Wir zeigen, dass $\mathcal{I}^*$ alle Ungleichungen erfüllt :-))

(1) Für *start* zeigen wir:

$$\begin{array}{lll} \mathcal{I}^*[\mathit{start}] & = & \bigsqcup\{[\![\pi]\!]^\sharp\, d_0 \mid \pi : \mathit{start} \rightarrow^* \mathit{start}\} \\ & \sqsupseteq & [\![\epsilon]\!]^\sharp\, d_0 \\ & \sqsupseteq & d_0 \qquad \text{:-)} \end{array}$$

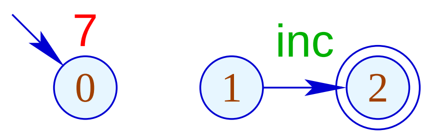
(2) Für jedes $k = (u, \_, v)$ zeigen wir:

$$\begin{array}{lll} \mathcal{I}^*[v] & = & \bigsqcup\{[\![\pi]\!]^\sharp\, d_0 \mid \pi : \mathit{start} \rightarrow^* v\} \\ & \sqsupseteq & \bigsqcup\{[\![\pi' k]\!]^\sharp\, d_0 \mid \pi' : \mathit{start} \rightarrow^* u\} \\ & = & \bigsqcup\{[\![k]\!]^\sharp\, ([\![\pi']\!]^\sharp\, d_0) \mid \pi' : \mathit{start} \rightarrow^* u\} \\ & = & [\![k]\!]^\sharp\, (\bigsqcup\{[\![\pi']\!]^\sharp\, d_0 \mid \pi' : \mathit{start} \rightarrow^* u\}) \\ & = & [\![k]\!]^\sharp\, (\mathcal{I}^*[u]) \end{array}$$

da $\{\pi' \mid \pi' : \mathit{start} \rightarrow^* u\}$ nicht-leer ist :-)

Achtung:

- Auf die Erreichbarkeit aller Programm-Punkt können wir nicht verzichten. Betrachte:

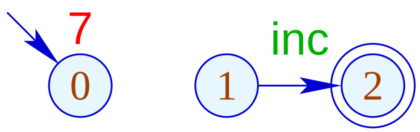


wobei $\quad \mathbb{D} = \mathbb{N} \cup \{\infty\}$

$$=$$

Achtung:

- Auf die Erreichbarkeit aller Programm-Punkt können wir nicht verzichten. Betrachte:



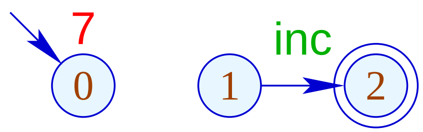
wobei $\mathbb{D} = \mathbb{N} \cup \{\infty\}$

Dann ist:

$$\begin{array}{lllll} \mathcal{I}[2] & = & \mathsf{inc}\, 0 & = & 1 \\ \mathcal{I}^*[2] & = & \bigsqcup \emptyset & = & 0 \end{array}$$

Achtung:

- Auf die Erreichbarkeit aller Programm-Punkt können wir nicht verzichten. Betrachte:



wobei $\mathbb{D} = \mathbb{N} \cup \{\infty\}$

Dann ist:

$$\begin{array}{lllll} \mathcal{I}[2] & = & \mathsf{inc}\, 0 & = & 1 \\ \mathcal{I}^*[2] & = & \bigsqcup \emptyset & = & 0 \end{array}$$

- Unerreichbare Programmpunkte können wir aber stets wegwerfen :-)

Zusammenfassung und Anwendung:

$\longrightarrow$ Die Kanteneffekte der Analyse zur Verfügbarkeit von Ausdrücken sind distributiv:

$$
\begin{array}{lll}
(a \cup (x_1 \cap x_2)) \backslash b & = & ((a \cup x_1) \cap (a \cup x_2)) \backslash b \\
& = & ((a \cup x_1) \backslash b) \cap ((a \cup x_2) \backslash b)
\end{array}
$$

