

## Idee:

- (1) Wende **T1** an, d.h. ersetze jede interessierende Zuweisung  $x=e;$  durch:  $T_e = e; x = T_e;$
- (2) Finde alle Stellen, an denen  $e$  **sicher** berechnet werden kann, ohne die Semantik zu zerstören.
- (3) Platziere (konzeptuell)  $T_e = e;$  an allen diesen Plätzen.  
Beseitige die redundanten Zuweisungen mittels **T2**.

$\implies$  wir benötigen eine neue Analyse :-))

Ein Ausdruck  $e$  heißt **aktiv** (busy) entlang eines Pfads  $\pi$ , falls der Wert von  $e$  berechnet wird, bevor eine der Variablen  $x \in \text{Vars}(e)$  überschrieben wird.

// Rückwärtsanalyse!

$e$  heißt **sehr aktiv** (very busy) an  $u$ , falls  $e$  aktiv ist entlang jedes Pfads  $\pi : u \rightarrow^* \text{stop}$ .

Ein Ausdruck  $e$  heißt **aktiv** (busy) entlang eines Pfads  $\pi$ , falls der Wert von  $e$  berechnet wird, bevor eine der Variablen  $x \in \text{Vars}(e)$  überschrieben wird.

// Rückwärtsanalyse!

$e$  heißt **sehr aktiv** (very busy) an  $u$ , falls  $e$  aktiv ist entlang jedes Pfads  $\pi : u \rightarrow^* \text{stop}$ .

Entsprechend benötigen wir:

$$\mathcal{B}[u] = \bigcap \{ \llbracket \pi \rrbracket^\# \emptyset \mid \pi : u \rightarrow^* \text{stop} \}$$

wobei für  $\pi = k_1 \dots k_m$ :

$$\llbracket \pi \rrbracket^\# = \llbracket k_1 \rrbracket^\# \circ \dots \circ \llbracket k_m \rrbracket^\#$$

Unser vollständiger Verband ist:

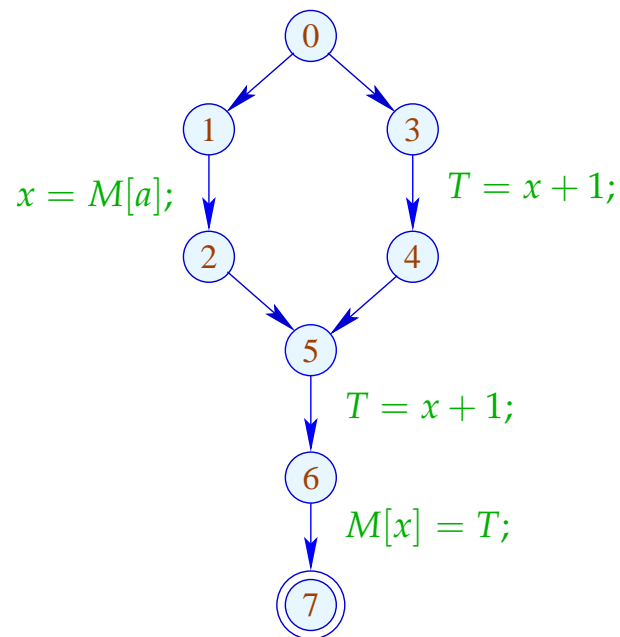
$$\mathbb{B} = 2^{Expr \setminus Vars} \quad \text{mit} \quad \sqsubseteq = \supseteq$$

Der Effekt  $\llbracket k \rrbracket^\#$  einer Kante  $k = (u, lab, v)$  hängt nur von  $lab$  ab, d.h.  $\llbracket k \rrbracket^\# = \llbracket lab \rrbracket^\#$  wobei:

$$\begin{aligned} \llbracket ; \rrbracket^\# B &= B \\ \llbracket Pos(e) \rrbracket^\# B &= \llbracket Neg(e) \rrbracket^\# B = B \cup \{e\} \\ \llbracket T_e = e; \rrbracket^\# B &= (B \setminus Expr_{T_e}) \cup \{e\} \\ \llbracket x = T; \rrbracket^\# B &= B \setminus Expr_x \\ \llbracket x = M[R]; \rrbracket^\# B &= B \setminus Expr_x \\ \llbracket M[R] = x; \rrbracket^\# B &= B \end{aligned}$$

Die Kanten-Effekte sind sämtlich **distributiv**. Deshalb liefert die kleinste Lösung des Constraint-Systems exakt den MOP :-)

Beispiel:



7	$\emptyset$
6	$\emptyset$
5	$\{x + 1\}$
4	$\{x + 1\}$
3	$\{x + 1\}$
2	$\{x + 1\}$
1	$\emptyset$
0	$\emptyset$

## Beachte:

- Im Beispiel enthalten die  $\mathcal{B}[u]$  maximal ein Element.
- Enthält  $\mathcal{B}[u]$  mehrere Ausdrücke  $e_1 \neq e_2$ , sind diese **unabhängig**, d.h.  $T_{e_1} \notin \text{Vars}(e_2)$  :-)
- Unabhängige Ausdrücke können in **beliebiger Reihenfolge** berechnet werden :-))

## Beachte:

- Im Beispiel enthalten die  $\mathcal{B}[u]$  maximal ein Element.
- Enthält  $\mathcal{B}[u]$  mehrere Ausdrücke  $e_1 \neq e_2$ , können diese stets in einer Reihenfolge  $e_1 \rightarrow e_2$  ausgewertet werden, so dass  $T_{e_2} \notin \text{Vars}(e_1)$  :-)

## Beweis der Anordbarkeit:

→ Wir zeigen Beh. für  $[[\pi]]^\# \emptyset$ . Die Beh. für  $u$  folgt, da die Eigenschaft unter  $\cap$  abgeschlossen ist.

→ Induktion über die Länge von  $\pi$ .

$$\boxed{\pi = \epsilon} \quad [[\pi]]^\# \emptyset = [[\epsilon]]^\# \emptyset = \emptyset \quad \text{:-)}$$

$$\boxed{\pi = k \pi'} \quad \text{Kanten-Effekte erhalten die Anordbarkeit :-))}$$

Ein  $u$  heißt **sicher** für  $e$ , sofern  $e \in \mathcal{A}[u] \cup \mathcal{B}[u]$ ; d.h.  $e$  ist entweder verfügbar oder sehr aktiv.

Ist  $u$  sicher, können wir dort  $e$  gefahrlos berechnen :-)

## Idee:

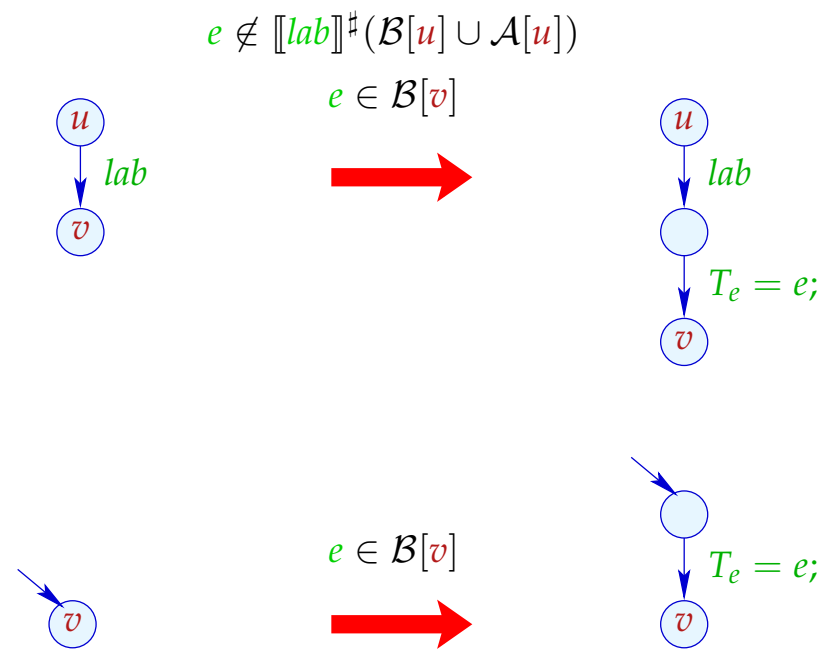
- Wir berechnen  $e$  zum frühest möglichen Zeitpunkt :-)
- Wir platzieren die Berechnung von  $e$  am Ende von  $k = (u, lab, v)$  falls:
  - $e \in \mathcal{B}[v]$  sowie
  - $e \notin \llbracket lab \rrbracket^\#(\mathcal{A}[u])$  (nicht verfügbar entlang  $k$ ) und
  - $e \notin \llbracket lab \rrbracket^\#(\mathcal{B}[u])$  (auch nicht nach Transformation)
- Weil alle  $e \in \mathcal{B}[v]$  anordbar sind, betrachten wir die Transformation für jedes  $e$  gesondert:



