Idee:

→ Greifen wir in f_i auf ein x_j zu, werten wir erst rekursiv aus. Dann fügen wir x_i zu $I[x_j]$ hinzu :-)

$$\begin{aligned} \mathsf{eval}\ x_i\ x_j \quad = \quad & \mathsf{solve}\, x_j; \\ & I[x_j] = I[x_j] \cup \{x_i\}; \\ & D[x_j]; \end{aligned}$$

→ Damit die Rekursion nicht unendlich absteigt, verwalten wir die Menge *Stable* von Variablen, für die solve den Wert nachschlägt :-)

Anfangs ist $\mathit{Stable} = \emptyset$ ...

Die Funktion solve :

$$
\begin{array}{lll}
\text{solve } x_i & = & \text{if } (x_i \notin \mathit{Stable}) \; \{ \\
& & \quad \mathit{Stable} = \mathit{Stable} \cup \{x_i\}; \\
& & \quad t = f_i \; (\text{eval } \; x_i); \\
& & \quad \text{if } (t \not\sqsubseteq D[x_i]) \; \{ \\
& & \qquad W = I[x_i]; \quad I[x_i] = \emptyset; \\
& & \qquad D[x_i] = D[x_i] \sqcup t; \\
& & \qquad \mathit{Stable} = \mathit{Stable} \backslash W; \\
& & \qquad \text{app } \; \text{solve } \; W; \\
& & \quad \} \\
& & \} \\
\end{array}
$$

Helmut Seidl, TU München ;-)

Beispiel:

Betrachte unser Standard-Beispiel:

$$\begin{aligned} x_1 &\supseteq \{a\} \cup x_3 \\ x_2 &\supseteq x_3 \cap \{a, b\} \\ x_3 &\supseteq x_1 \cup \{c\} \end{aligned}$$

Dann sieht ein Trace des Fixpunkt-Algorithmus etwa so aus:

solve x_2 | eval x_2 x_3 | solve x_3 | eval x_3 x_1 | solve x_1 | eval x_1 x_3 | solve x_3

stable!

$I[x_3] = \{x_1\}$

$\Rightarrow \quad \emptyset$

$D[x_1] = \{a\}$

$I[x_1] = \{x_3\}$

$\Rightarrow \quad \{a\}$

$D[x_3] = \{a, c\}$

$I[x_3] = \emptyset$

solve x_1 | eval x_1 x_3 | solve x_3

stable!

$I[x_3] = \{x_1\}$

$\Rightarrow \quad \{a, c\}$

$D[x_1] = \{a, c\}$

$I[x_1] = \emptyset$

solve x_3 | eval x_3 x_1 | solve x_1

stable!

$I[x_1] = \{x_3\}$

$\Rightarrow \quad \{a, c\}$

ok

$I[x_3] = \{x_1, x_2\}$

$\Rightarrow \quad \{a, c\}$

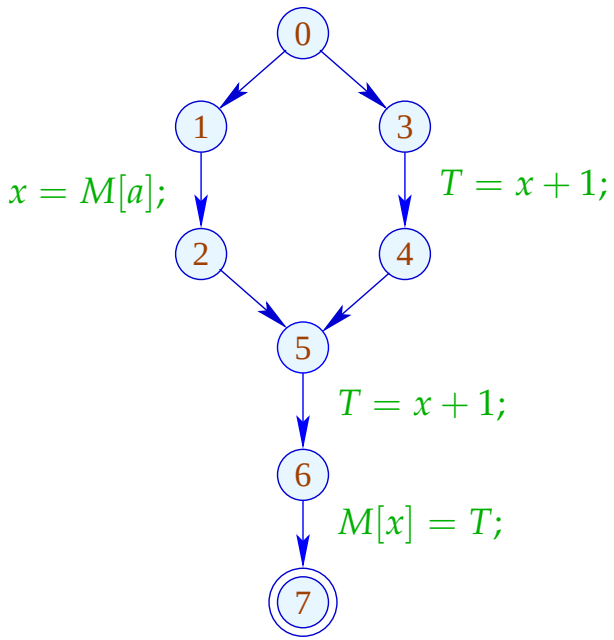
$D[x_2] = \{a\}$

- $\rightarrow$ Die Auswertung startet mit einer interessierenden Variable x_i (z.B. dem Wert für *stop*)
- $\rightarrow$ Es werden automatisch alle Variablen ausgewertet, die x_i beeinflussen :-)
- $\rightarrow$ Die Anzahl der Auswertungen ist i.a. kleiner als die bei normaler Iteration ;-)
- $\rightarrow$ Der Algorithmus ist komplizierter, benötigt aber keine Vorberechnung der Variablen-Abhängigkeiten :-))
- $\rightarrow$ Er funktioniert auch, wenn die Variablen-Abhängigkeiten sich während der Iteration ändern !!!

