(7)                 e_puf[idx] = p_cd[PA];
(8)                 if ((p_cd[CTRL2] & 0x10) != 0) {
(9)                     switch(e_puf[idx]) {
(10)                        case '+': kal_kg *= 1.01; break;  /* unerlaubter */
(11)                        case '-': kal_kg *= 0.99; break;  /* Datenfluss  */
                        }
                    }
                }
(12)            e_puf[idx] = '\0';
            }
(13)        printf ("Artikel : %07.7s\n,e_puf);
(14)        printf ("     %6.2f kg     ", u_kg);
        }
    }
```

p_cd: Keyboard-Eingaberegister; Kontrollbits

p_ab: Messwert-Eingaberegister

Beispiel: Programm für elektronische Waage – PDG



fett: Kontrollabhängigkeiten

dünn: Datenabhängigkeiten

Program Slicing (1/2)

Welche Anweisungen y können Programmpunkt x beeinflussen, und welche tun dies garantiert nicht?

Definition: Der Rückwärtsslice $BS(x)$ enthält alle Anweisungen, die x beeinflussen können.

Im PDG ist

$$BS(x) = \{y \mid y \rightarrow^* x\}$$

Bemerkung: $BS(x)$ darf zu groß sein, aber niemals zu klein (Prinzip der konservativen Approximation). In der Praxis will man natürlich möglichst kleine $BS(x)$

Beispiel: Für die elektronische Waage ist $(5) \in BS((14))$. Deshalb potentielle Beeinflussung des angezeigten Messwertes durch das Keyboard.

Program Slicing (2/2)

Definition: Vorwärtsslice

$$FS(x) = \{y \mid x \rightarrow^* y\}$$

Definition: Chop

$$CH(x, y) = \{z \mid x \rightarrow^* z \rightarrow^* y\} = BS(y) \cap FS(x)$$

Bemerkung: Diese einfachen Formeln gelten nicht mehr im interprozeduralen Fall sowie für komplexe Datenstrukturen.

Slicing Theorem: Wenn es keinen Pfad $x \rightarrow^* y$ im PDG gibt, so kann x garantiert nicht y beeinflussen.

Ausblick

Datenflussanalyse ist ein weites Feld, in das enorme Forschungsanstrengungen geflossen sind und fließen.

Erwähnt seien

- interprozedurale Analyse
- Analyse von Arrays
- Analyse von Pointern
- effiziente Implementierungstechniken
- inkrementelle / bedarfsgetriebene Datenflussanalyse
- automatische Parallelisierung
- Program Slicing
- Anwendungen in Wartung, Reengineering, Sicherheitsanalyse, ...

Datenflussanalyse = abstrakte Interpretation + Model Checking
(B. Steffen)

Literatur: Nielson/Nielson/Hankin: Principles of Program Analysis. Springer 1999