## Transformation 6.1:



## Transformation 6.2:



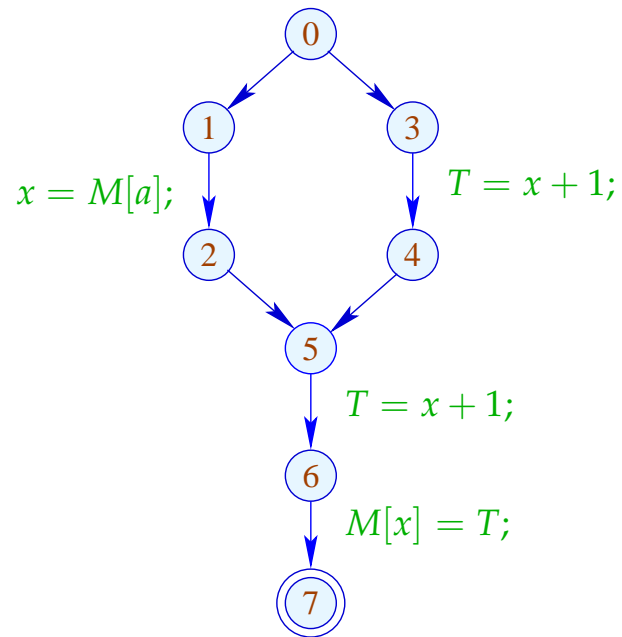


Bernhard Steffen, Dortmund



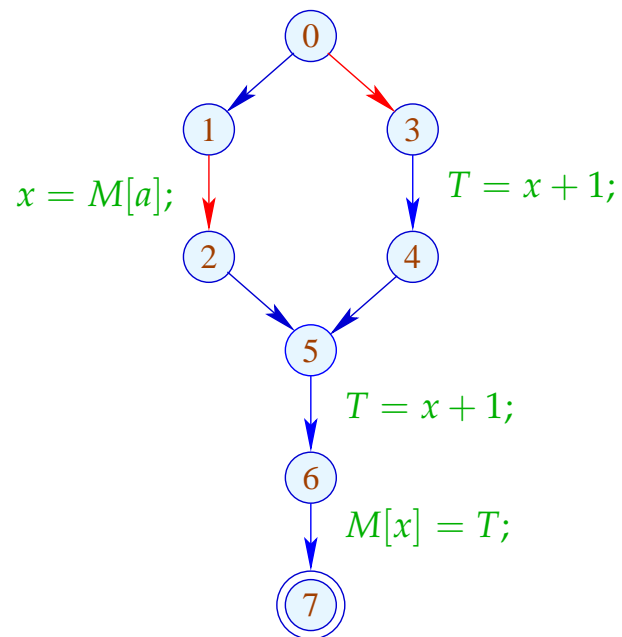
Jens Knoop, Wien

Im Beispiel:



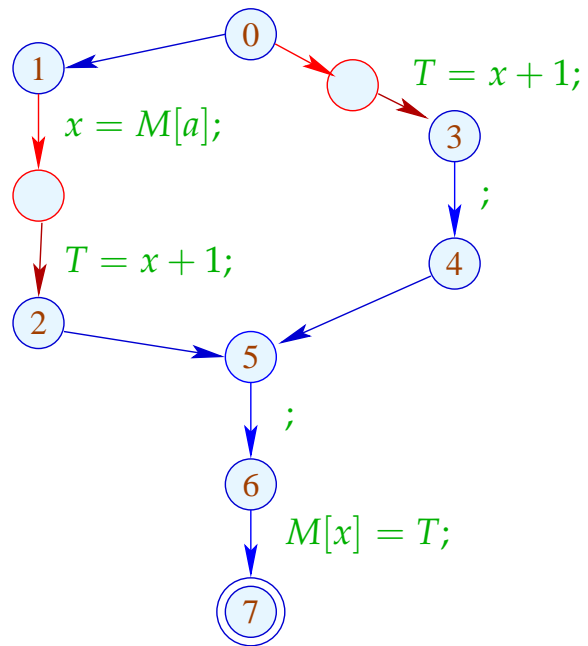
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\emptyset$
7	$\{x + 1\}$	$\emptyset$

Im Beispiel:



	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\emptyset$
7	$\{x + 1\}$	$\emptyset$

Im Beispiel:



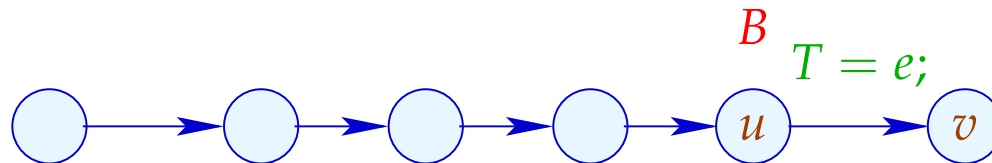
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\emptyset$
7	$\{x + 1\}$	$\emptyset$

## Zur Korrektheit:

Sei  $\pi = \pi' k$  ein Pfad mit  $k = (u, T = e, v)$ .

Dann gilt:  $e \in \mathcal{B}[u]$  da  $u$  nur eine ausgehende Kante hat :-)

Wir haben:

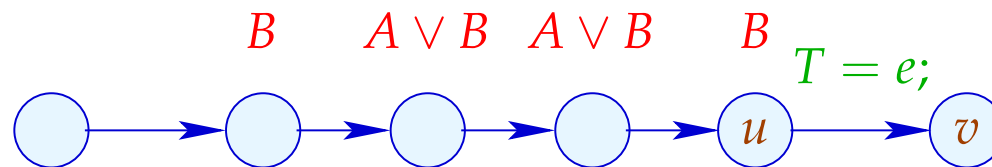


## Zur Korrektheit:

Sei  $\pi = \pi' k$  ein Pfad mit  $k = (u, T = e, v)$ .

Dann gilt:  $e \in \mathcal{B}[u]$  da  $u$  nur eine ausgehende Kante hat :-)

Wir haben:

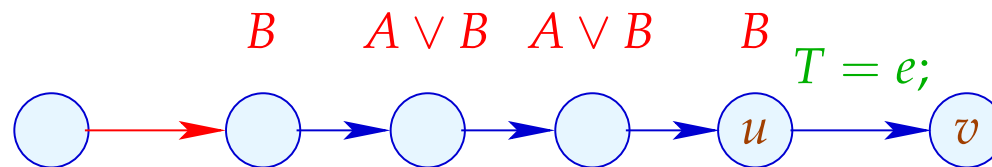


## Zur Korrektheit:

Sei  $\pi = \pi' k$  ein Pfad mit  $k = (u, T = e, v)$ .

Dann gilt:  $e \in \mathcal{B}[u]$  da  $u$  nur eine ausgehende Kante hat :-)

Wir haben:



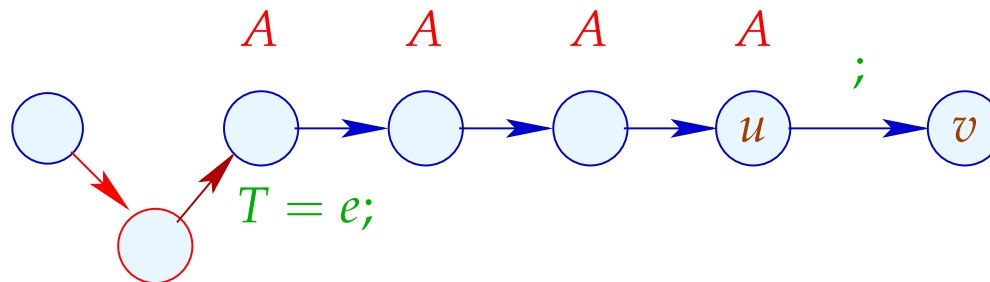


## Zur Korrektheit:

Sei  $\pi = \pi' k$  ein Pfad mit  $k = (u, T = e, v)$ .

