Datenflussanalyse und Codeoptimierungen

Kapitel 9, Compilers

Michael Leuschel

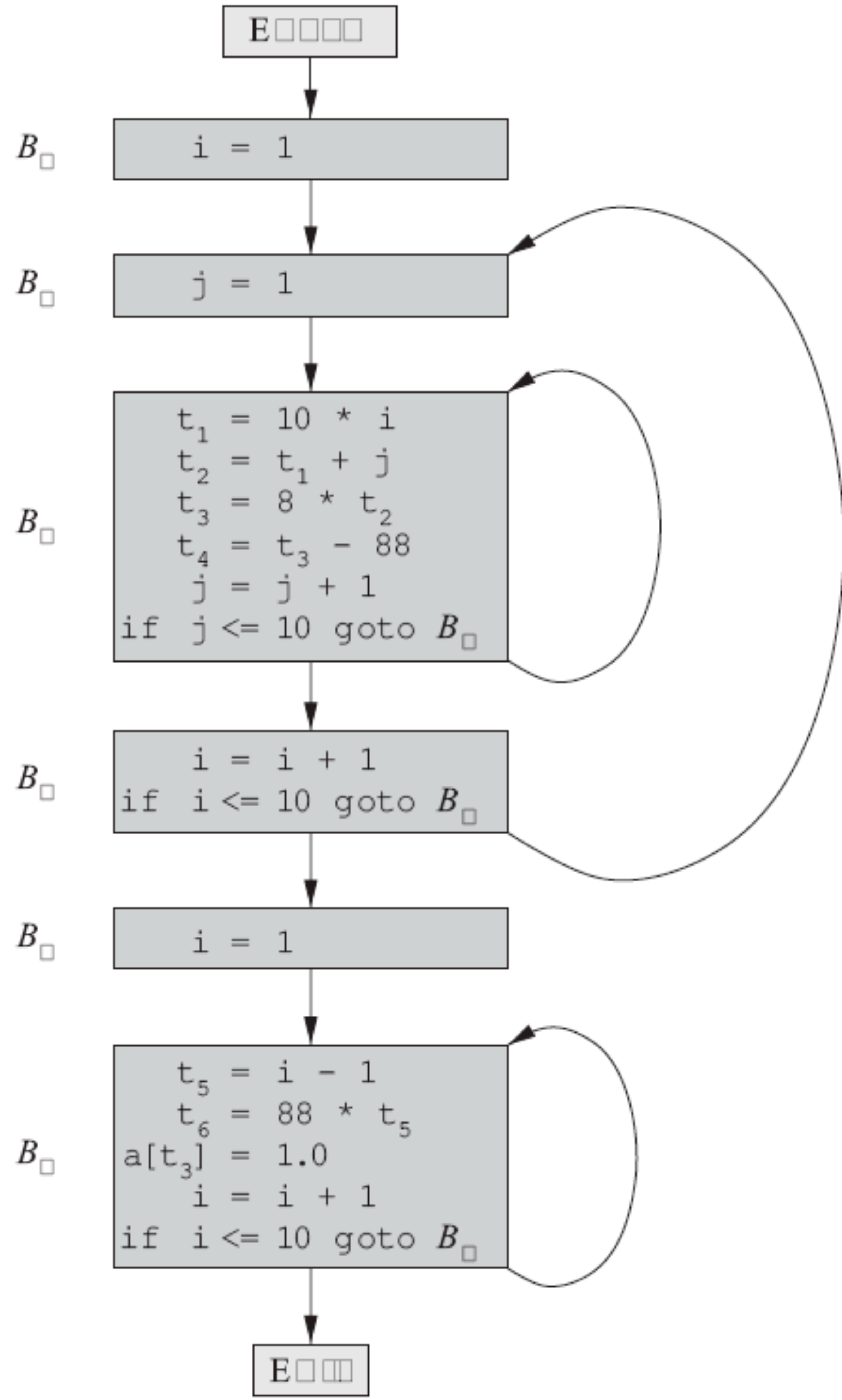
Grundblöcke (Kap. 8.4)

- Maximale konsekutive Sequenz von Drei-Adress Anweisungen so dass:
  - Kontrollfluss kann nur durch den ersten Befehl in den Block gelangen
  - Steuerung verlässt den Block ohne Halt/Verzweigung mit Ausnahme des letzten Befehls
- Werden in Flussgraphen zusammengefasst

Beispiel

```
for i from 1 to 10 do
   for j from 1 to 10 do
      a[i, j] = 0.0;
for i from 1 to 10 do
   a[i, i] = 1.0;
```

```
1) i = 1
2) j = 1
3) t1 = 10 * i
4) t2 = t1 + j
5) t3 = 8 * t2
6) t4 = t3 - 88
7) a[t4] = 0.0
8) j = j + 1
9) if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13
```



Datenflussanalysen

- lokale Codeoptimierungen
  - Verbesserung innerhalb eines Grundblocks
- globale Codeoptimierungen
  - betrachten das gesamte Programm
  - beruhen oft auf Datenflussanalysen:
    - sammeln Informationen über ein Programm
    - Für jeden Programmbefehl: Eigenschaften die bei allen Ausführungen beibehalten werden

Beispiel: Konstantenpropagation

- Für jede Variable und jeden Programmpunkt:
  - hat die Variable einen eindeutigen konstanten Wert an diesem Punkt
- Darauf basierende Optimierung:
  - Variablenreferenzen durch Konstanten ersetzen

Beispiel: Liveness

- Für jede Variable und jeden Programmpunkt:
  - wird der Wert der Variable verwendet bevor er überschrieben wird (sonst tot)
- Darauf basierende Optimierung:
  - Tote Variablen brauchen nicht in Registern oder im Speicher festgehalten werden

Beispiel: Gemeinsame Ausdrücke

- E ist common subexpression:
  - E vorher berechnet
  - Werte der Variablen in E nicht verändert
- Datenflussanalyse?

$$
\begin{array}{rcl}
t_6 & = & 4 * i \\
x & = & a[t_6] \\
t_7 & = & 4 * i \\
t_8 & = & 4 * j \\
t_9 & = & a[t_8] \\
a[t_7] & = & t_9 \\
t_{10} & = & 4 * j \\
a[t_{10}] & = & x \\
\multicolumn{3}{l}{\texttt{goto } B_2}
\end{array}
$$

(lokal)

$$
\begin{array}{rcl}
t_6 & = & 4 * i \\
x & = & a[t_6] \\
t_8 & = & 4 * j \\
t_9 & = & a[t_8] \\
a[t_6] & = & t_9 \\
a[t_8] & = & x \\
\multicolumn{3}{l}{\texttt{goto } B_2}
\end{array}
$$

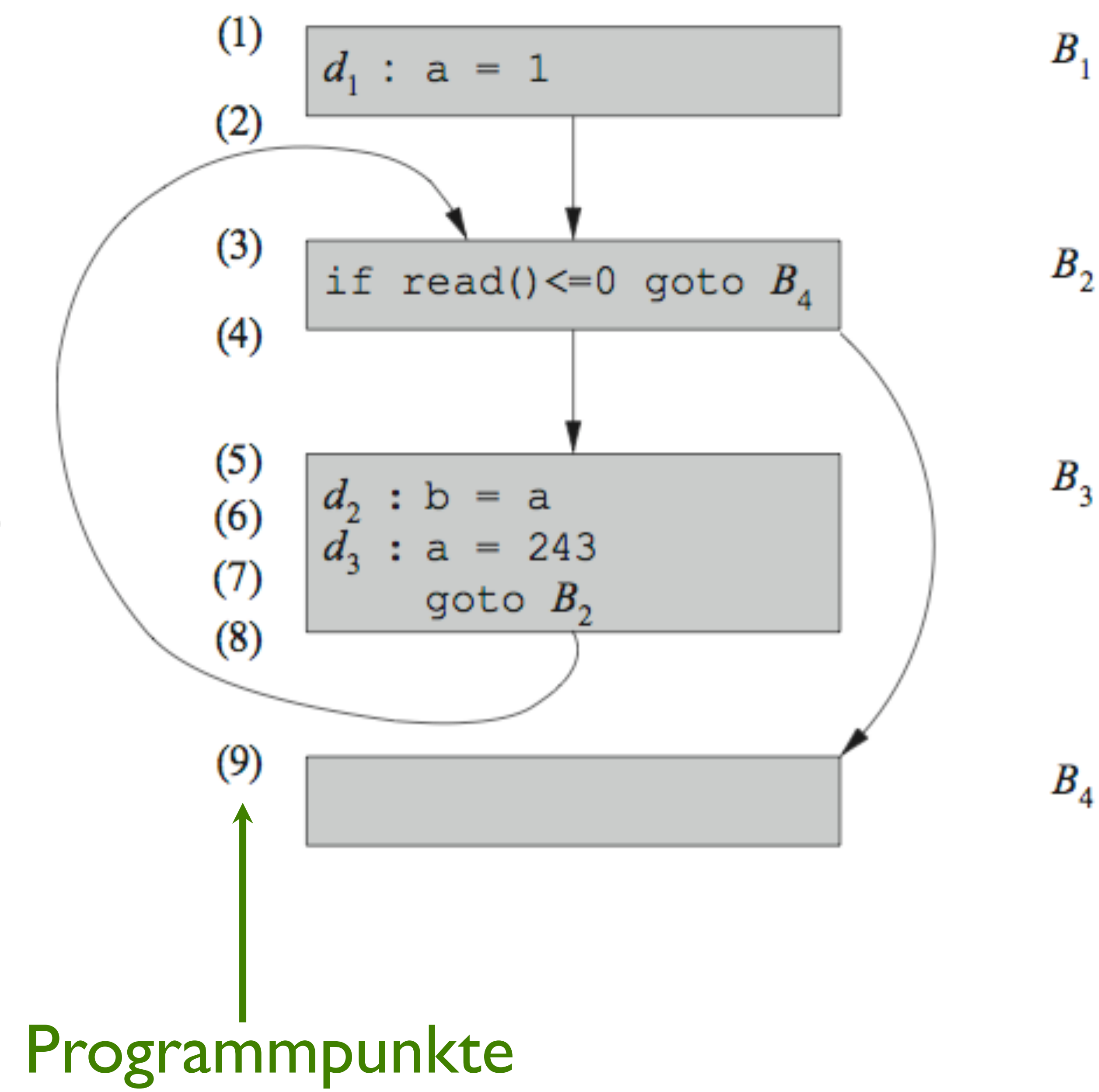
Einführung in die Datenflussanalyse

Ausführungspfade

- Ausführung/Interpretation eines Programms:
  - Reihe von Transformationen des Zustandes:
    - $s_i \longrightarrow_{\text{Anweisung } i} s_{i+1} \longrightarrow_{\text{Anw. } i+1} \ldots$
  - Ausführungspfad: Sequenz der Programmpunkte (Punkte zwischen den Anweisungen)

Beispiel: Programmpunkte und Ausführungspfade

- Pfade:
- (1,2,3,4,9)
- (1,2,3,4,5,6,7,8,3,4,9)
- ...