$\Longrightarrow$ interprozedurale Analyse

1.7 Beseitigung partieller Redundanzen

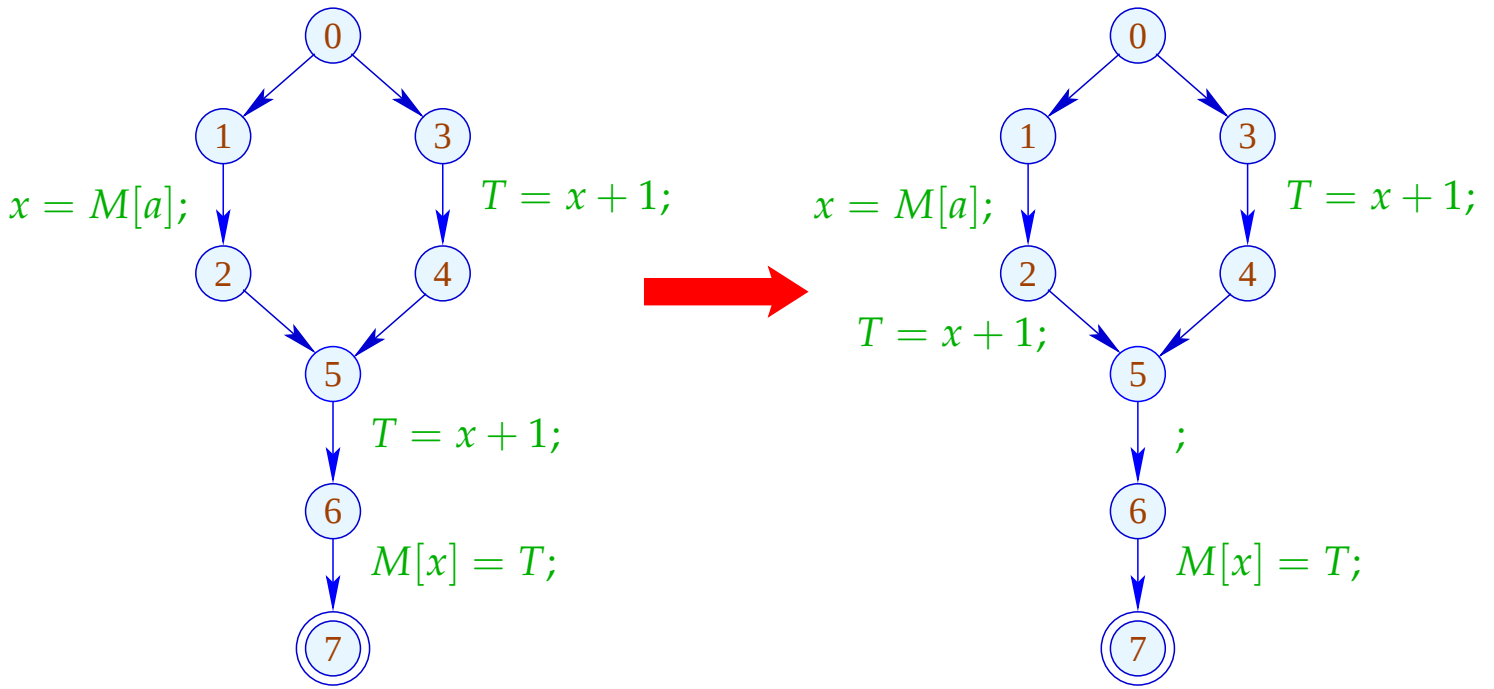
Beispiel:



// $e = x + 1$ wird auf jedem Pfad ausgewertet ...

// leider auf einem Pfad sogar zweimal :-(

Ziel:



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.

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

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

// Rückwärtsanalyse!

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

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

// Rückwärtsanalyse!

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

Entsprechend benötigen wir:

$$\mathcal{B}[u] \;=\; \bigcap \{ [\![\pi]\!]^\sharp \, \emptyset \mid \pi : u \to^* \mathit{stop} \}$$

wobei für $\pi = k_1 \ldots k_m$:

$$[\![\pi]\!]^\sharp \;=\; [\![k_1]\!]^\sharp \circ \ldots \circ [\![k_m]\!]^\sharp$$

Unser vollständiger Verband ist:

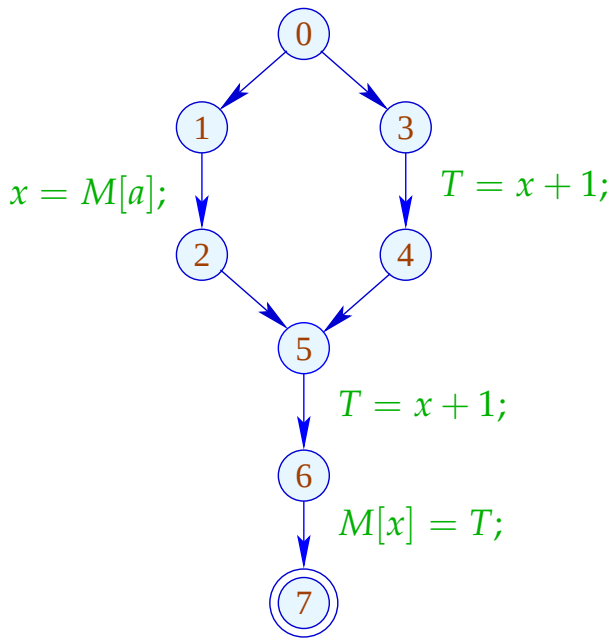
$$\mathbb{B} = 2^{Expr \setminus Vars} \qquad \text{mit} \quad \sqsubseteq \; = \; \supseteq$$

Der Effekt $[\![k]\!]^\sharp$ einer Kante $k = (u, lab, v)$ hängt nur von lab ab, d.h. $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ wobei:

$$\begin{array}{lcl}
[\![;]\!]^\sharp\, B & = & B \\
[\![Pos(e)]\!]^\sharp\, B & = & [\![Neg(e)]\!]^\sharp\, B \quad = \quad B \cup \{e\} \\
[\![T_e = e;]\!]^\sharp\, B & = & (B \setminus Expr_{T_e}) \cup \{e\} \\
[\![x = T;]\!]^\sharp\, B & = & B \setminus Expr_x \\
[\![x = M[R];]\!]^\sharp\, B & = & B \setminus Expr_x \\
[\![M[R] = x;]\!]^\sharp\, B & = & B
\end{array}$$

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 unahängig, d.h. $T_{e_1} \notin Vars(e_2)$:-)
- 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 $T_{e_2} \notin \mathit{Vars}\,(e_1)$:-)

Beweis der Anordbarkeit:

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

$\rightarrow$ Induktion über die Länge von π.

$\boxed{\pi = \epsilon}$ $[\![\pi]\!]^\sharp\,\emptyset = [\![\epsilon]\!]^\sharp\,\emptyset = \emptyset$:-)

$\boxed{\pi = k\ \pi'}$ 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:

  $\rightarrow$ $e \in \mathcal{B}[v]$ sowie

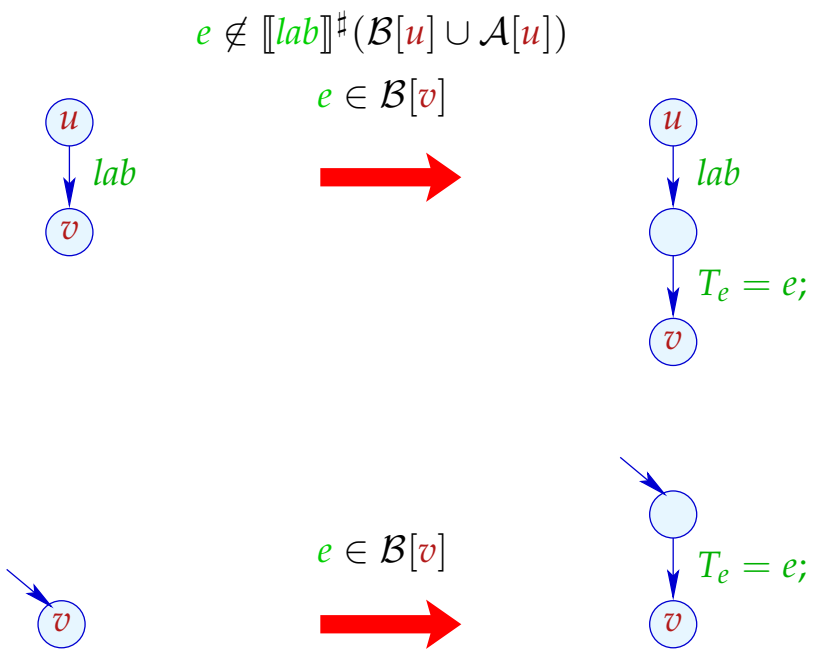
  $\rightarrow$ $e \notin [\![lab]\!]^\sharp(\mathcal{A}[u])$ (nicht verfügbar entlang k) und

  $\rightarrow$ $e \notin [\![lab]\!]^\sharp(\mathcal{B}[u])$ (auch nicht nach Transformation)

- Weil alle $e \in \mathcal{B}[v]$ anordbar sind, betrachten wir die Transformation für jedes e gesondert:

Transformation 6.1:

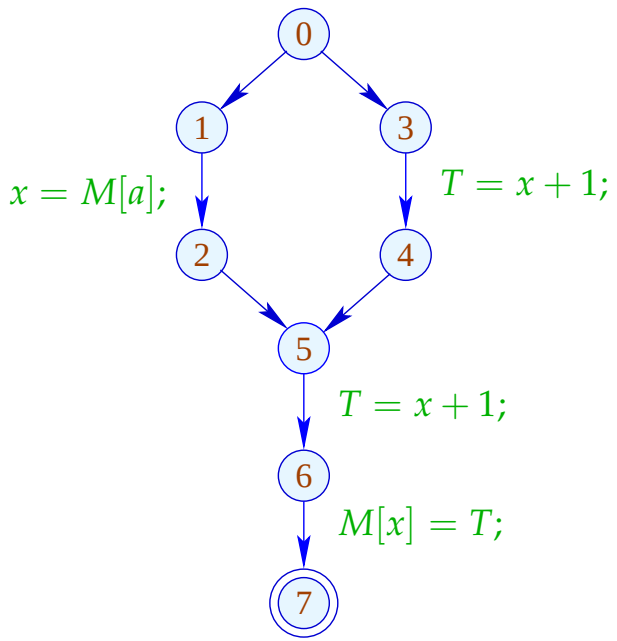
Transformation 6.2:



Bernhard Steffen, Dortmund

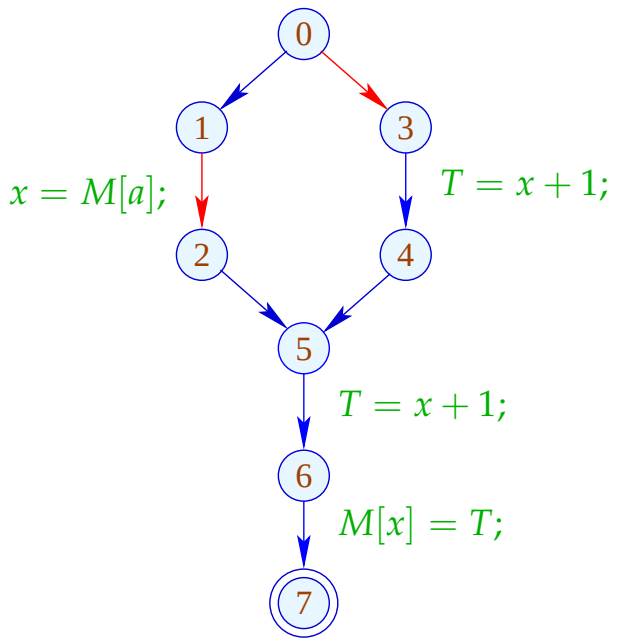
Jens Knoop, Wien

Im Beispiel:



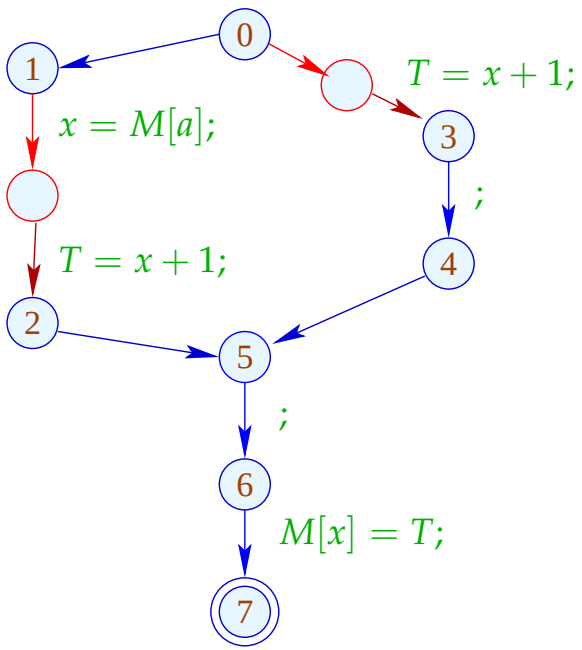
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x+1\}$
3	$\emptyset$	$\{x+1\}$
4	$\{x+1\}$	$\{x+1\}$
5	$\emptyset$	$\{x+1\}$
6	$\{x+1\}$	$\emptyset$
7	$\{x+1\}$	$\emptyset$

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:



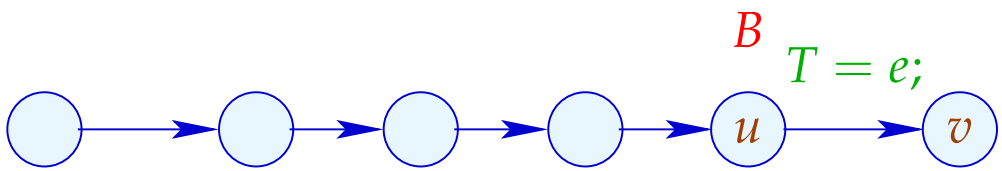
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x+1\}$
3	$\emptyset$	$\{x+1\}$
4	$\{x+1\}$	$\{x+1\}$
5	$\emptyset$	$\{x+1\}$
6	$\{x+1\}$	$\emptyset$
7	$\{x+1\}$	$\emptyset$