Dann gilt:  $e \in \mathcal{B}[u]$  da  $u$  nur eine ausgehende Kante hat :-)

Wir haben:



## Wir schließen:

- Überall, wo wir  $T = e;$  gestrichen haben, ist  $e$  verfügbar :-)
- ⇒ **Korrektheit** der Transformation
- Jedem  $T = e;$ , das wir in einen Pfad einfügen, entspricht ein  $T = e;$ , das wir gestrichen haben :-))
- ⇒ **Nicht-Verschlechterung** der Transformation

## 1.8 Anwendung: Schleifen-invarianter Code

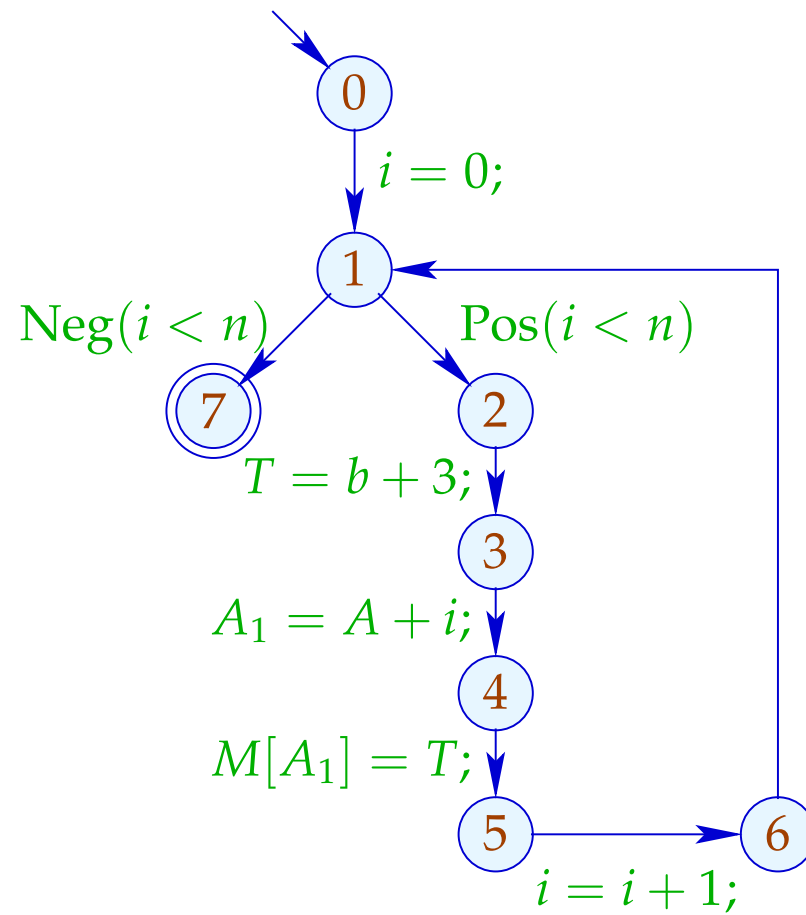
Beispiel:

```
for ( $i = 0; i < n; i++$ )  
     $a[i] = b + 3;$ 
```

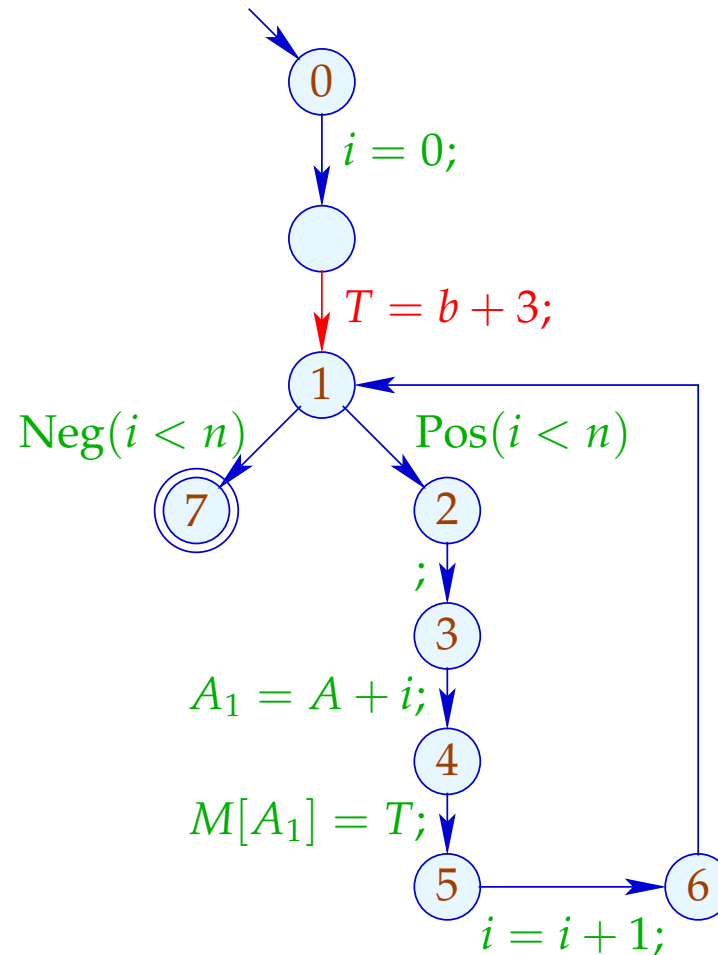
// Der Ausdruck  $b + 3$  wird in jeder Iteration berechnet :-)

// Das wollen wir vermeiden :-)

# Der Kontrollfluss-Graph:

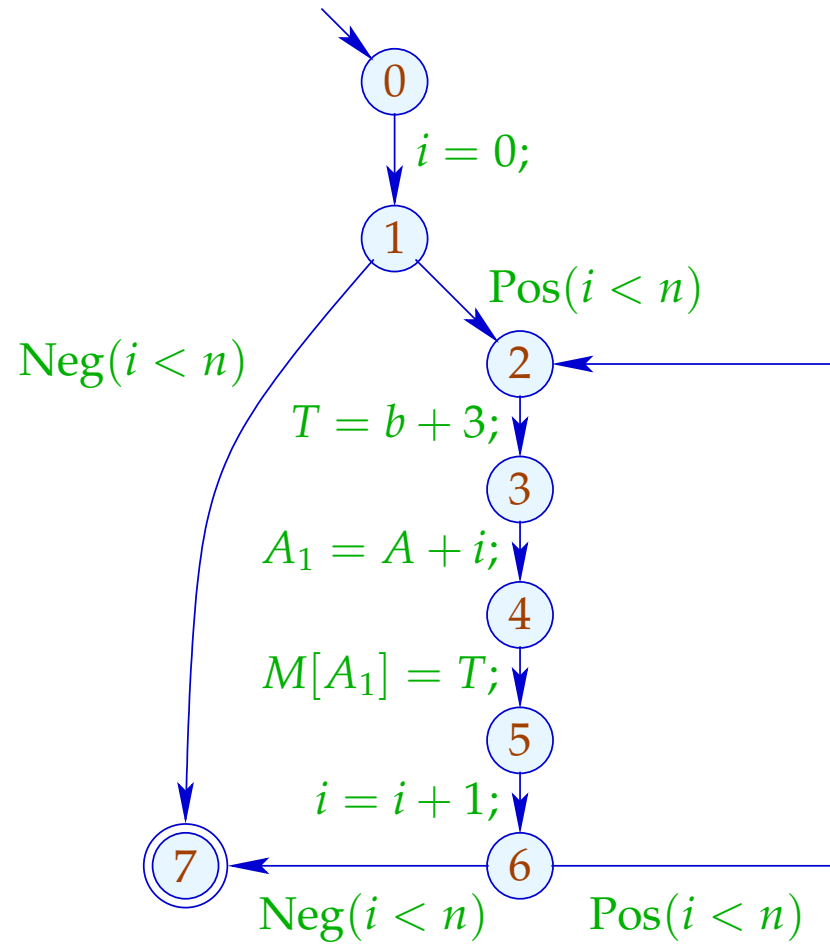


Achtung:  $T = b + 3$ ; darf nicht vor der Schleife stehen :