Datenflussabstraktion

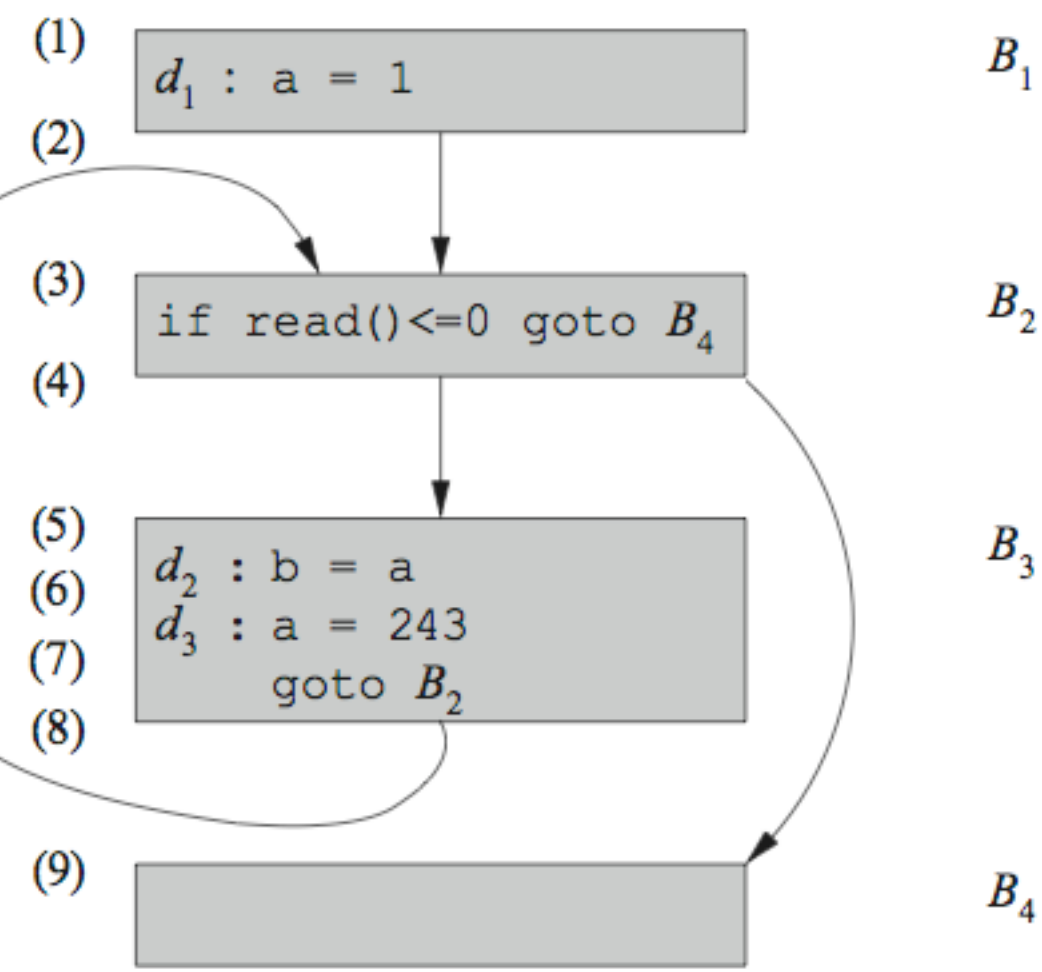
Anmerkung: nicht alle Pfade unbedingt konkret ausführbar!

- Analyse:
  - Berücksichtigung aller Pfade durch Flussgraphen
  - Extraktion/Zusammenfassung der Information pro Programmpunkt:
    - Keine Unterscheidung wie (über welchen Pfad) ein Programmpunkt errreicht wurde
    - Keine Arbeit mit vollständigen Zuständen:
      - Abstraktion von Einzelheiten

Abstraktionsbeispiele

Datenflusswerte

- PP (5):
  - a ∈ {1,243}
  - def(a) ∈ {d1,d3} (Erreichende Definitionen)
  - cst(a) = NAC (Konstante Propagation)



Datenflussanalyseschema

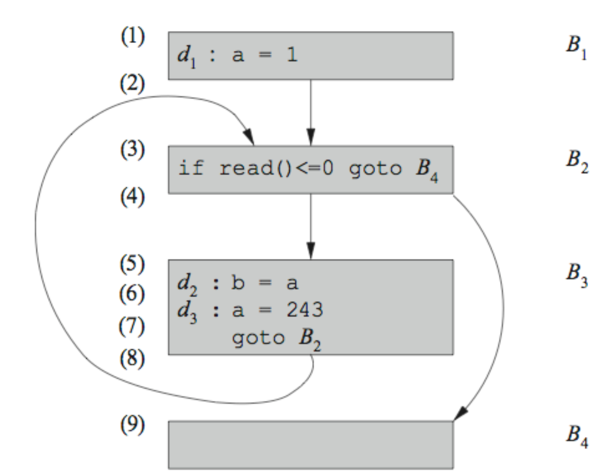
- Bereich: Menge der möglichen Datenflusswerte
- Für jede Anweisung s:
  - IN[s]: Datenflusswert vor s
  - OUT[s]: Datenflusswert nach s
- Einschränkungen (Constraints)
  - Anweisungen (→Transferfunktion)
  - Flusssteuerung

Constraints durch Transferfunktionen

- Vorwärtsfluss:
  - $OUT[s] = f_s(IN[s])$
- Rückwartsfluss:
  - $IN[s] = f_s(OUT[s])$

Constraints durch Kontrollfluss

- Innerhalb eines Grundblocks:
  - $IN[s_{i+1}] = OUT[s_i]$
- Was tun zwischen den Blöcken?



Analyse für Grundblöcke

- Zusammenfassung der Transferfunktion:
  - $f_B = f_{sn} \circ \ldots \circ f_{s2} \circ f_{s1}$
- Vorwärtsfluss
  - $OUT[B] = f_B(IN[B])$
  - $IN[B] = \bigcup_{P \text{ Vorgänger von } B} OUT[P]$

Hängt von der Analyse ab

Analyse für Grundblöcke II

- Vorwärtsfluss
  - $OUT[B] = f_B(IN[B])$
  - $IN[B] = \cup_{P \text{ Vorgänger von } B} OUT[P]$
- Rückwärtsfluss
  - $IN[B] = f_B(OUT[B])$
  - $OUT[B] = \cup_{P \text{ Nachfolger von } B} IN[P]$

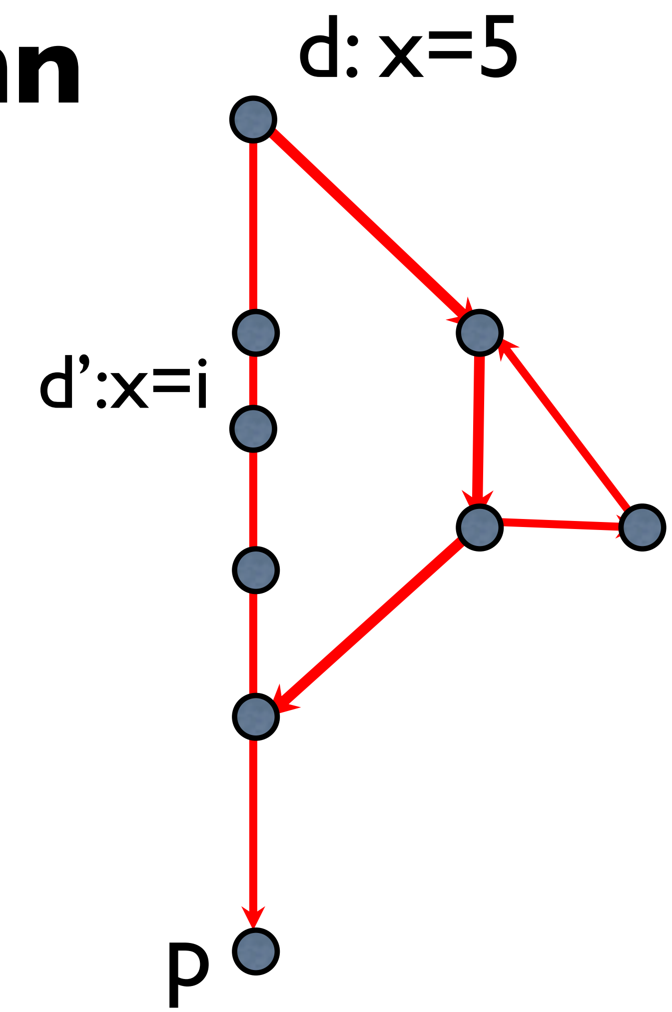
Lösungen

- Generell gibt es mehrere Lösungen für die Datenflussgleichungen
- Wir suchen die präziseste Lösung

Beispiel: Erreichende Definitionen

Erreichende Def. (Reaching Definitions)

- Definition von x: Anweisung die x Wert zuweist oder zuweisen **kann**
- Definition d erreicht Programmpunkt p wenn:
  - ein Pfad vom Programmpunkt direkt nach d hinzu p existiert, sodass d nicht auf dem Pfad zerstört (kill) wird (von einer anderen Definition)



Konservative Analyse I

- Nicht alle Datenflusspfade ausführbar

```
if (a == b) Anweisung 1; else if (a == b) Anweisung 2;
```

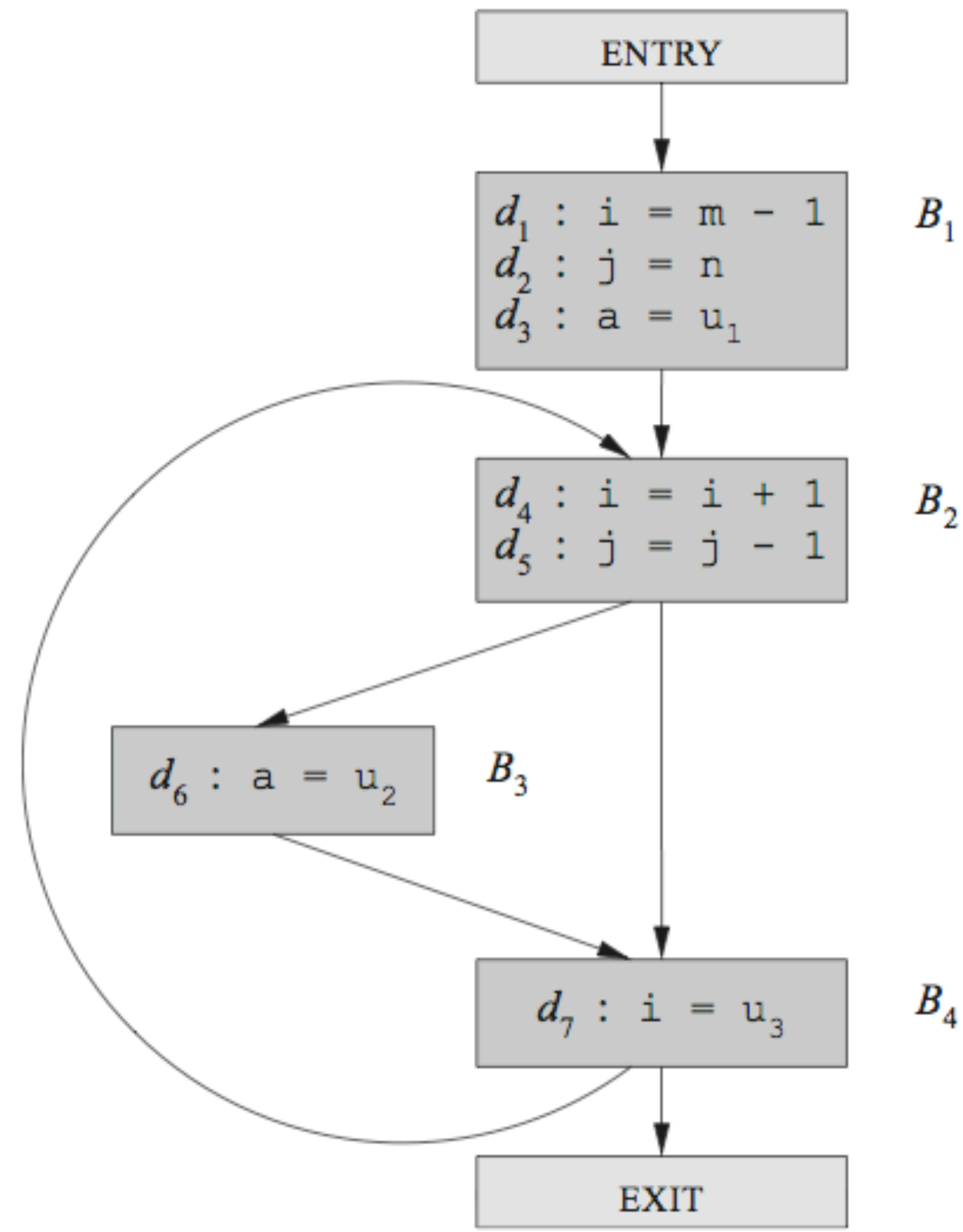
- Definition von x: Anweisung die x Wert zuweist oder zuweisen **kann**
  - x = 1 → weist x sicher einen Wert zu
  - y = 2 → kann theoretisch auch x zuweisen. Warum?
  - a[i] = 2 → Def. von a[j] ?
- Annahme für Folien/Kapitel 9: keine Aliase