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:

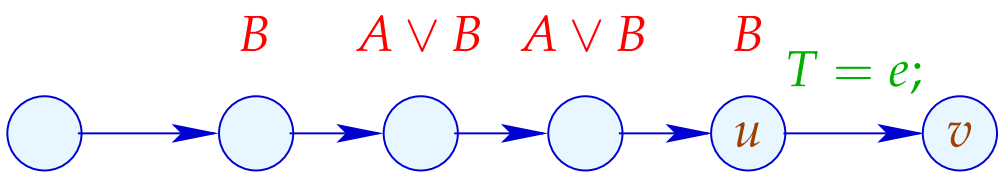


Zur Korrektheit:

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:

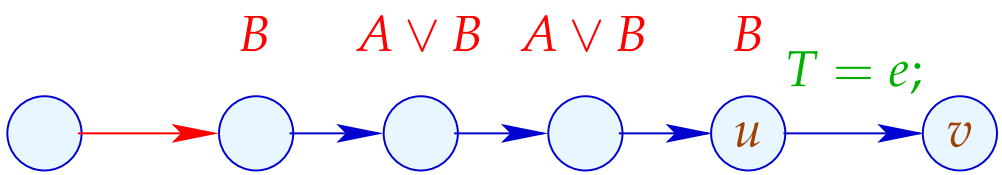


Zur Korrektheit:

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:

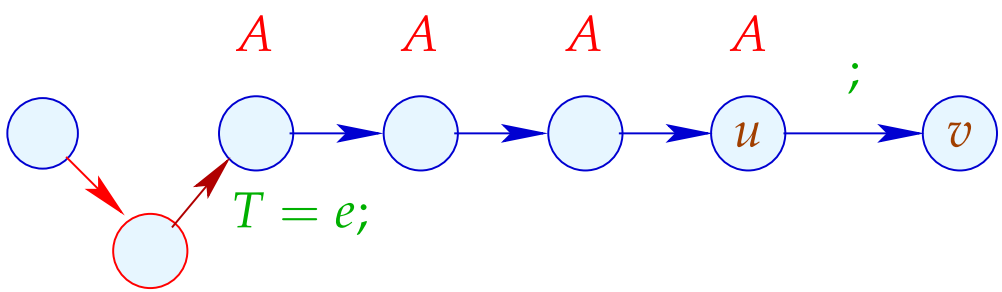


Zur Korrektheit:

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 :-)

  $\Longrightarrow$ Korrektheit der Transformation

- Jedem $T = e;$, das wir in einen Pfad einfügen, entspricht ein $T = e;$, das wir gestrichen haben :-))

  $\Longrightarrow$ Nicht-Verschlechterung der Transformation

1.8 Anwendung: Schleifen-invarianter Code

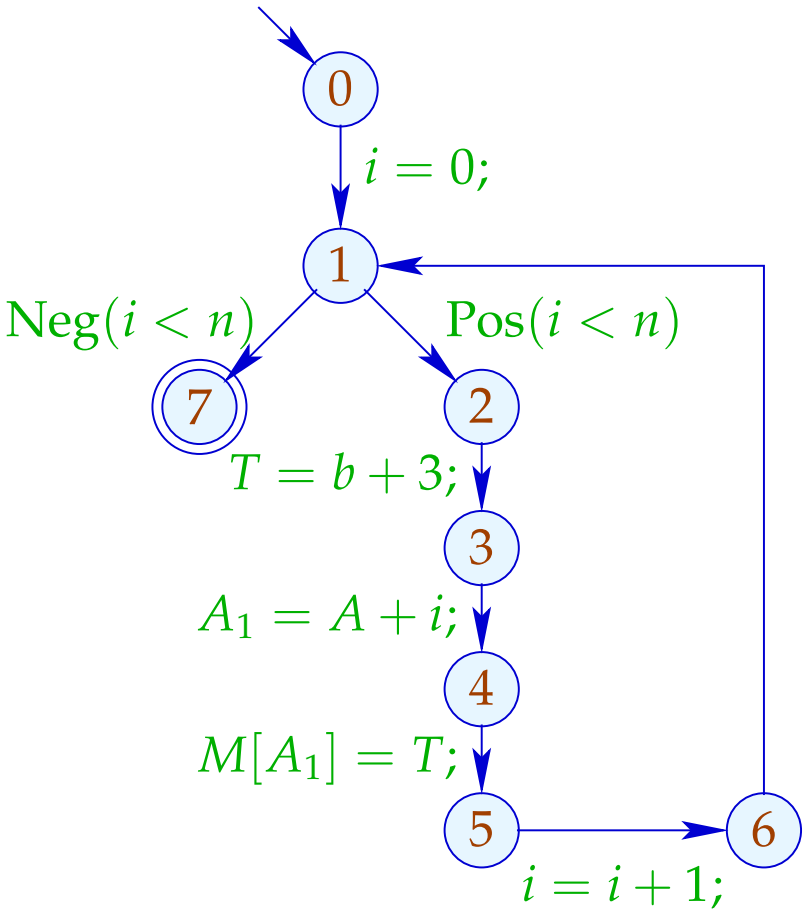
Beispiel:

$$\begin{aligned} &\text{for } (i = 0; i < n; i{+}{+}) \\ &\qquad a[i] = b + 3; \end{aligned}$$