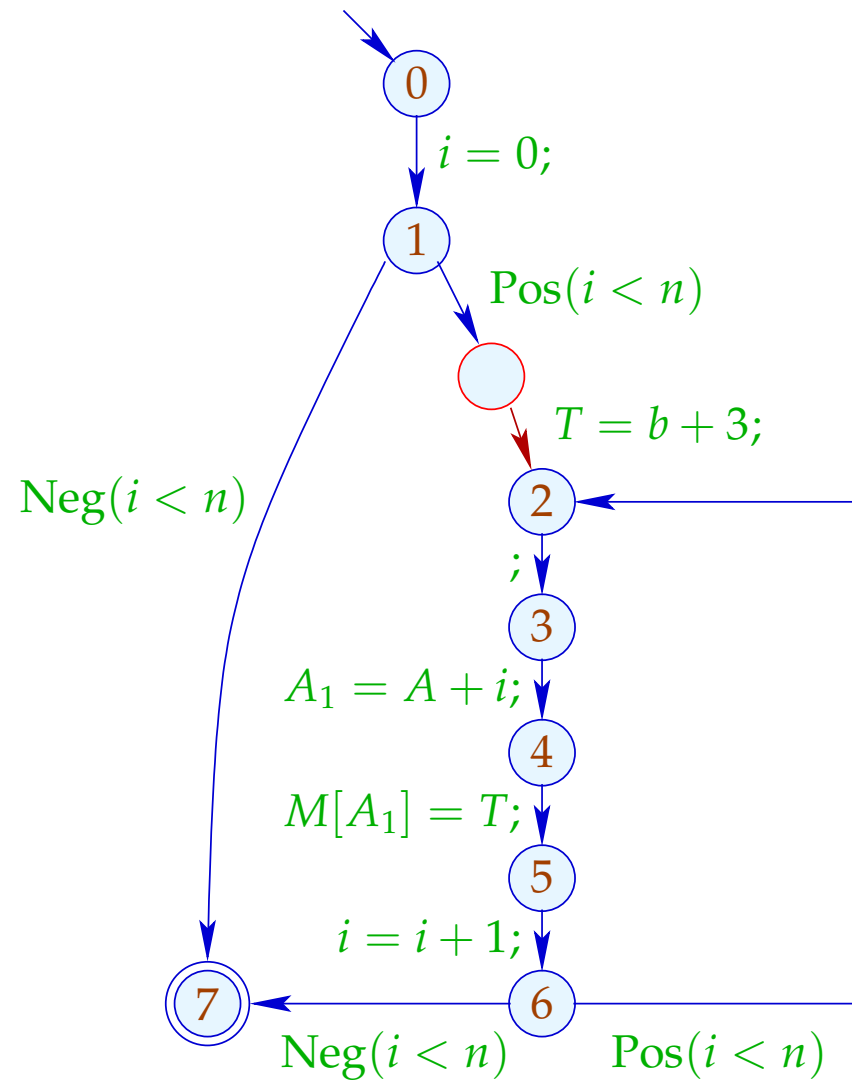


⇒ Es gibt keinen guten Platz für  $T = b + 3$ ; :-)

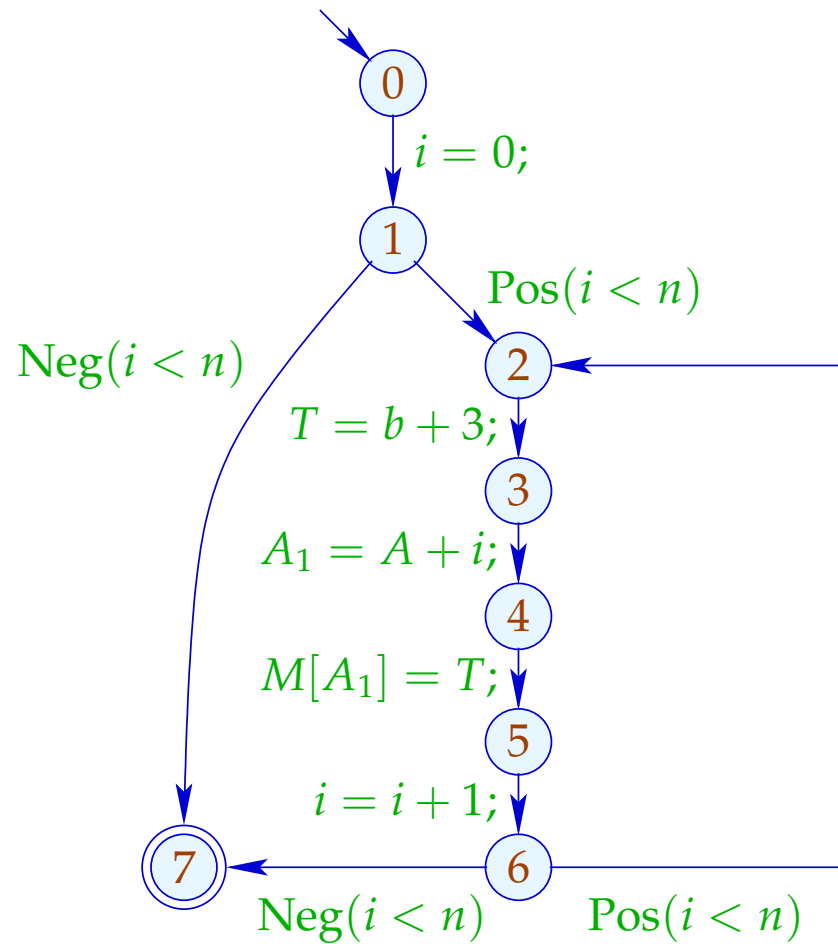
Idee: Transformiere in eine **do-while**-Schleife ...



... jetzt gibt es eine Stelle für  $T = e;$  :-)



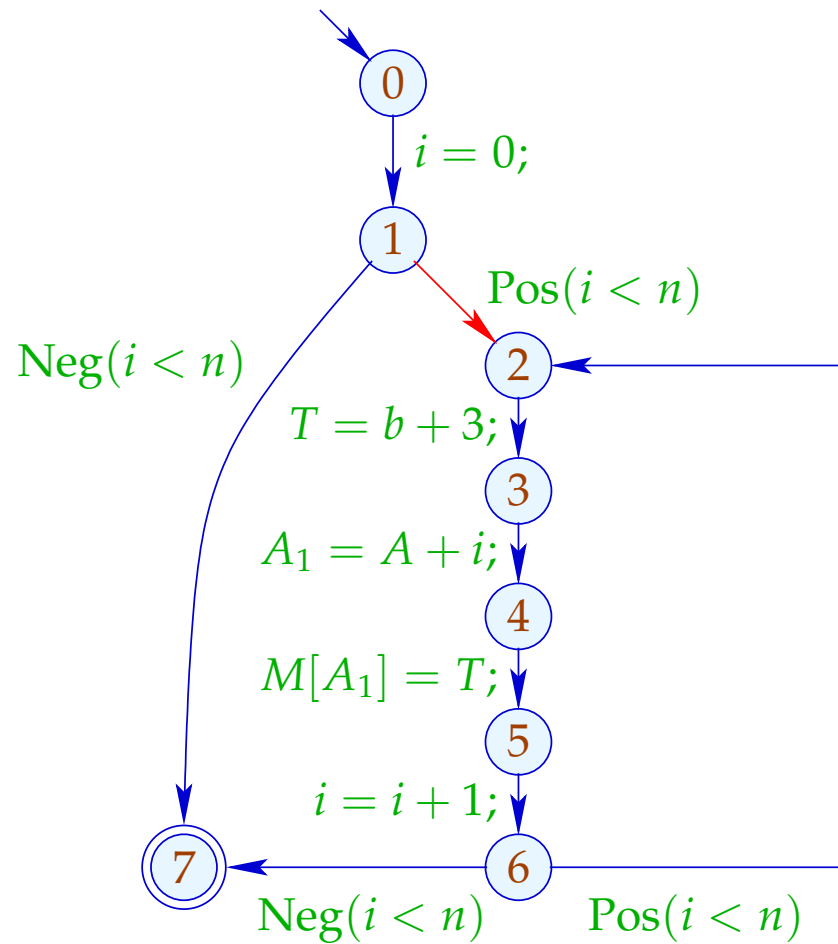
Anwendung von **T6** (PRE) :