Anwendung

- Fehlerhafte (mögliche) Verwendung vor Definition
  - Scheindefinition am Eingang des Flussgraphen jede Variable x
  - Wenn Scheindefinition eine Verwendung von x erreicht:
    - Ausgabe eine Warnung

Beispiel

- Was erreicht Anfang von B2:
  - Alle Def. von B1
  - d5, d6, d7
  - aber **nicht** d4



Transfergleichungen

- Für eine Definition d:
- $gen_d = \{d\}$: Menge der definierten Variablen
- $kill_d$: Menge der zerstörten Definitionen
- $f_d(x) = gen_d \cup (x - kill_d)$

Zusammenfassung

- $f_1(x) = gen_1 \cup (x - kill_1)$ und $f_2(x) = gen_2 \cup (x - kill_2)$
- : $$\begin{aligned} f_2\,(f_1(x)) &= gen_2 \cup (gen_1 \cup (x - kill_1) - kill_2) \\ &= (gen_2 \cup (gen_1 - kill_2)) \cup (x - (kill_1 \cup kill_2)) \end{aligned}$$
- Blöcke

$$f_B(x)) = gen_B \cup (x - kill_B)$$

wobei

$$kill_B = kill_1 \cup kill_2 \cup \ldots \cup kill_n$$

und

$$\begin{aligned} gen_B = gen_n &\cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \\ &\ldots \cup (gen_1 - kill_2 - kill_3 \ldots - kill_n) \end{aligned}$$

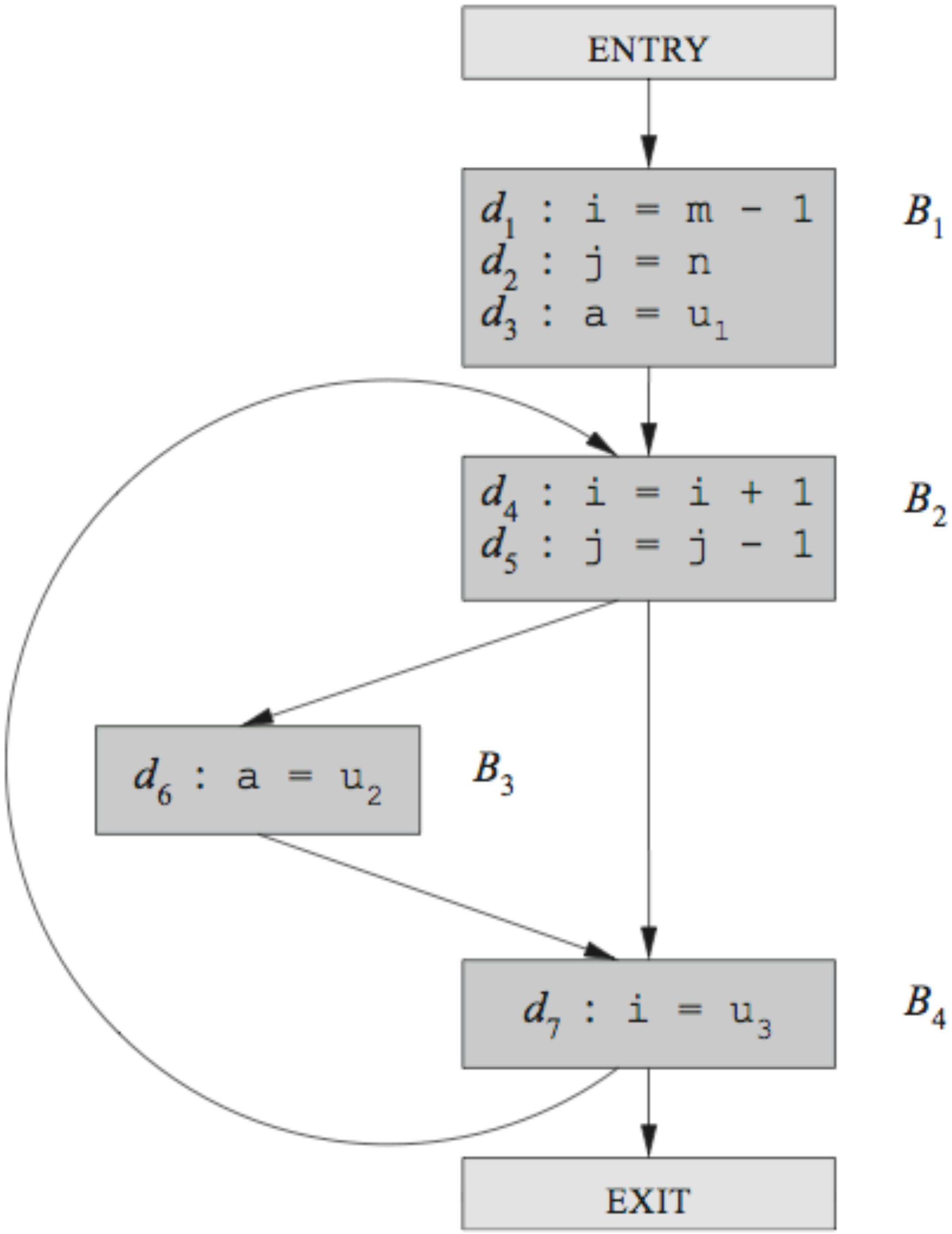
Beispiel

d_1: a = 3

d_2: a = 4

- gen_B = {d2}
- $kill_B$ = {d1,d2}

Beispiel II



$gen_{B_1} = \{ d_1, d_2, d_3 \}$

$kill_{B_1} = \{ d_4, d_5, d_6, d_7 \}$

$gen_{B_2} = \{ d_4, d_5 \}$

$kill_{B_2} = \{ d_1, d_2, d_7 \}$

$gen_{B_3} = \{ d_6 \}$

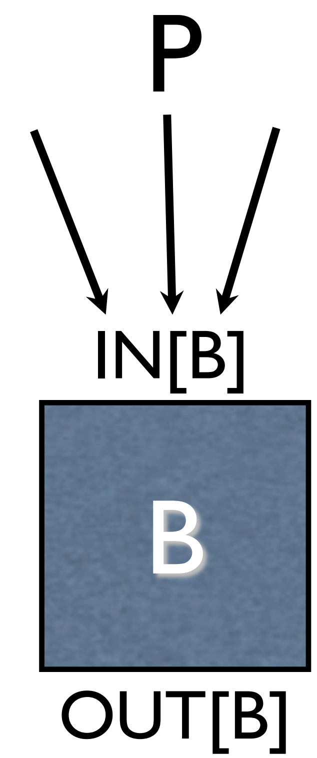
$kill_{B_3} = \{ d_3 \}$

$gen_{B_4} = \{ d_7 \}$

$kill_{B_4} = \{ d_1, d_4 \}$

ENTRY

Gleichungen



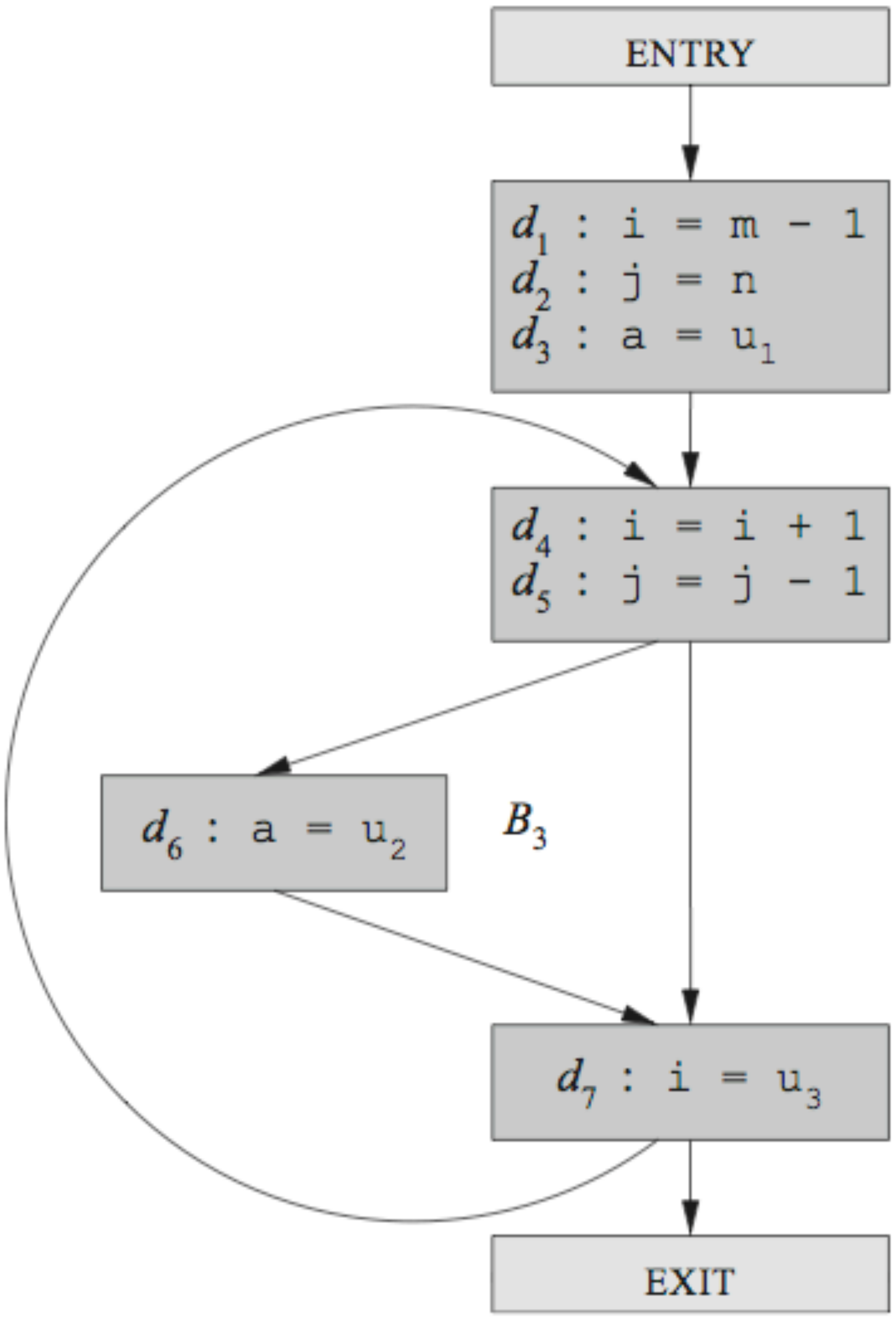
$$IN[B] = \bigcup_{P \text{ ist ein Vorgänger von } B} OUT[P]$$