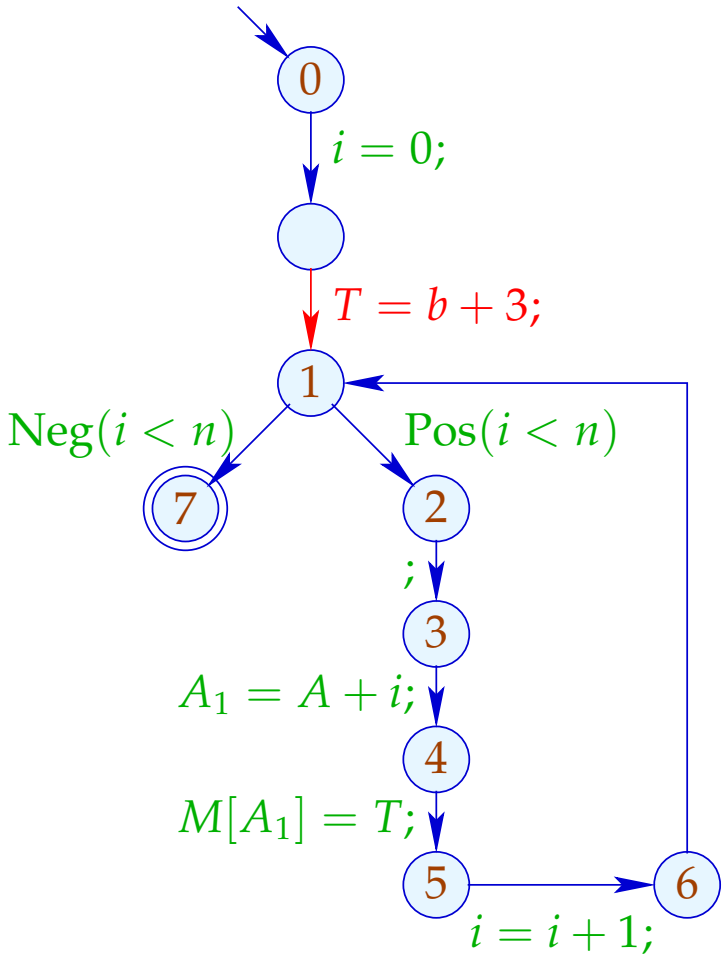
// Der Ausdruck $b + 3$ wird in jeder Iteration berechnet :-(

// Das wollen wir vermeiden :-)

Der Kontrollfluss-Graph:

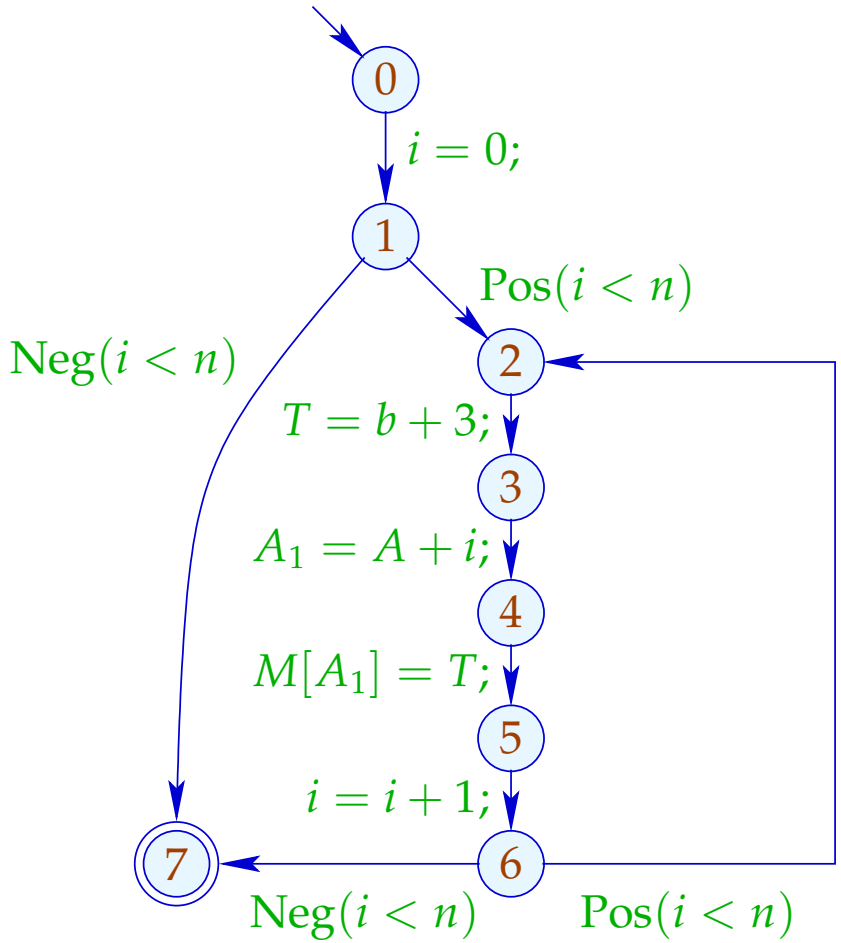


Achtung: $T = b + 3;$ darf nicht vor der Schleife stehen :