	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{b + 3\}$
3	$\{b + 3\}$	$\emptyset$
4	$\{b + 3\}$	$\emptyset$
5	$\{b + 3\}$	$\emptyset$
6	$\{b + 3\}$	$\emptyset$
6	$\emptyset$	$\emptyset$
7	$\emptyset$	$\emptyset$



Anwendung von **T6** (PRE) :



	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{b + 3\}$
3	$\{b + 3\}$	$\emptyset$
4	$\{b + 3\}$	$\emptyset$
5	$\{b + 3\}$	$\emptyset$
6	$\{b + 3\}$	$\emptyset$
6	$\emptyset$	$\emptyset$
7	$\emptyset$	$\emptyset$

## Fazit:

- Beseitigung partieller Redundanzen kann loop-invarianten Code aus Schleifen heraus schieben :-))
- Das funktioniert nur für do-while-Schleifen :-(
- Um andere Schleifen zu optimieren, wandeln wir sie in do-while-Schleifen um:

`while (b) stmt`  $\implies$  `if (b)`  
`do stmt`  
`while (b);`

$\implies$  Schleifen-Rotation

## Problem:

Haben wir das Quell-Programm nicht (mehr) zur Verfügung, müssen wir nachträglich die Schleifen (-köpfe) identifizieren ;-)

$\Longrightarrow$  Prädominatoren

$u$  prädominiert  $v$ , falls jeder Pfad  $\pi : start \rightarrow^* v$  Knoten  $u$  enthält. Wir schreiben:  $u \Rightarrow v$ .

“ $\Rightarrow$ ” ist reflexiv, transitiv und anti-symmetrisch :-)

## Berechnung:

Wir sammeln die Knoten entlang Pfaden auf mithilfe der Analyse:

$$\mathbb{P} = 2^{\text{Nodes}}, \quad \sqsubseteq = \supseteq$$

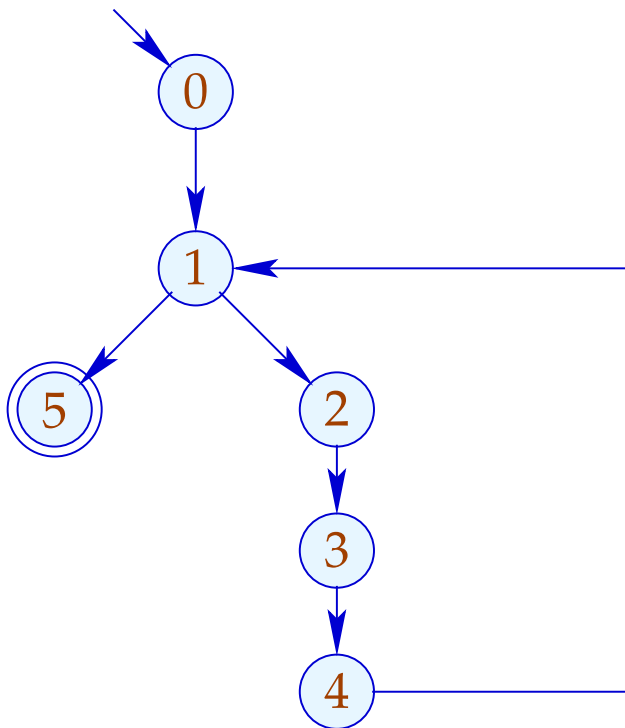
$$[[(\_, \_, v)]]^\# P = P \cup \{v\}$$

Dann ist die Menge  $\mathcal{P}[v]$  der Prädominatoren:

$$\mathcal{P}[v] = \bigcap \{ [[\pi]]^\# \{start\} \mid \pi : start \rightarrow^* v \}$$

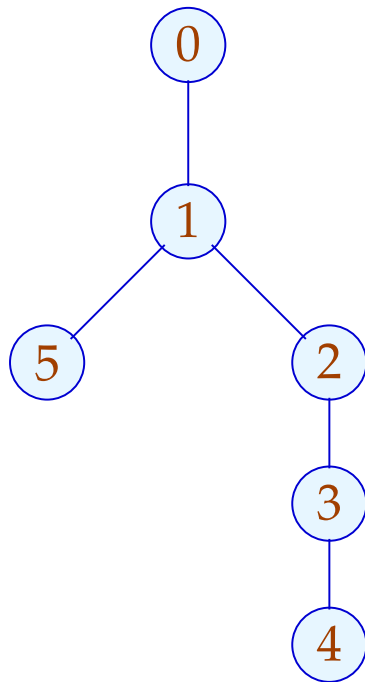
Da die  $\llbracket k \rrbracket^\#$  distributiv sind, können wir die  $\mathcal{P}[v]$  mithilfe von Fixpunkt-Iteration berechnen :-)

Beispiel:



	$\mathcal{P}$
0	$\{0\}$
1	$\{0, 1\}$
2	$\{0, 1, 2\}$
3	$\{0, 1, 2, 3\}$
4	$\{0, 1, 2, 3, 4\}$
5	$\{0, 1, 5\}$

Die partielle Ordnung " $\Rightarrow$ " im Beispiel:



	$\mathcal{P}$
0	$\{0\}$
1	$\{0, 1\}$
2	$\{0, 1, 2\}$
3	$\{0, 1, 2, 3\}$
4	$\{0, 1, 2, 3, 4\}$
5	$\{0, 1, 5\}$

Offenbar ist das Ergebnis ein Baum :-)

Tatsächlich gilt:

**Satz:**

Jeder Punkt  $v$  hat maximal einen unmittelbaren Prädominator.

**Beweis:**

**Annahme:**

Es gäbe  $u_1 \neq u_2$ , die  $v$  unmittelbar prädominieren.

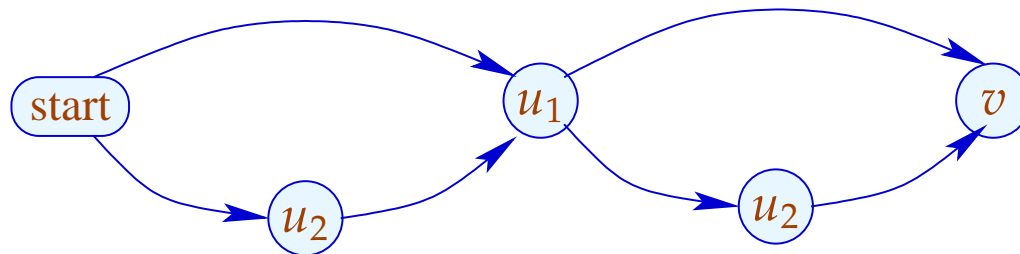
Gälte  $u_1 \Rightarrow u_2$ , wäre  $u_1$  nicht unmittelbar.

Folglich müssen  $u_1, u_2$  unvergleichbar sein :-)

Nun gilt für jedes  $\pi : \text{start} \rightarrow v$  :

$$\pi = \pi_1 \pi_2 \quad \text{mit} \quad \begin{aligned} \pi_1 &: \text{start} \rightarrow u_1 \\ \pi_2 &: u_1 \rightarrow v \end{aligned}$$

Sind  $u_1, u_2$  aber unvergleichbar, gibt es einen Pfad:  $\text{start} \rightarrow^* v$   
ohne  $u_2$  :

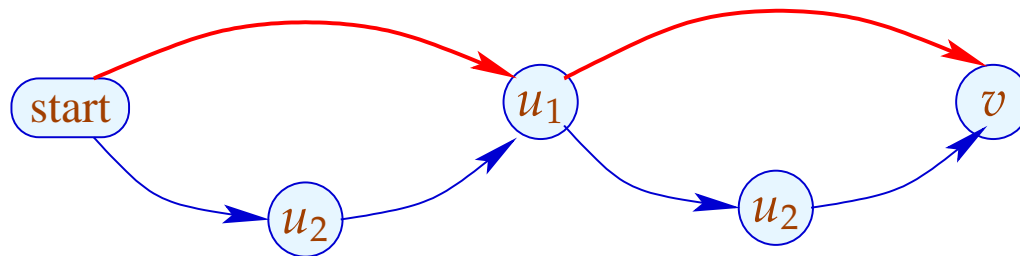




Nun gilt für jedes  $\pi : \text{start} \rightarrow v$  :

$$\pi = \pi_1 \pi_2 \quad \text{mit} \quad \begin{aligned} \pi_1 &: \text{start} \rightarrow u_1 \\ \pi_2 &: u_1 \rightarrow v \end{aligned}$$

Sind  $u_1, u_2$  aber unvergleichbar, gibt es einen Pfad:  $\text{start} \rightarrow^* v$   
ohne  $u_2$  :



## Beobachtung:

Der Schleifenkopf einer while-Schleife dominiert jeden Knoten des Rumpfs.