- Mengenvereinigung: Durchschnittsoperator (meet)
- $OUT[B] = gen_B \cup (IN[B] - kill_B)$
- Annahme: zwei leere Grundblöcke ENTRY und EXIT
- $OUT[ENTRY] = \emptyset$

Iterativer Algorithmus

```
OUT[ENTRY] = ∅;
for (jeder Grundblock B außer ENTRY) OUT[B] = ∅;
while (ändert sich in ein beliebiges Vorkommen von OUT)
    for (jeder Grundblock B außer ENTRY) {
        IN[B] = ⋃_{P ein Vorgänger von B} OUT[P];
        OUT[B] = gen_B ∪ (IN[B] - kill_B);
    }
```

Beispiel II



B_1 $\quad gen_{B_1} = \{ d_1, d_2, d_3 \}$

$\quad kill_{B_1} = \{ d_4, d_5, d_6, d_7 \}$

B_2 $\quad gen_{B_2} = \{ d_4, d_5 \}$

$\quad kill_{B_2} = \{ d_1, d_2, d_7 \}$

$\quad gen_{B_3} = \{ d_6 \}$

$\quad kill_{B_3} = \{ d_3 \}$

B_4 $\quad gen_{B_4} = \{ d_7 \}$

$\quad kill_{B_4} = \{ d_1, d_4 \}$

```
OUT[ENTRY] = Ø;
for (jeder Grundblock B außer ENTRY) OUT[B] = Ø;
while (ändert sich in ein beliebiges Vorkommen von OUT)
    for (jeder Grundblock B außer ENTRY) {
        IN[B] = ⋃_{P ein Vorgänger von B} OUT[P];
        OUT[B] = gen_B ∪ (IN[B] - kill_B);
    }
```

Bitvektordarstellung

$d_1d_2d_3$ $d_4 \ldots d_7$

Block *B*	OUT[*B*]^0	IN[*B*]^1	OUT[*B*]^1	IN[*B*]^2	OUT[*B*]^2
B_1	000 0000	000 0000	111 0000	000 0000	111 0000
B_2	000 0000	111 0000	001 1100	111 0111	001 1110
B_3	000 0000	001 1100	000 1110	001 1110	000 1110
B_4	000 0000	001 1110	001 0111	001 1110	001 0111
EXIT	000 0000	001 0111	001 0111	001 0111	001 0111

Andere Analysen

	Erreichende Definitionen	Lebendige Variablen	Verfügbare Ausdrücke
Bereich	Mengen von Definitionen	Mengen von Variablen	Mengen von Ausdrücken
Richtung	Vorwärts	Rückwärts	Vorwärts
Transferfunktion	$gen_B \cup (x - kill_B)$	$use_B \cup (x - def_B)$	$e\_gen_B \cup (x - e\_kill_B)$
Grenze	OUT[ENTRY] $= \emptyset$	IN[EXIT] $= \emptyset$	OUT(ENTRY] $= \emptyset$
Durchschnitts-operator ($\wedge$)	$\cup$	$\cup$	$\cap$
Gleichungen	OUT[B] = f_B(IN[B]) IN[B] = $\bigwedge_{P \text{ ist Vorgänger von } B}$ OUT[P]	IN[B] = f_B(OUT[B]) OUT[B] = $\bigwedge_{S \text{ ist Nachfolger von } B}$ IN[S]	OUT[B] = f_B(IN[B]) IN[B] = $\bigwedge_{P \text{ ist Vorgänger von } B}$ OUT[P]
Initialisieren	OUT[B] $= \emptyset$	IN[B] $= \emptyset$	OUT[B] = U

Analyse lebendiger Variablen

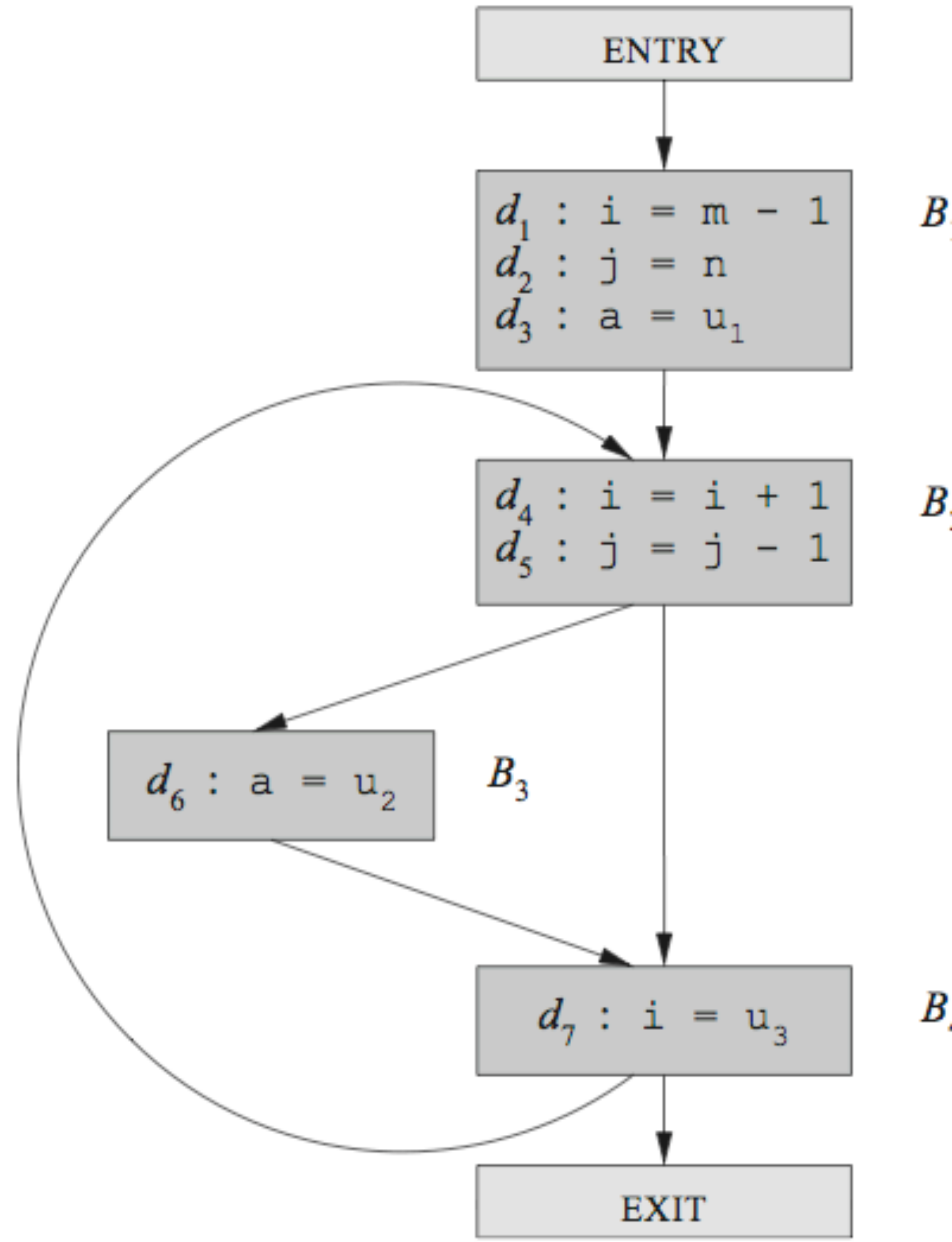
- Variable x and Programmpunkt p:
  - wird der Wert von x bei p entlang eines Pfades (bei p beginnend) verwendet?
    - Wenn ja: x an p lebendig; sonst tot
- Verwendungszwecke: Registervergabe

Def/Use für Blöcke

- def_B = Menge der Variable die in B definitiv definiert werden, bevor sie in B verwendet werden
- use_B: Menge der Variablen deren Werte in B vor einer Definition in B verwendet werden
- Variable in use_B muss lebendig am Eingang von B sein; Variable in def_B muss am Eingang tot sein

Beispiel

- In B2:
  - $use_{B2} = \{i,j\}$
  - $def_{B2} = \{i,j\}$



Gleichungen

- IN[EXIT] = ∅
- Für alle B außer EXIT:

$$\mathrm{IN}[B] = use_{\boldsymbol{B}} \cup (\mathrm{OUT}[B] - def_{\boldsymbol{B}})$$