Zusammenfassung und Anwendung:

$\longrightarrow$ Die Kanteneffekte der Analyse zur Verfügbarkeit von Ausdrücken sind distributiv:

$$\begin{aligned}(a \cup (x_1 \cap x_2)) \backslash b &= ((a \cup x_1) \cap (a \cup x_2)) \backslash b \\ &= ((a \cup x_1) \backslash b) \cap ((a \cup x_2) \backslash b)\end{aligned}$$

$\longrightarrow$ Sind alle Kanteneffekte distributiv, lässt sich der MOP mithilfe des Constraint-Systems und RR-Iteration ausrechnen :-)

Zusammenfassung und Anwendung:

→ Die Kanteneffekte der Analyse zur Verfügbarkeit von Ausdrücken sind distributiv:

$$\begin{aligned}(a \cup (x_1 \cap x_2)) \backslash b &= ((a \cup x_1) \cap (a \cup x_2)) \backslash b \\ &= ((a \cup x_1) \backslash b) \cap ((a \cup x_2) \backslash b)\end{aligned}$$

→ Sind alle Kanteneffekte distributiv, lässt sich der MOP mithilfe des Constraint-Systems und RR-Iteration ausrechnen :-)

→ Sind nicht alle Kanteneffekte distributiv, lässt sich eine sichere obere Schranke für den MOP mithilfe des Constraint-Systems und RR-Iteration berechnen :-)

1.2 Beseitigung überflüssiger Zuweisungen

Beispiel:

$$
\begin{array}{lll}
1: & & x = y + 2; \\
2: & & y = 5; \\
3: & & x = y + 3;
\end{array}
$$

Der Wert von x an den Programmpunkten $1, 2$ wird überschrieben, bevor er benutzt werden kann.

Die Variable x nennen wir deshalb an diesen Programmpunkten tot :-)

Beachte:

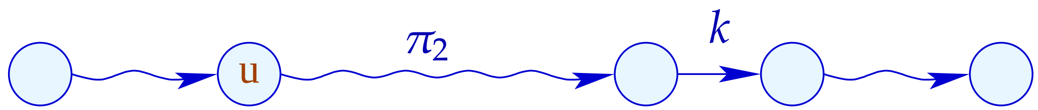
$\longrightarrow$ Zuweisungen an tote Variablen können wir uns schenken ;-)

$\longrightarrow$ Solche Ineffizienzen können u.a. durch andere Transformationen hervorgerufen werden.

Formale Definition:

Die Variable x heißt lebendig an u entlang des Pfads π, falls sich π zerlegen lässt in $\pi = \pi_1\,\pi_2\,k\,\pi_3$ so dass gilt:

- π_1 erreicht u;
- k ist eine Benutzung von x;
- π_2 enthält keine Überschreibung von x.

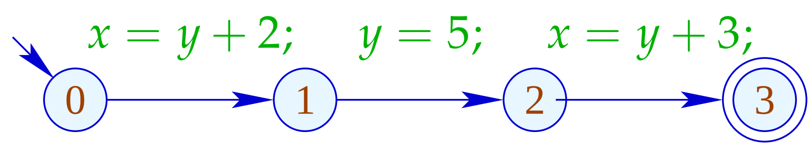


Die Menge der an einer Kante $k = (\_, lab, \_)$ benutzten bzw. überschriebenen Variablen ist dabei gegeben durch:

lab	benutzt	überschrieben
$;$	$\emptyset$	$\emptyset$
$\text{Pos}\,(e)$	$\mathit{Vars}\,(e)$	$\emptyset$
$\text{Neg}\,(e)$	$\mathit{Vars}\,(e)$	$\emptyset$
$R = e;$	$\mathit{Vars}\,(e)$	$\{R\}$
$R_1 = M[R_2];$	$\{R_2\}$	$\{R_1\}$
$M[R_1] = R_2;$	$\{R_1, R_2\}$	$\emptyset$

Eine Variable x , die nicht lebendig an u entlang π ist, heißt tot an u entlang π.