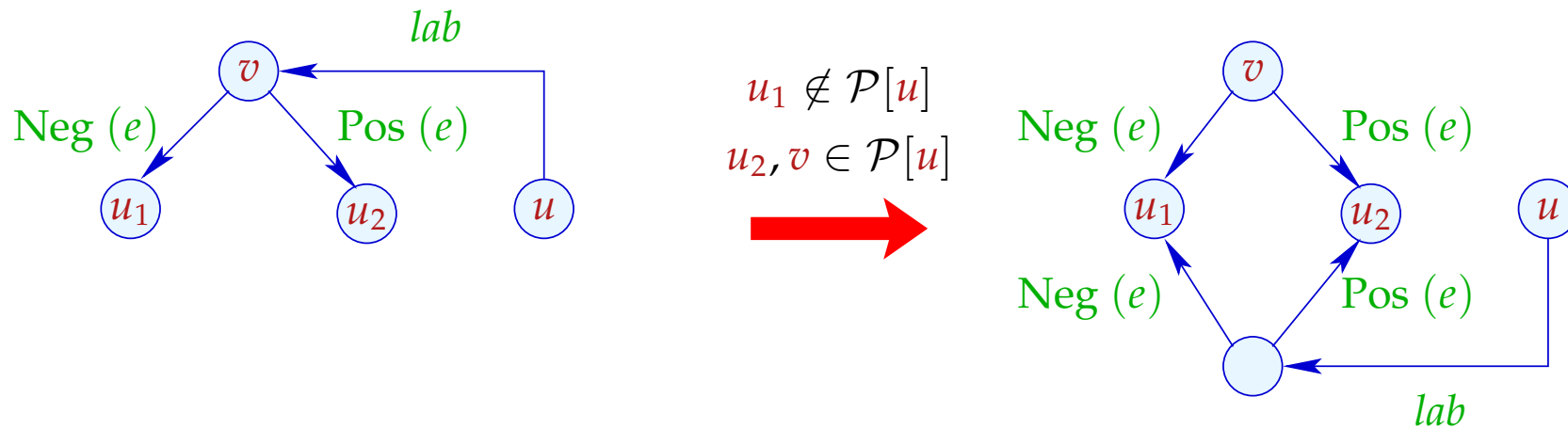
Einen Rücksprung vom Ende  $u$  zum Schleifenkopf  $v$  erkennt man daran, dass

$$v \in \mathcal{P}[u]$$

:-)

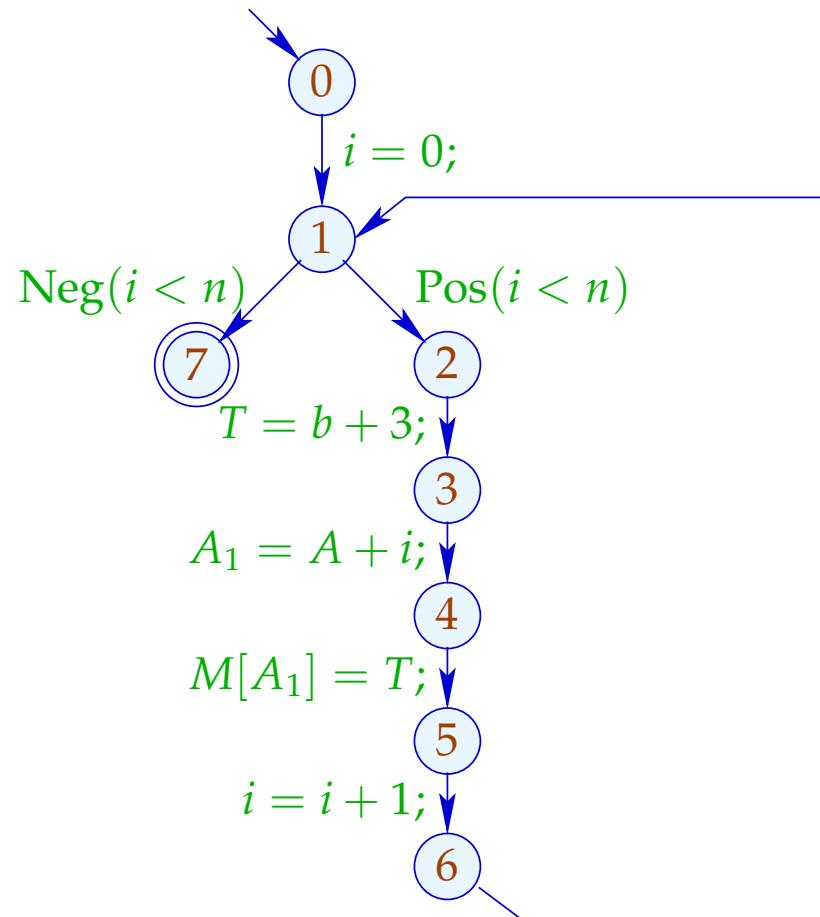
Damit definieren wir:

## Transformation 7:

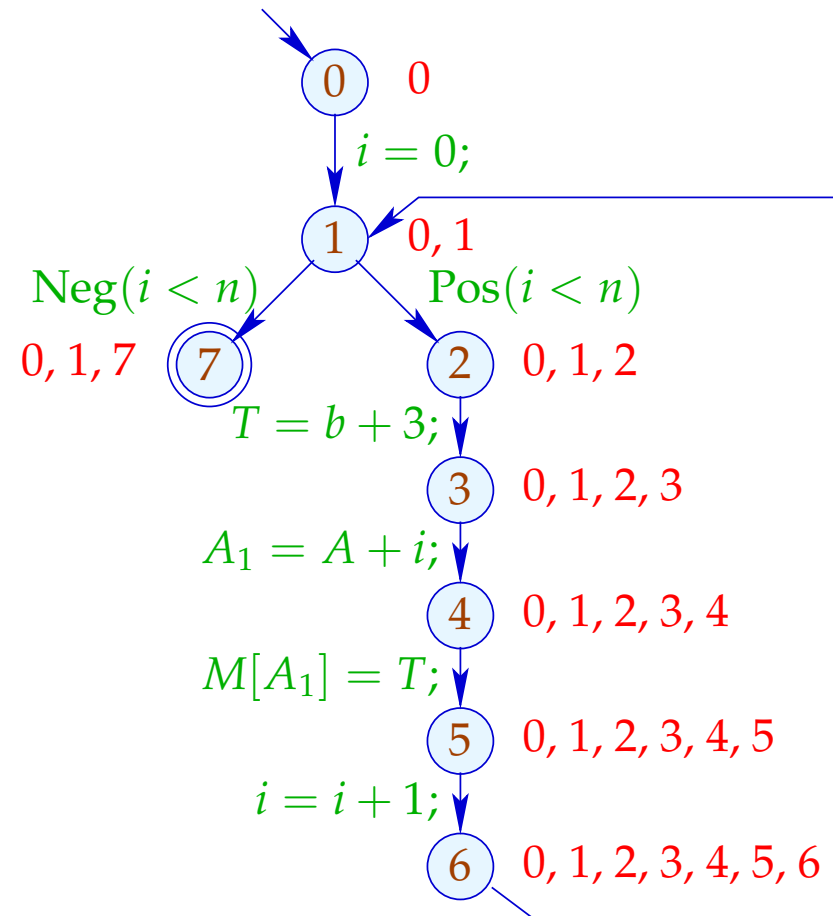


Wir duplizieren den Eintritts-Test an alle Rücksprung-Stellen :-)