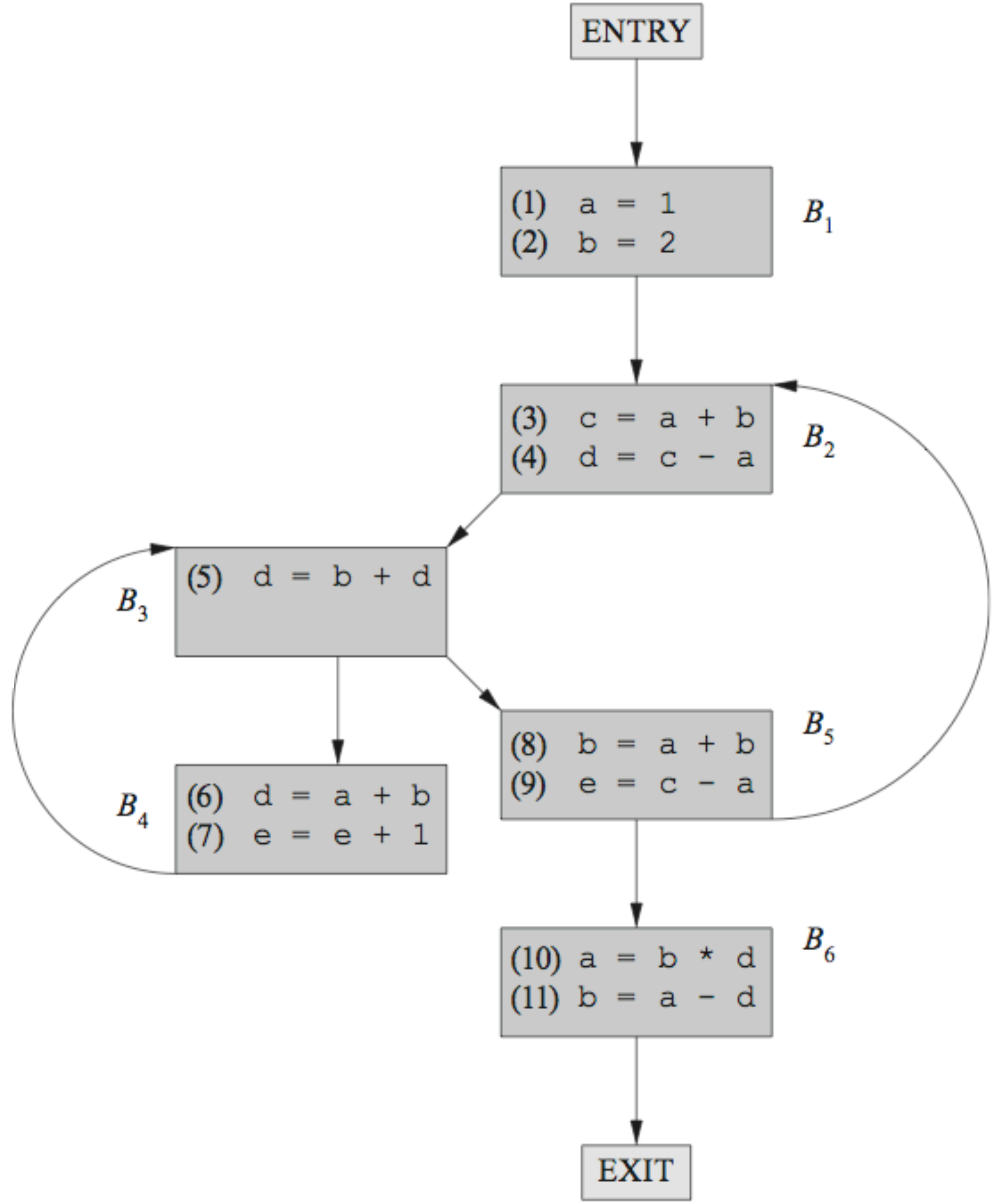
$$\mathrm{OUT}[B] = \bigcup_{S \text{ ist ein Nachfolger von } B} \mathrm{IN}[S]$$

Algorithmus

```
IN[EXIT] = ∅;
for (alle Grundblöcke B außer EXIT) IN[B] = ∅;
while (Änderungen treten an IN auf)
    for (alle Grundblöcke B außer EXIT) {
        OUT[B] = ⋃_{S ist ein Nachfolger von B} IN[S];
        IN[B] = use_B ∪ (OUT[B] - def_B);
    }
```

Beispiel

```
IN[EXIT] = ∅;
for (alle Grundblöcke B außer EXIT) IN[B] = ∅;
while (Änderungen treten an IN auf)
    for (alle Grundblöcke B außer EXIT) {
        OUT[B] = ⋃_{S ist ein Nachfolger von B} IN[S];
        IN[B] = use_B ∪ (OUT[B] - def_B);
    }
```



Verfügbare Ausdrücke

- **x OP y** ist an p **verfügbar** wenn:
  - alle Pfade von ENTRY aus werten x OP y aus und
  - nach der letzten Auswertung vor dem Erreichen von p erfolgen keine Zuweisungen zu x oder y

Transferfunktion

- Bereich: Menge an verfügbaren Ausdrücken
- Transferfunktion f(S) für x = y+z:
  - Füge zu S den Ausdruck y+z hinzu
  - Lösche alle Ausdrücke aus S, die die Variable x enthalten
- Für Blöcke: e_gen_B und e_kill_B

Beispiel

Anweisung	Verfügbare Ausdrücke
	Ø
`A = b + c`	
	$\{b+c\}$
`B = a - d`	
	$\{a-d\}$
`c = b + c`	
	$\{a-d\}$
`d = a - d`	
	Ø

Gleichungen

- OUT[ENTRY] = ∅

$$OUT[B] = e\_gen_B \cup (IN[B] - e\_kill_B)$$

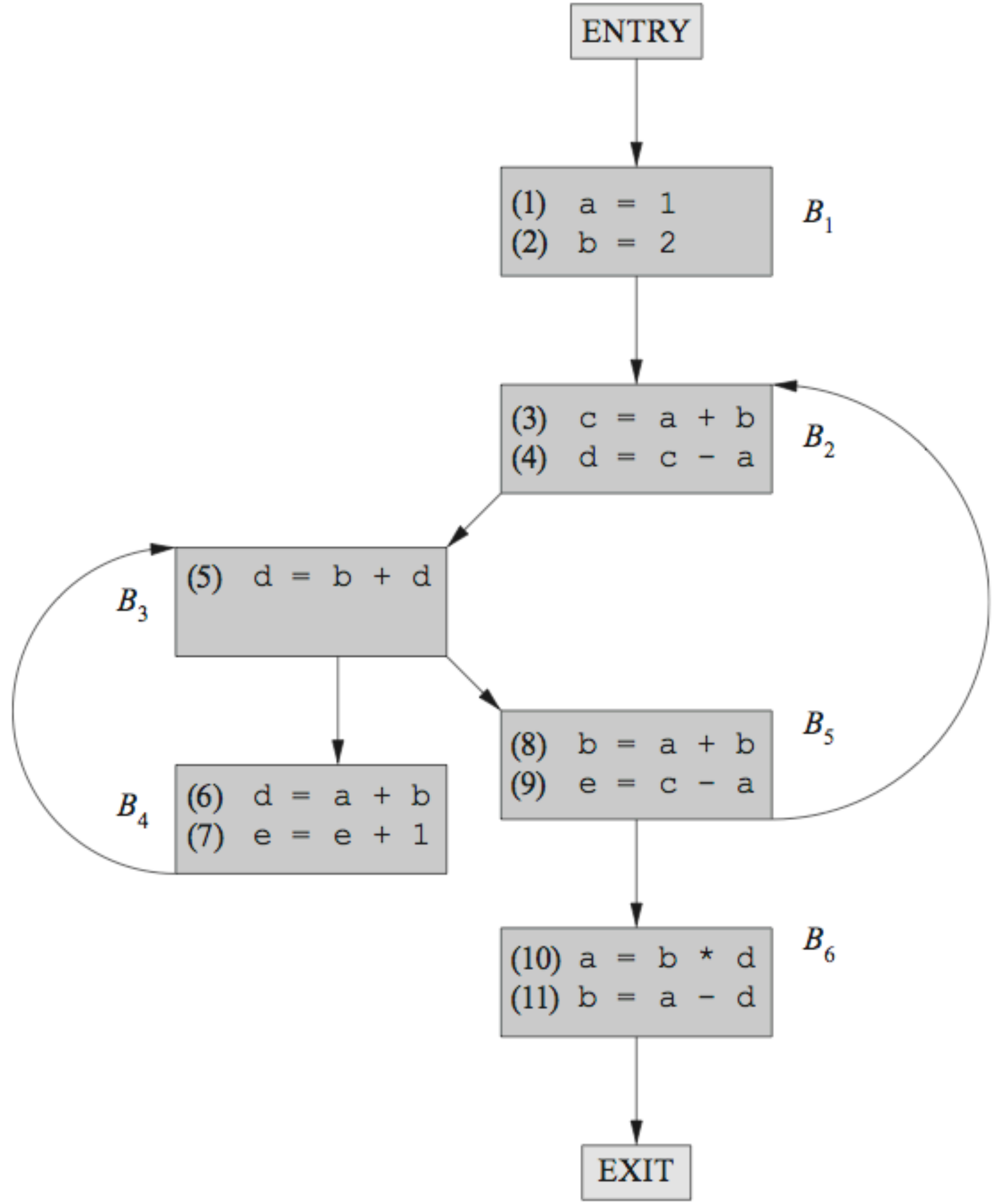
$$IN[B] = \bigcap_{P \text{ ist ein Vorgänger von } B} OUT[P]$$

Algorithmus

```
OUT[ENTRY] = Ø ;
for (jeder Grundblock B außer ENTRY) OUT[B] = U;
while (Änderungen werden an OUT vorgenommen)
     for (jeder Grundblock B außer ENTRY) {
          IN[B] = ⋂_{P ein Vorgänger von B} OUT[P];
          OUT[B] = e_gen_B ∪ (IN[B] - e_kill_B)
     }
```

Beispiel

```
OUT[ENTRY] = Ø ;
for (jeder Grundblock B außer ENTRY) OUT[B] = U
while (Änderungen werden an OUT vorgenommen)
    for (jeder Grundblock B außer ENTRY) {
        IN[B] = ∩_P ein Vorgänger von B OUT[P];
        OUT[B] = e_gen_B ∪ (IN[B] - e_kill_B)
    }
```



Warum der Algorithmus der verfügbaren Ausdrücke funktioniert

Wir müssen erklären, warum wir eine konservative Lösung für die Datenflussgleichungen bekommen, wenn wir für alle `OUT`s außer beim Eingangsblock mit der Menge aller Ausdrücke U initialisieren; d. h., warum alle Ausdrücke, deren Verfügbarkeit ermittelt wird, auch wirklich verfügbar *sind*. Erstens: Da die Meet-Operation in diesem Datenflussschema aus der Schnittmenge besteht, werden alle Gründe dafür, dass ein Ausdruck $x + y$ an einem Punkt nicht verfügbar ist, im Flussgraphen auf allen möglichen Pfaden vorwärts propagiert, bis $x + y$ neu berechnet und wieder verfügbar wird. Zweitens gibt es nur zwei Gründe, aus denen $x + y$ nicht verfügbar sein könnte:

1. $x + y$ wird in Block B zerstört, da x oder y neu definiert wird, ohne dass eine anschließende Berechnung von $x + y$ erfolgt. In diesem Fall wird $x + y$ bei der ersten Anwendung der Transferfunktion f_B aus `OUT`[B] entfernt.

2. $x + y$ wird entlang eines Pfades niemals berechnet. Da sich $x + y$ nie in `OUT`[ENTRY] befindet und niemals am fraglichen Pfad generiert wird, können wir durch Induktion entlang der Strecke des Pfades zeigen, dass $x + y$ letztendlich aus den `IN` und `OUT` entlang des Pfades entfernt wird.

Daher enthält die von dem iterativen Algorithmus aus Abbildung 9.20 bereitgestellte Lösung nach erfolgten Änderungen nur wirklich verfügbare Ausdrücke.

Datenflussanalyse

- Wann ist der Algorithmus korrekt?
- Wie genau ist die ermittelte Lösung?
- Konvergiert der iterative Algorithmus?
- Welche Bedeutung hat die Lösung der Gleichungen?