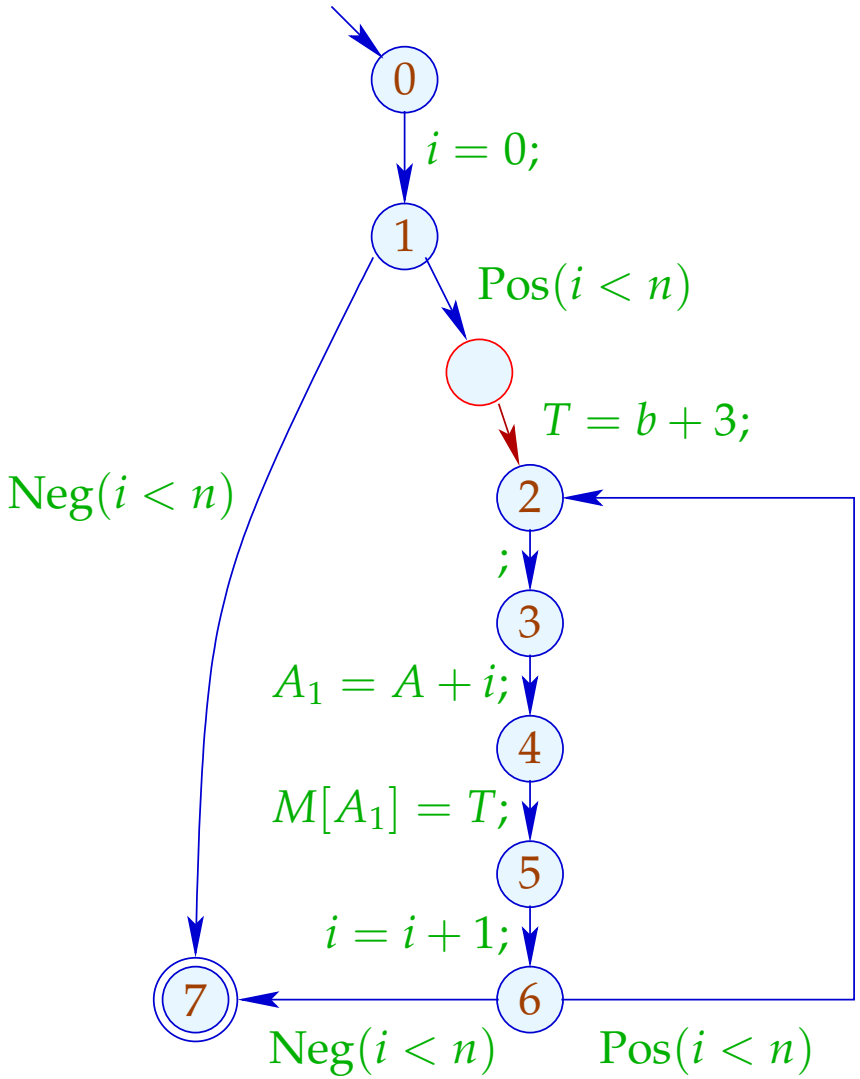


$\Longrightarrow$ Es gibt keinen guten Platz für $T = b + 3;$:-(

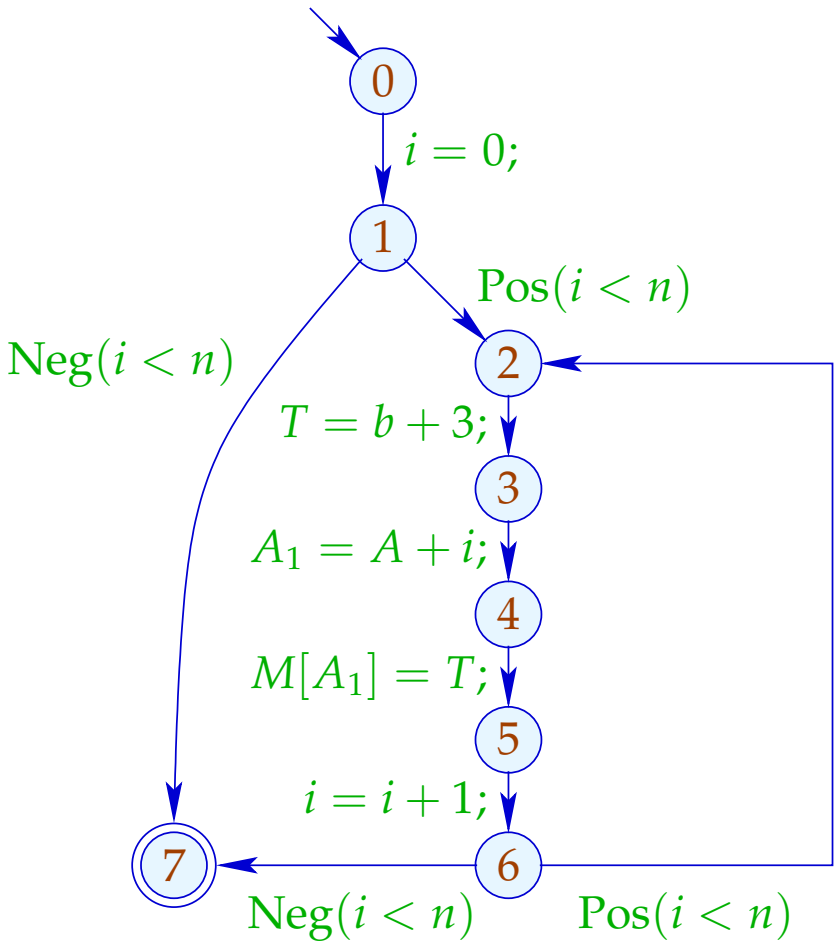
Idee: Transformiere in eine do-while-Schleife ...



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

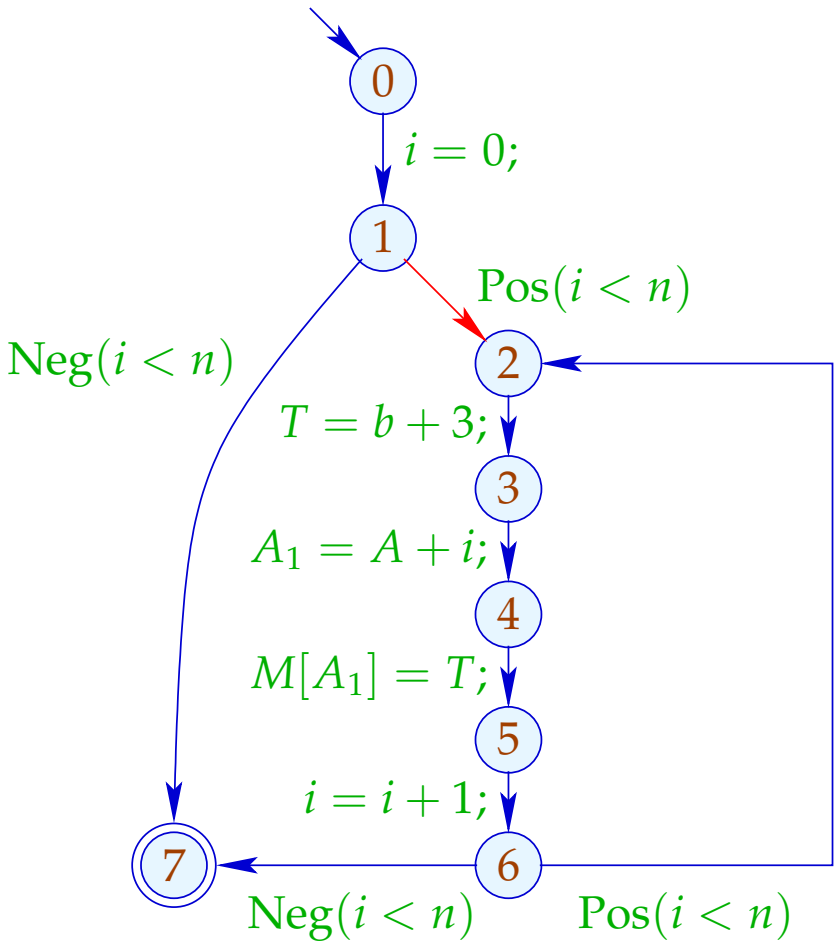


Anwendung von T6 (PRE) :



	$\mathcal{A}$	$\mathcal{B}$
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-Scheifen :-(
- Um andere Scheifen zu optimieren, wandeln wir sie in do-while-Scheifen um:

$$\texttt{while}\ (b)\ stmt \quad \Longrightarrow \quad \texttt{if}\ (b)$$
$$\texttt{do}\ stmt$$
$$\texttt{while}\ (b);$$

$$\Longrightarrow \quad \text{Schleifen-Rotation}$$

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 :-)