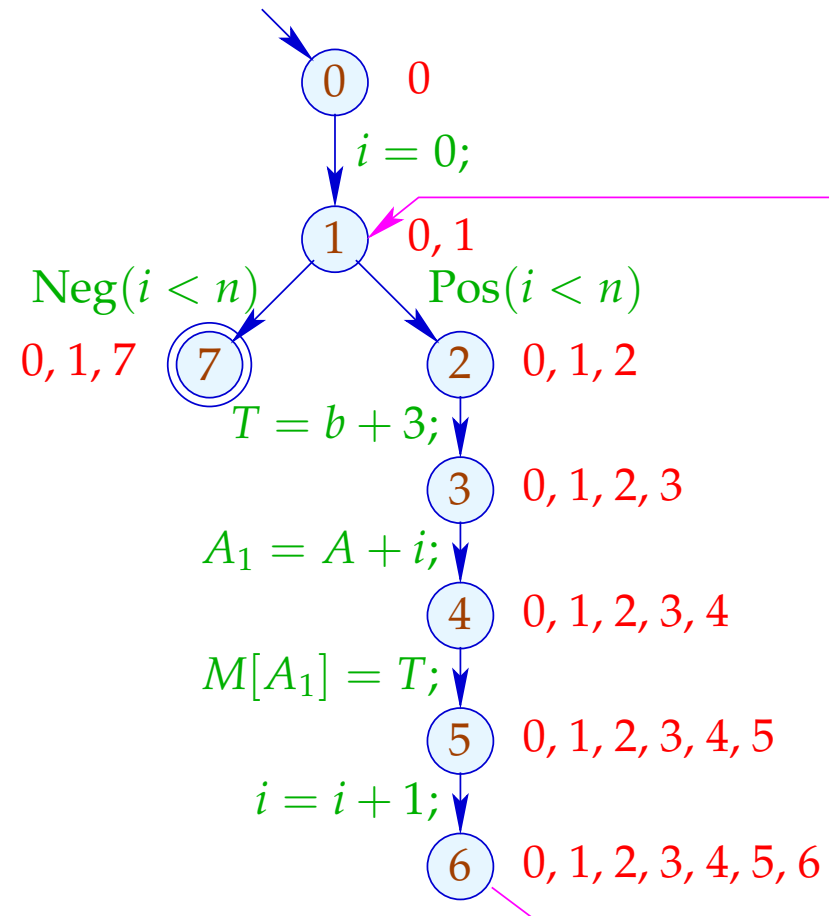
... im Beispiel:



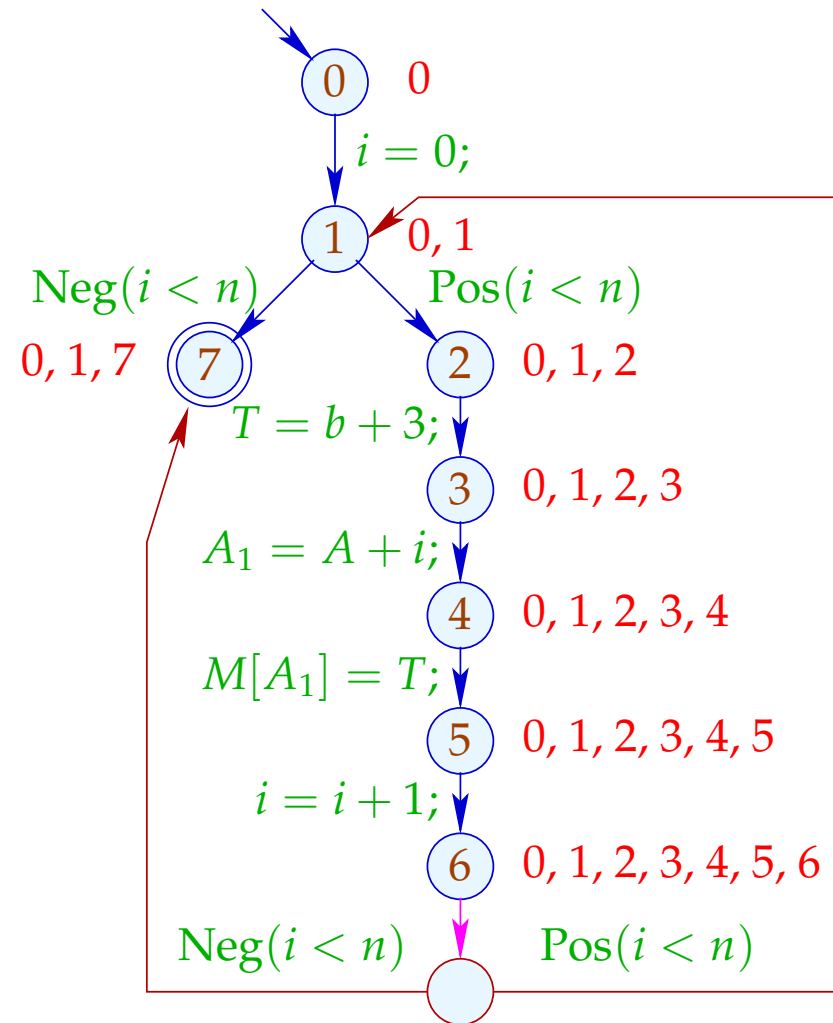
... im Beispiel:



... im Beispiel:

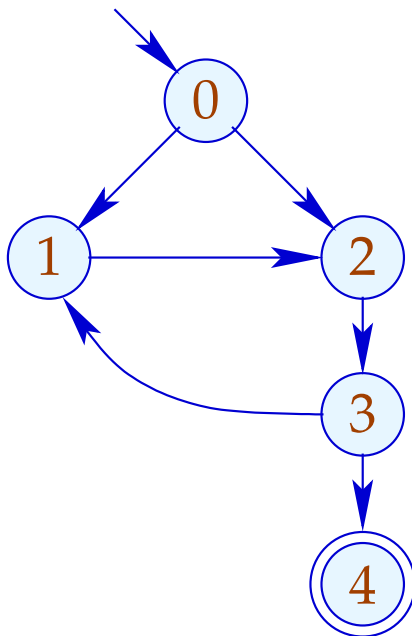


... im Beispiel:

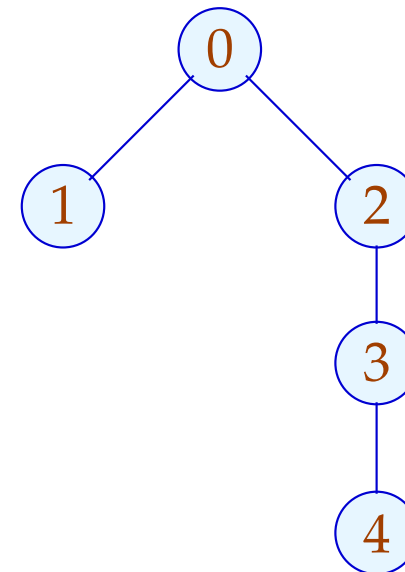


Achtung:

Es gibt **ungewöhnliche** Schleifen, die so nicht rotiert werden:

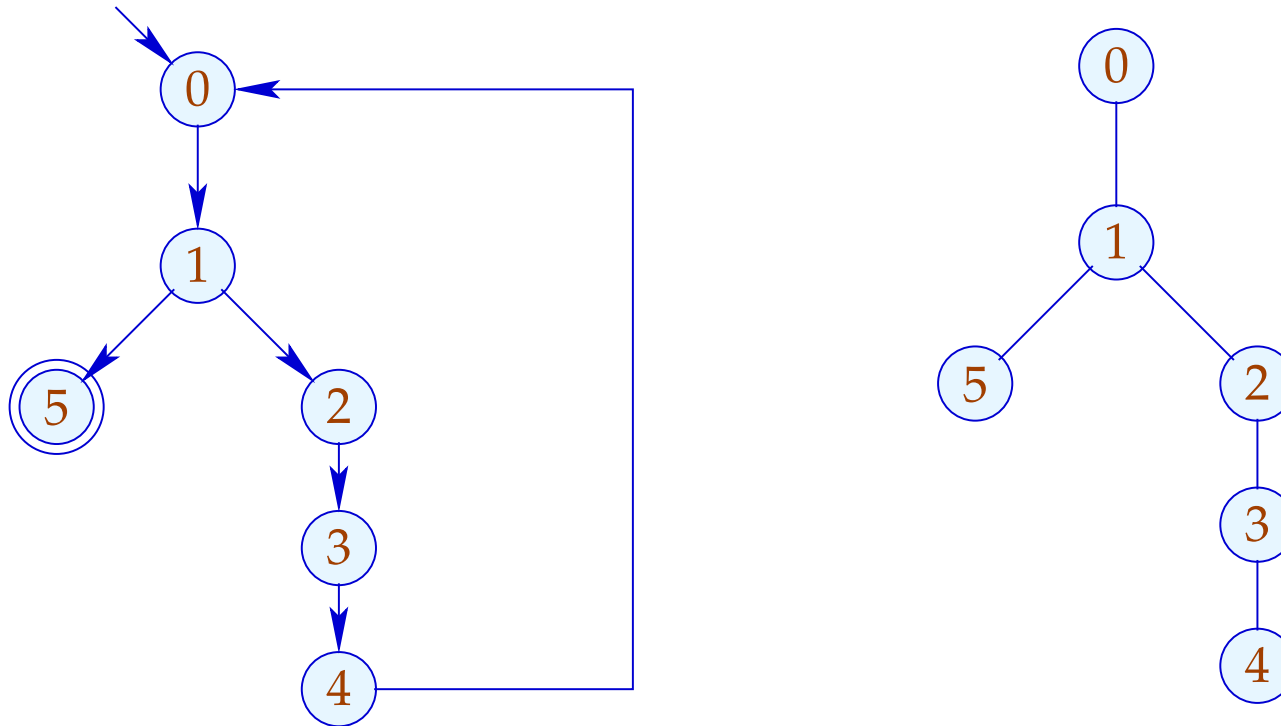


Prädominatoren:



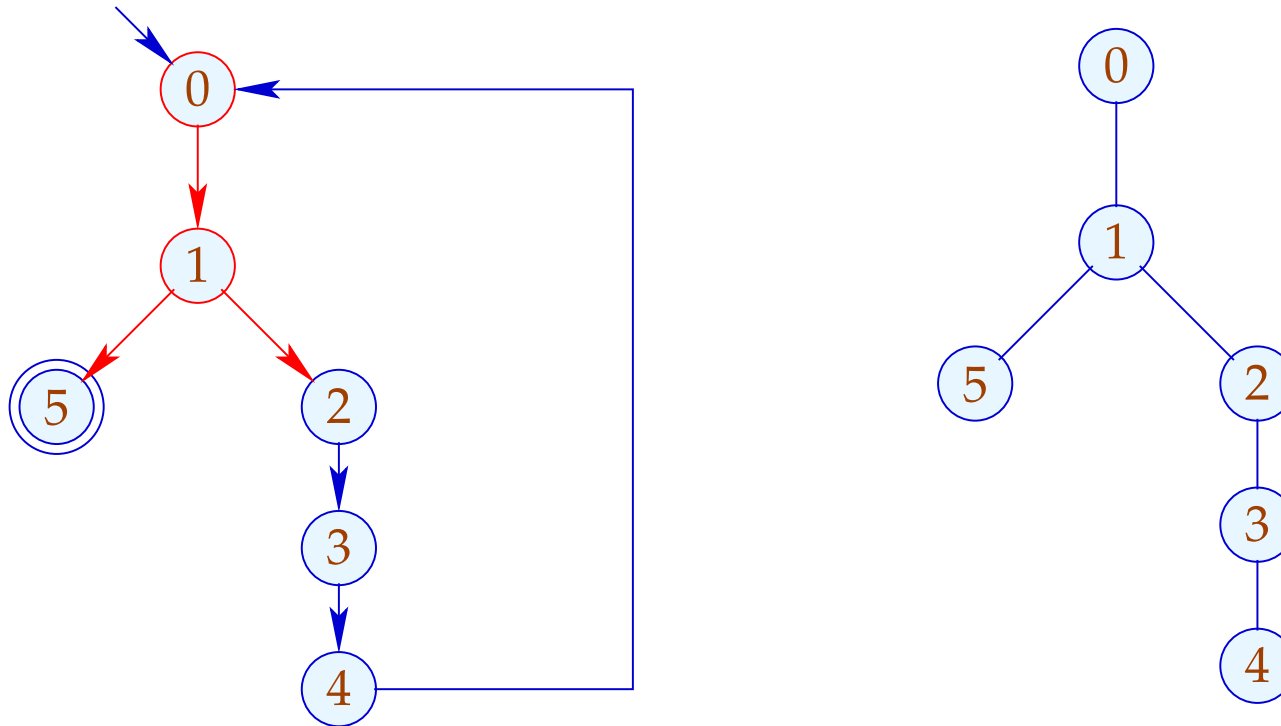


... leider aber auch **gewöhnliche**, die nicht rotiert werden:



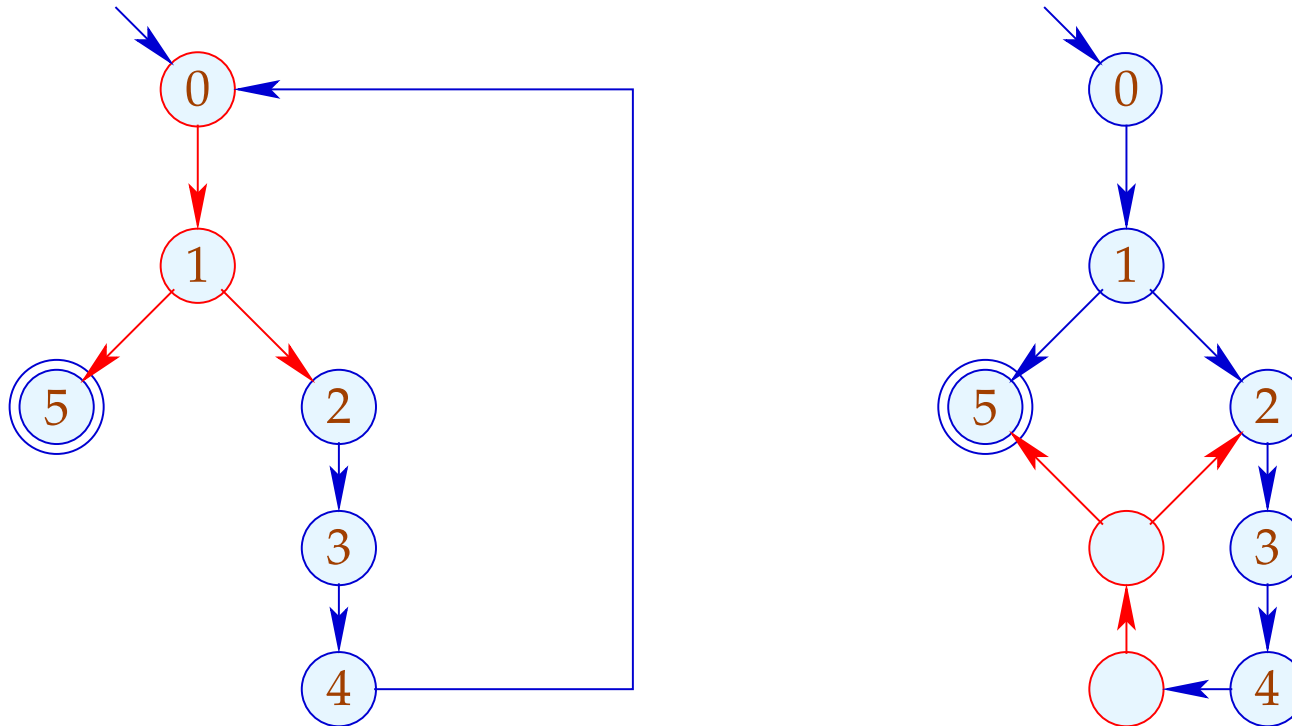
Hier müsste man den ganzen Pfad zwischen Rücksprung und Bedingung duplizieren :-)

... leider aber auch **gewöhnliche**, die nicht rotiert werden:



Hier müsste man den ganzen Pfad zwischen Rücksprung und Bedingung duplizieren :-)

... leider aber auch **gewöhnliche**, die nicht rotiert werden:



Hier müsste man den ganzen Pfad zwischen Rücksprung und Bedingung duplizieren :-)