Framework der Datenflussanalyse

- (D,V, ∧,F)
  - Datenflussrichtung: Forwards, Backwards
  - Halbverband mit Wertebereich V und Durchschnittsoperator ∧
  - Familie F von Transferfunktionen von V nach V (beinhaltet auch die Grenzbedingungen für ENTRY, EXIT)

Halbverband

Bei einem *Halbverband* (semilattice) handelt es sich um eine Menge V und einen binären Durchschnittsoperator (meet) $\wedge$, sodass für alle x, y und z in V Folgendes gilt:

1. $x \wedge x = x$ (der Durchschnitt ist *idempotent*).
2. $x \wedge y = y \wedge x$ (der Durchschnitt ist *kommutativ*).
3. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ (der Durchschnitt ist *assoziativ*).

Ein Halbverband verfügt über ein oberstes Element, das mit $\top$ bezeichnet wird, sodass

$\top \wedge x = \bot$ für alle x in V

Optional kann ein Halbverband ein unterstes Element aufweisen, dass mit $\bot$ bezeichnet wird, sodass

$\bot \wedge x = x$ für alle x in V

Partielle Ordnung

- meet $\wedge$ definiert eine partielle Ordnung (reflexiv, antisymmetrisch, transitiv):
  - **x≤y genau dann wenn x** $\wedge$ y = x

- x<y g.d.w. x**≤y und x≠y**

Beispiel

- Bisher: meet $\wedge$ = $\cup$ und $\cap$
  - idempotent, kommutativ, assoziativ
- oberstes Element für $\cup$: $\emptyset$, unterstes Element für $\cup$: U da
  - $\emptyset \cup x = x \Rightarrow x \leq \emptyset$ und $U \cup x = U$
- Wertebereich V = POW(U)

GLB (Greatest Lower Bound)

- **Größte untere Schranke** GLB g von x und y ist so dass:
  - g **≤ x und g≤y und**
  - ∀z. (z**≤x & z≤y)** ⇒ z**≤g**
- meet (x∧y) ist GLB:
  - x∧y **≤ x da (x**∧y)∧x = x∧y + gleiches für y
  - \+ Beweis für ∀z... im Buch

Verbanddiagramme

- Kante nach unten x → y wenn y**≤x**
- GLB kann abgelesen werden

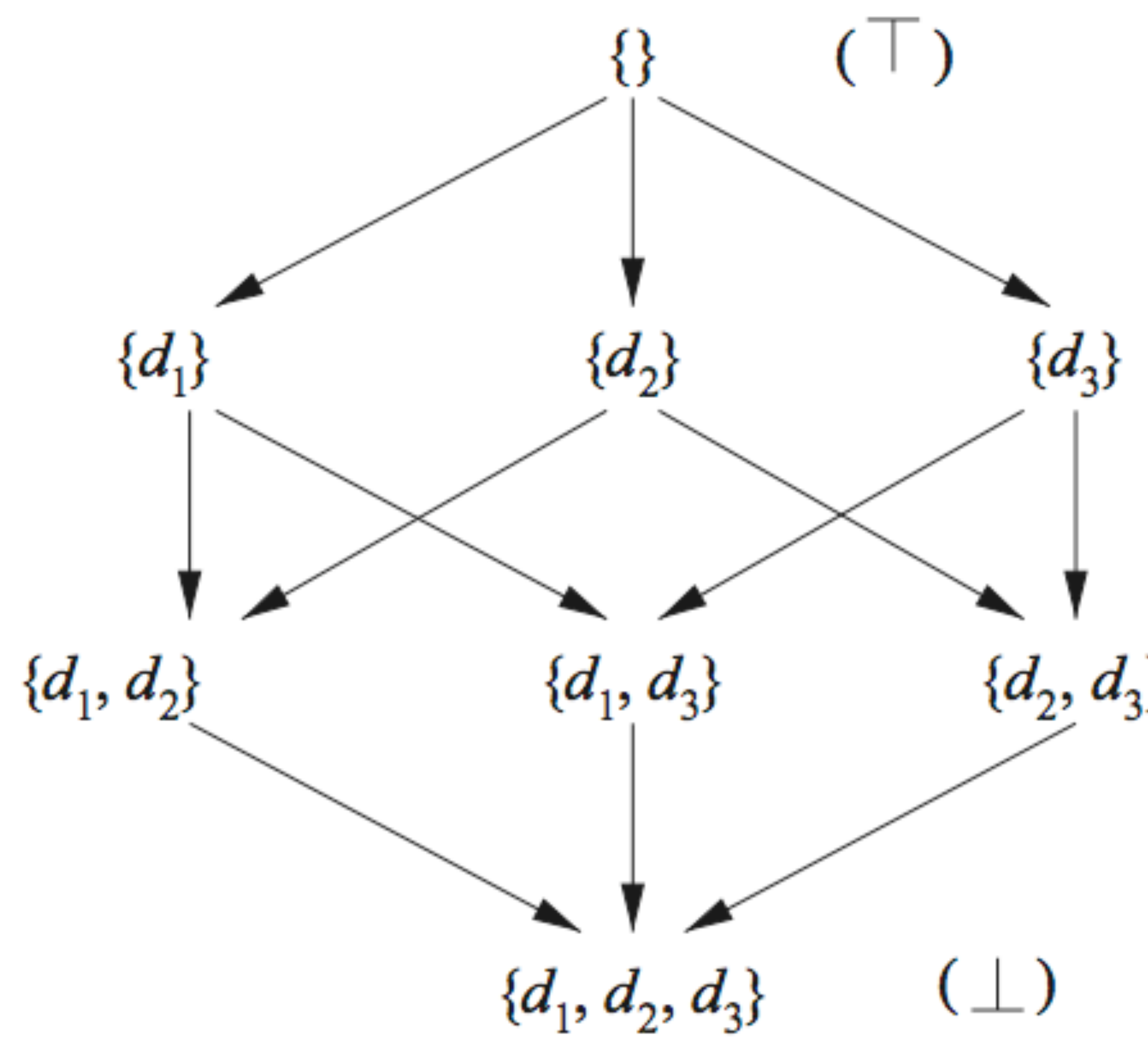


Abbildung 9.22: Verband aus Teilmengen von Definitionen

Produktverbände

- Produktverband von $(A,\wedge_A)$ und $(B,\wedge_B)$
  - Bereich $A\times B$
  - $(a,b) \wedge (a',b') = (a\wedge_A a', b\wedge_B b')$
  - (a,b) **$\leq$ (a',b') bedeutet a**$\leq_A a'$ und $b\leq_B b'$

Anwendung: Bei erreichenden Definition ein Unter-Verband pro Def.

Höhe eines Halbverbands

- aufsteigende Kette: Folge $x_1 < x_2 < \ldots < x_n$
- Höhe: größte Anzahl von <-Relatonen in einer aufsteigenden Kette

Transferfunktionen

Die Familie der Transferfunktionen $F: V \to V$ in einem Datenfluss-Framework verfügt über folgende Eigenschaften:

1. F hat eine Identitätsfunktion I, sodass $I(x) = x$ für alle x in V.
2. F ist geschlossen unter Komposition; d. h., für zwei beliebige Funktionen f und g in F ist die von $h(x) = g(f(x))$ definierte Funktion auch in F.

Monotone Frameworks

- (D,F,V, $\wedge$) monoton wenn $\forall x,y \in D$, $f \in F$:
  - **$x \leq y$ impliziert $f(x) \leq f(y)$**
  - (äquiv. mit: $f(x \wedge y)$ **$\leq$ $f(x)$** $\wedge$ $f(y)$)
- (D,F,V, $\wedge$) distributiv wenn $f(x \wedge y)$ **=** $f(x) \wedge f(y)$

Generischer Algorithmus (Forward)

```
1) OUT[ENTRY] = v_ENTRY;
2) for (jeder Grundblock B außer ENTRY) OUT[B] = ⊤;
3) while (Änderungen an OUT erfolgen)
4)      for (jeder Grundblock B außer ENTRY) {
5)           IN[B] = ∧_{P ist ein Vorgänger von B} OUT[P];
6)           OUT[B] = f_B (IN[B]);
        }
```

Generischer Algorithmus (Backward)

IN[EXIT] = v_{EXIT};
for (jeder Grundblock B außer EXIT) IN[B] = $\top$;
while (Änderungen an IN erfolgen)
    for (jeder Grundblock B außer EXIT) {
        OUT[B] = $\wedge_{S \text{ ist ein Nachfolger von } B}$ IN[S];
        IN[B] = f_B (OUT[B]);
    }

Eigenschaften

1. Wenn Algorithmus 9.25 konvergiert, ist das Ergebnis eine Lösung der Datenflussgleichungen.

2. Wenn das Framework monoton ist, dann ist die ermittelte Lösung der maximale Fixpunkt (MFP) der Datenflussgleichungen. Bei einem maximalen Fixpunkt handelt es sich um eine Lösung mit der Eigenschaft, dass in jeder anderen Lösung die Werte von $\mathtt{IN}[B]$ und $\mathtt{OUT}[B]$ kleiner oder gleich ($\leq$) den entsprechenden Werten des MFP sind.

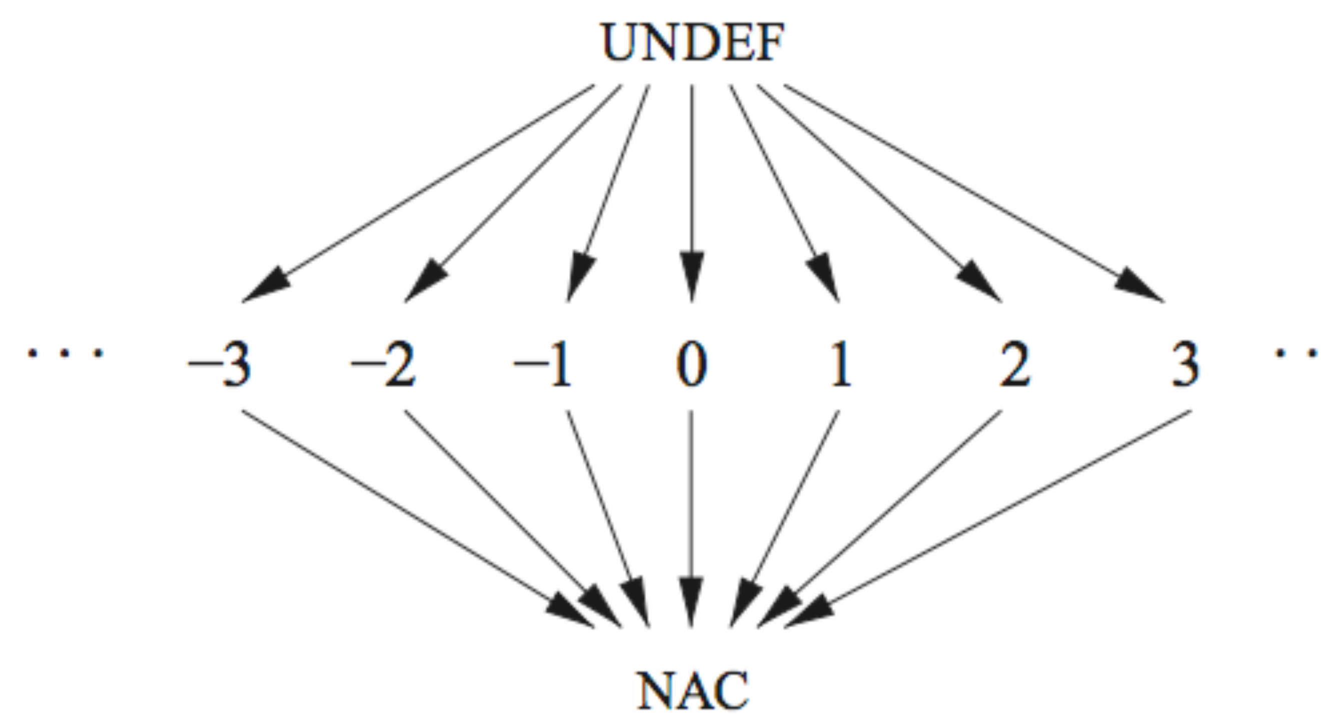
3. Wenn der Halbverband des Framework monoton ist und eine endliche Höhe aufweist, dann ist die Konvergenz des Algorithmus gewährleistet.

Neues Beispiel: Konstantenpropagation

- uneingeschränkte Menge an Datenflusswerten
- Pro Variable Verband mit:
  - alle möglichen Konstantenwerte
  - NAC (not a constant)
  - UNDEF (undefiniert)

Meet

- UNDEF ∧ v = v
- NAC ∧ v = NAC
- c ∧ c = c
- c1 ∧ c2 = NAC

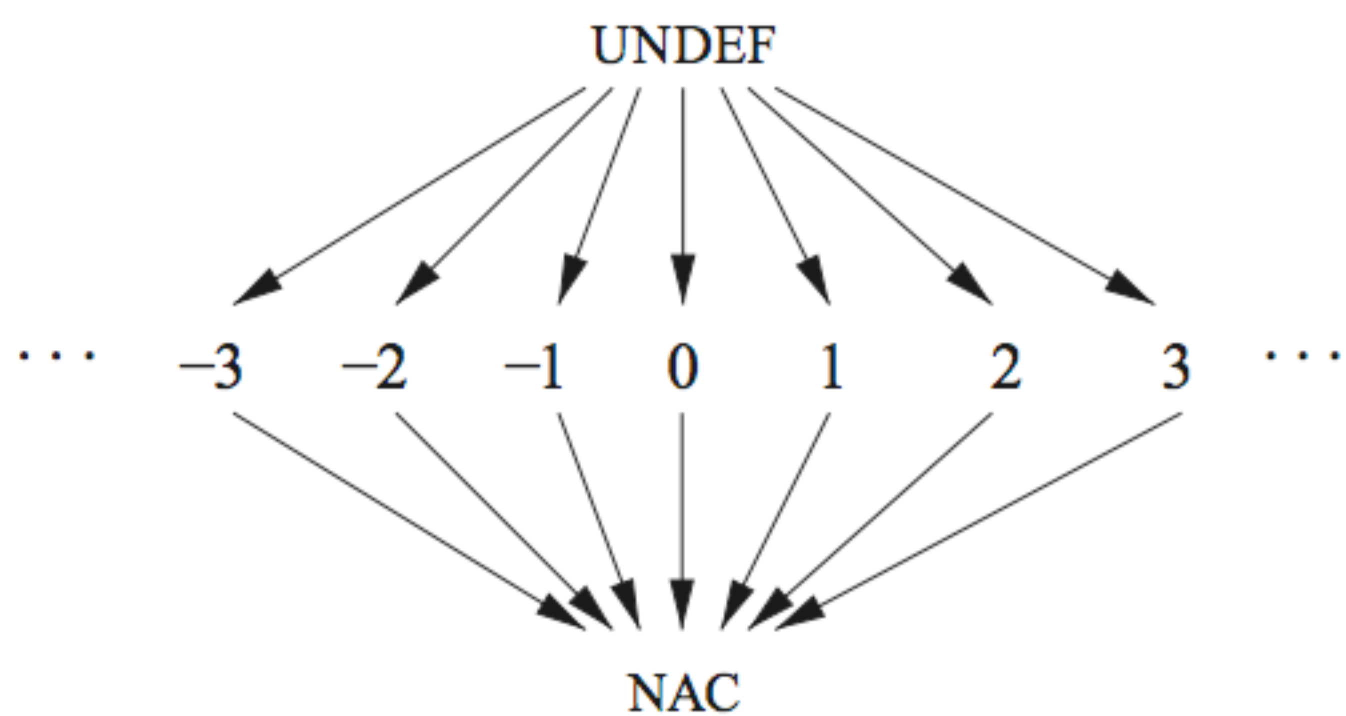


Transferfunktionen

- s Anweisung; $m' = f_s(m)$
  - (1) Falls s keine Zuweisung: $m' = m$
  - (2) Falls s Zuweisung x = RHS dann $m'(y)=m(y)$ für $x \neq y$ und
    - (a) $m'(x) = c$ falls RHS Konstante c
    - (b) RHS = y OP z dann
      - $m'(x) = m(y)+m(z)$ wenn $m(y),m(z)$ Konstanten
      - $m'(x)$ = NAC wenn $m(y)$ oder $m(z)$ = NAC
      - sonst $m'(x)$ = UNDEF
    - (c) sonst (Funktionsaufruf,...): $m'(x)$ = NAC

Monotonie ?

- x≤**y impliziert f(x) ≤ f(y) ?**



$m(y)$	$m(z)$	$m'(x)$
UNDEF	UNDEF	UNDEF
	c_2	UNDEF
	NAC	NAC
c_1	UNDEF	UNDEF
	c_2	$c_1 + c_2$
	NAC	NAC
NAC	UNDEF	NAC
	c_2	NAC
	NAC	NAC

Optimale/Ideale Lösung

- Ideallösung:

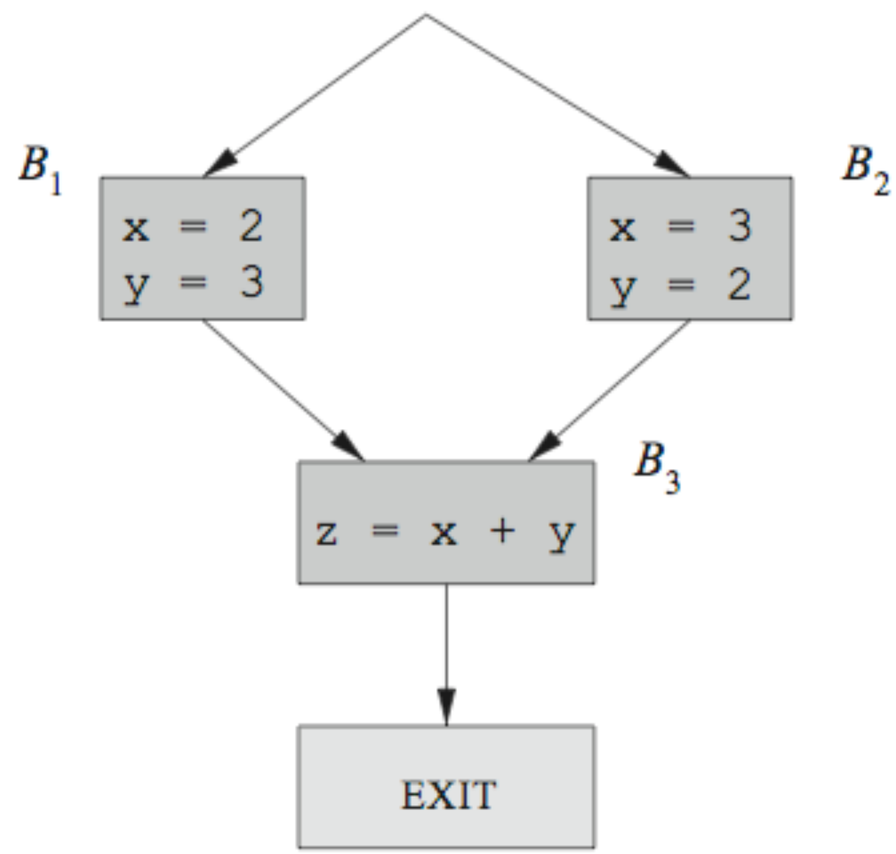
$$\texttt{IDEAL}[B] = \bigwedge_{P,\text{ ein möglicher Pfad von ENTRY nach } B} f_P(v_{\text{ENTRY}})$$

- Meet Over Paths Lösung:

$$\texttt{MOP}[B] = \bigwedge_{P,\text{ ein Pfad von ENTRY nach } B} f_P(v_{\text{ENTRY}})$$

- MFP **≤ MOP ≤ IDEAL**
Bei distributivem Framework (f(x∧y) = f(x) ∧ f(y)): MFP = MOP

Konstantpropagation: Nicht Distributiv



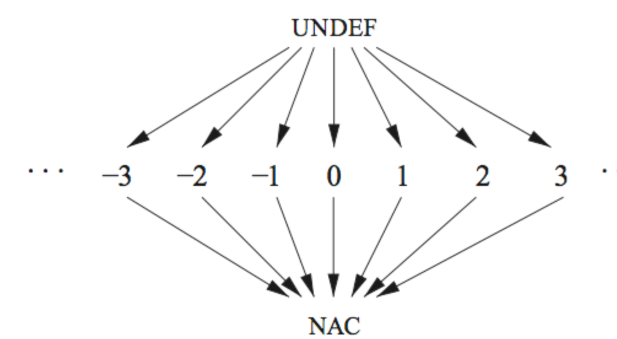
- $f(x \wedge y) \neq f(x) \wedge f(y)$

Zusatzmaterial

- (Nicht im Drachenbuch)
- Zusammenhang zwischen Datenflussanalyse und abstrakter Interpretation?

Konstantenpropagation als Abstr. Interp.

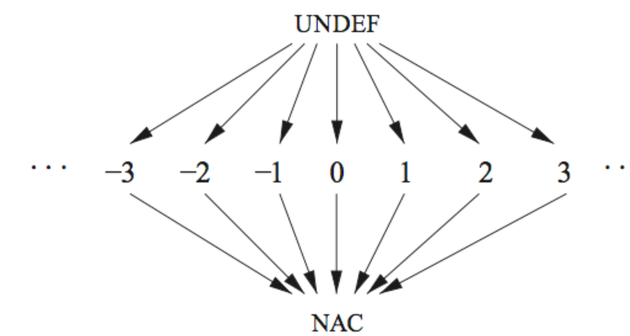
- γ(NAC) = Z und γ(UNDEF) = ∅
- γ(x) = {x}   für x∈Z
- α(∅) = UNDEF, α({x}) = x
- α(S) = NAC in allen anderen Fällen
- Transferfunktion = abstrakte Version
  - Korrekt?



$m(y)$	$m(z)$	$m'(x)$
UNDEF	UNDEF	UNDEF
	c_2	UNDEF
	NAC	NAC
c_1	UNDEF	UNDEF
	c_2	c_1+c_2
	NAC	NAC
NAC	UNDEF	NAC
	c_2	NAC
	NAC	NAC

Transferfunktion: Korrektheit

- $X +^{*} Y \subseteq \gamma(\ \alpha(X)\ \ +^{\alpha}\ \ \alpha(Y)\ )$



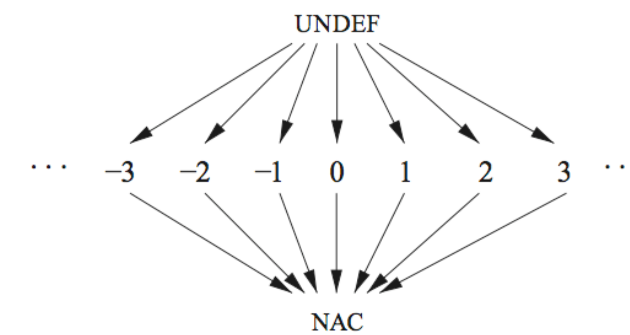
  - $\{x\} +^{*} \{y\} = \{x+y\}$
    $\gamma(\ \alpha(\{x\})\ \ +^{\alpha}\ \ \alpha(\{y\})\ ) = \gamma(x +^{\alpha} y) = \gamma(x+y)$

  - $\{x\} +^{*} \{y,z\} = \{x+y, x+z\}\ \ (y \neq z)$
    $\gamma(\ \alpha(\{x\})\ \ +^{\alpha}\ \ \alpha(\{y,z\})\ ) =$
    $\gamma(x +^{\alpha} \text{NAC}) = \gamma(\text{NAC}) = Z$

  - $\{\} +^{*} \{y,z\} = \{\}$
    $\gamma(\ \alpha(\{\})\ \ +^{\alpha}\ \ \alpha(\{y,z\})\ ) = Z$

$m(y)$	$m(z)$	$m'(x)$
UNDEF	UNDEF	UNDEF
	c_2	UNDEF
	NAC	NAC
c_1	UNDEF	UNDEF
	c_2	c_1+c_2
	NAC	NAC
NAC	UNDEF	NAC
	c_2	NAC
	NAC	NAC

Transferfunktion: Optimalität

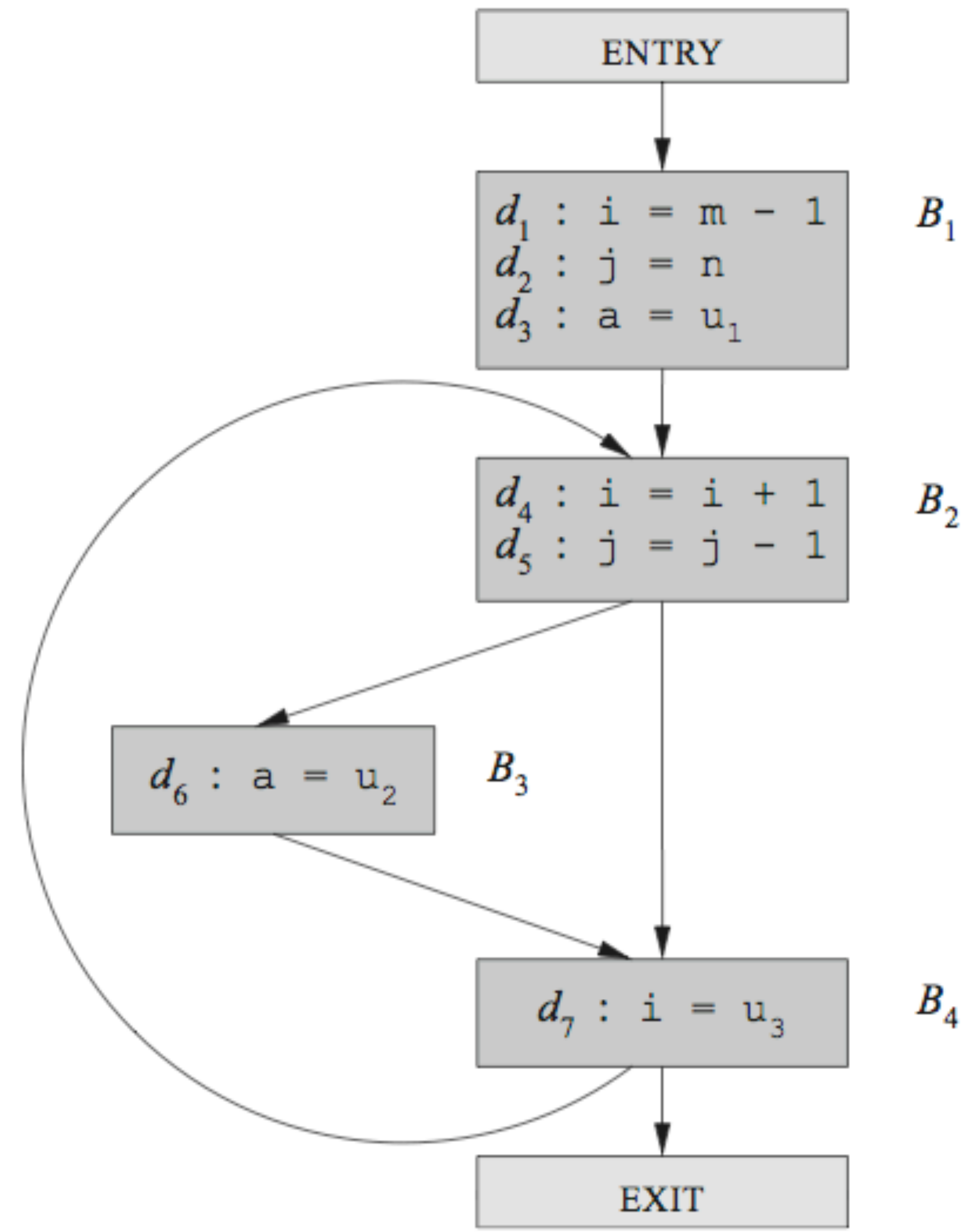


- $A +^{\alpha} B = \alpha(\ \gamma(A)\ \ +^{*}\ \ \gamma(B)\ )$
  - $\alpha(\{x\} +^{*} \{y\}) = \alpha(\{x+y\}) = x+y$
    $x\ \ +^{\alpha} y = x+y$
  - $\alpha(\{x\} +^{*} Z) =$
    $\alpha(Z) = NAC$
    $x\ \ +^{\alpha} NAC = NAC$
  - $\alpha(\{\} +^{*} Z) = \alpha(\{\}) = UNDEF$
    $UNDEF\ \ +^{\alpha}\ \ NAC =$ **NAC**

$m(y)$	$m(z)$	$m'(x)$
UNDEF	UNDEF	UNDEF
	c_2	UNDEF
	NAC	NAC
c_1	UNDEF	UNDEF
	c_2	c_1+c_2
	NAC	NAC
NAC	UNDEF	NAC
	c_2	NAC
	NAC	NAC

Erreichende Definitionen als A.I.?

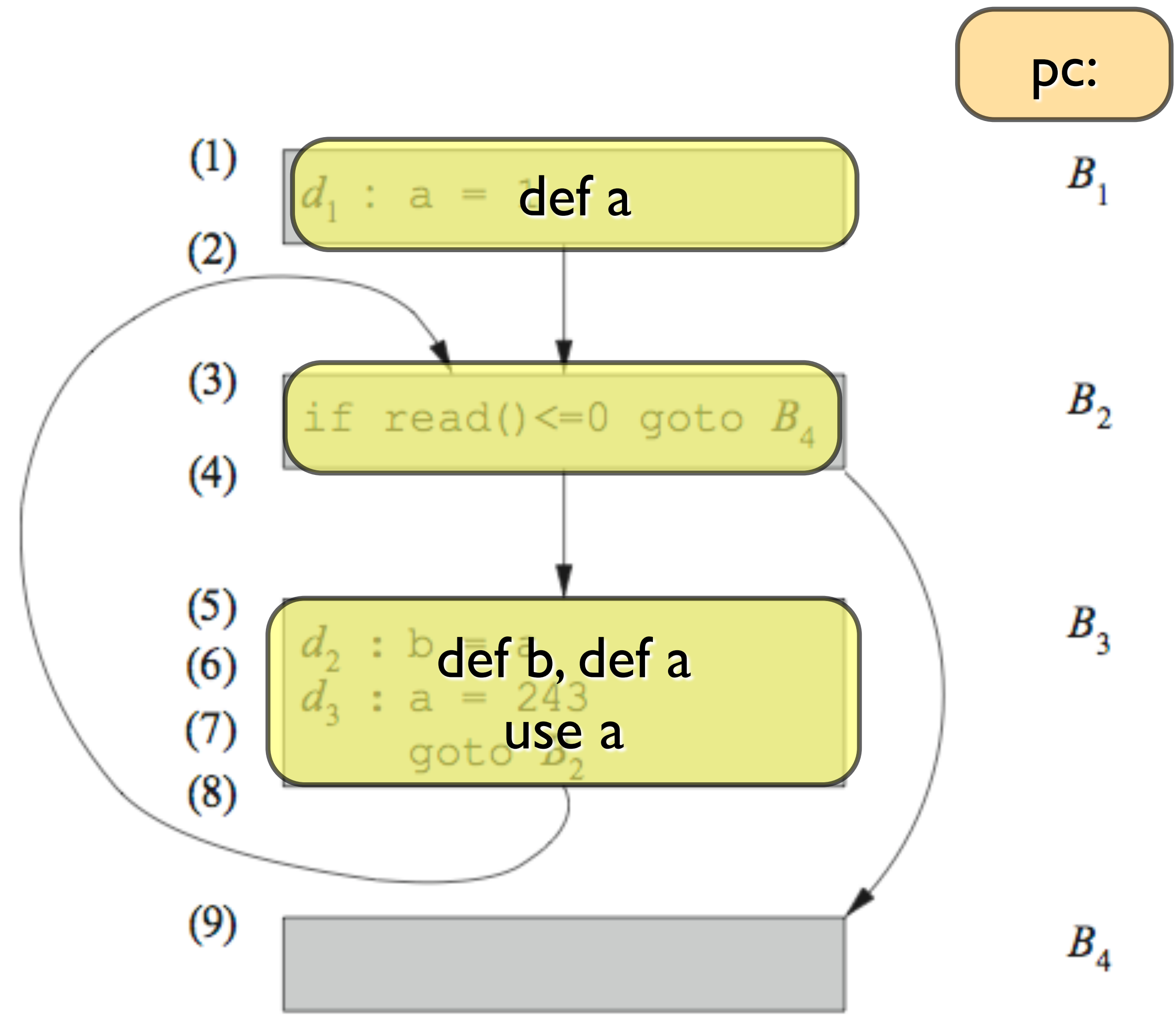
- Anfang von B2: {d1,d2,d3,d5,d6,d7}
- γ({d1,d2,d3,d5,d6,d7}) ?



Erreichende Definitionen als A.I.?

- def x in Bi reaches Bj:

  EF def x & pc=Bi =>
  (not def x EU pc=Bj)



Paper by Schmidt: Dataflow Analysis is Model Checking of Abstract Interpretations, POPL 98.

Zusammenfassung

- Grundlagen der Datenflussanalyse
  - monotone Frameworks, generischer Algorithmus, ...
- Verschiedene Analysen
  - Konstanteprop, Erreichende Def, Liveness, Verfügbare Ausdrücke
- Zusammenhang mit Abs. Int. + MC

Ausblick

- Interprocedurale Analyse
  - Kapitel 12
  - Problem der Rekursion
  - Datalog, BDDs