Beispiel:



Wir bemerken:

	lebendig	tot
0	$\{y\}$	$\{x\}$
1	$\emptyset$	$\{x, y\}$
2	$\{y\}$	$\{x\}$
3	$\emptyset$	$\{x, y\}$

Die Variable x ist lebendig an u falls x lebendig ist an u entlang irgend eines Pfads. Andernfalls ist x tot an u .

Die Variable x ist lebendig an u falls x lebendig ist an u entlang irgend eines Pfads. Andernfalls ist x tot an u .

Frage:

Wie berechnet man für jedes u die Menge der dort lebendigen/toten Variablen ???

Die Variable x ist lebendig an u falls x lebendig ist an u entlang irgend eines Pfads. Andernfalls ist x tot an u .

Frage:

Wie berechnet man für jedes u die Menge der dort lebendigen/toten Variablen ???

Idee:

Definiere für jede Kante $k = (u, \_, v)$ eine Funktion $[\![k]\!]^\sharp$, die die Menge der an v lebendigen Variablen in die Menge der an u lebendigen Variablen transformiert ...

Sei $\mathbb{L} = 2^{Vars}$.

Für $k = (\_, lab, \_)$ definieren wir $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ durch:

$$
\begin{array}{lcl}
[\![;]\!]^\sharp\, L & = & L \\
[\![\mathrm{Pos}(e)]\!]^\sharp\, L & = & [\![\mathrm{Neg}(e)]\!]^\sharp\, L \quad = \quad L \cup Vars(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![R_1 = M[R_2];]\!]^\sharp\, L & = & (L \backslash \{R_1\}) \cup \{R_2\} \\
[\![M[R_1] = R_2;]\!]^\sharp\, L & = & L \cup \{R_1, R_2\}
\end{array}
$$

Sei $\mathbb{L} = 2^{Vars}$.

Für $k = (\_, lab, \_)$ definieren wir $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ durch:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, L & = & L \\
[\![\text{Pos}(e)]\!]^\sharp\, L & = & [\![\text{Neg}(e)]\!]^\sharp\, L \quad = \quad L \cup Vars(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![R_1 = M[R_2];]\!]^\sharp\, L & = & (L \backslash \{R_1\}) \cup \{R_2\} \\
[\![M[R_1] = R_2;]\!]^\sharp\, L & = & L \cup \{R_1, R_2\}
\end{array}
$$

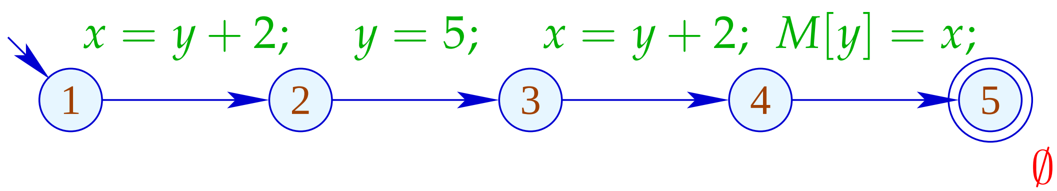
$[\![k]\!]^\sharp$ können wir wieder zu Effekten $[\![\pi]\!]^\sharp$ ganzer Pfade $\pi = k_1 \ldots k_r$ fortsetzen durch:

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

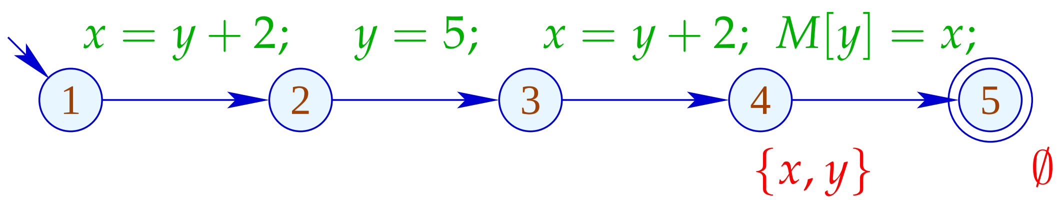
Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)

$$\begin{tikzcd} 1 \arrow[r, "x = y+2;"] & 2 \arrow[r, "y = 5;"] & 3 \arrow[r, "x = y+2;"] & 4 \arrow[r, "M[y] = x;"] & 5 \end{tikzcd}$$

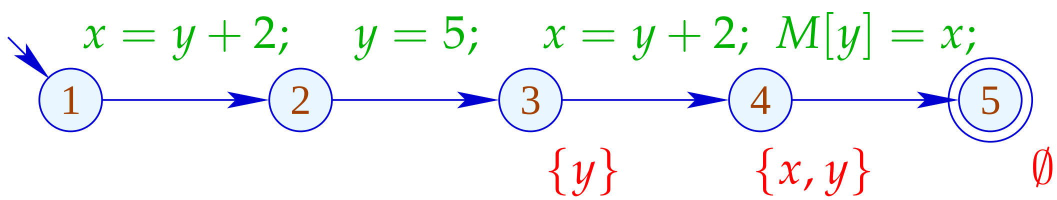
Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)



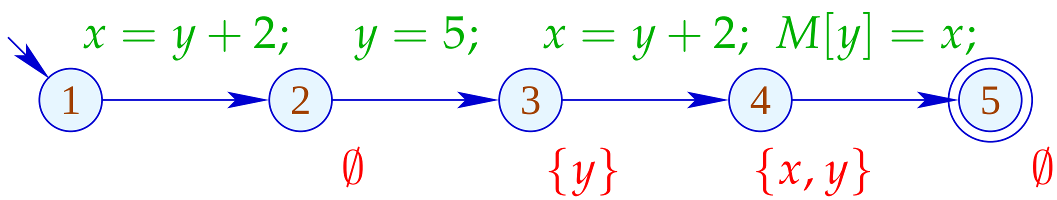
Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)



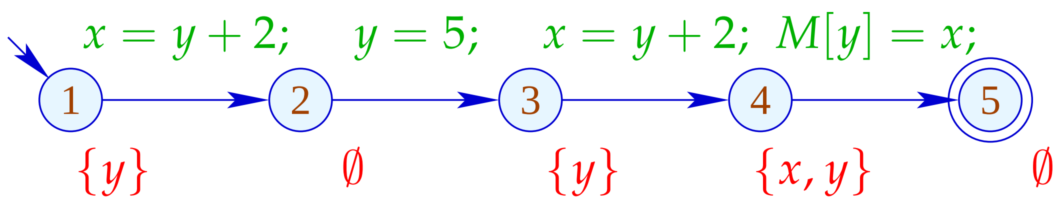
Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)



Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)



Wir vergewissern uns, dass diese Definitionen vernünftig sind :-)



Die Menge der an u lebendigen Variablen ist dann:

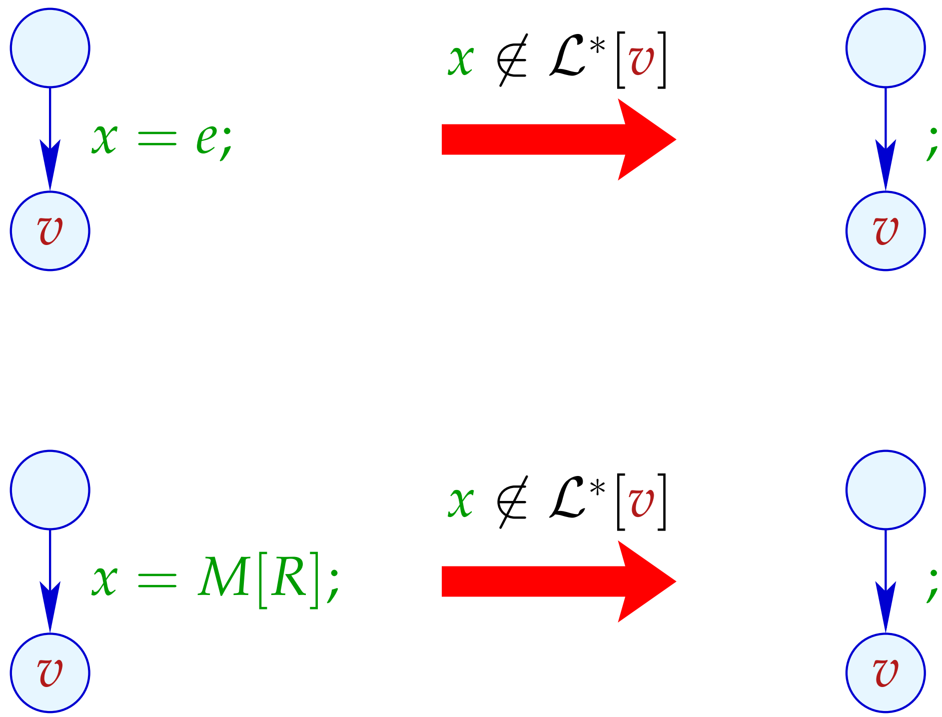
$$\mathcal{L}^*[u] \;=\; \bigcup\{[\![\pi]\!]^\sharp\,\emptyset \mid \pi : u \to^* \mathit{stop}\}$$

... in Worten:

- Die Pfade starten in u :-)
- x ist lebendig, wenn es nur entlang irgend eines Pfads lebendig ist :-)

  $\Longrightarrow$ Als Halbordnung für $\mathbb{L}$ benötigen wir $\sqsubseteq \, = \, \subseteq$.
- Am Programmende ist keine Variable mehr lebendig :-)

Transformation 3:



Zur Korrektheit zeigt man:

→ Korrektheit der Kanten-Effekte: Falls L die Menge der legendigen Variablen am Ende eines Pfads π sind, dann ist $[\![\pi]\!]^\sharp\, L$ die Menge der am Anfang lebendigen Variablen :-)

→ Korrektheit der Transformation auf einem Pfad: Wird auf den Wert einer Variable zugegriffen, ist diese stets lebendig. Der Wert toter Variablen ist darum egal :-)

→ Korrektheit der Transformation: Bei Ausführung des transformierten Programms haben bei jedem Besuch eines Programmpunkts die lebendigen Variablen den gleichen Wert :-))

Berechnung der Mengen $\mathcal{L}^*[u]$:

(1) Aufstellen des Constraint-Systems:

$$\begin{array}{lll} \mathcal{L}[stop] & \supseteq & \emptyset \\ \mathcal{L}[u] & \supseteq & [\![k]\!]^\sharp(\mathcal{L}[v]) \qquad k = (u, \_, v) \quad \text{Kante} \end{array}$$

(2) Lösen des Constraint-Systems mittels RR-Iteration.

Da $\mathbb{L}$ endlich ist, terminiert die Iteration :-)

(3) Die kleinste Lösung $\mathcal{L}$ des Constraint-Systems ist gleich $\mathcal{L}^*$ da alle $[\![k]\!]^\sharp$ distributiv sind :-))

Berechnung der Mengen $\mathcal{L}^*[u]$:

(1) Aufstellen des Constraint-Systems:

$$\begin{array}{lll} \mathcal{L}[stop] & \supseteq & \emptyset \\ \mathcal{L}[u] & \supseteq & [\![k]\!]^\sharp\,(\mathcal{L}[v]) \qquad k = (u, \_, v) \quad \text{Kante} \end{array}$$

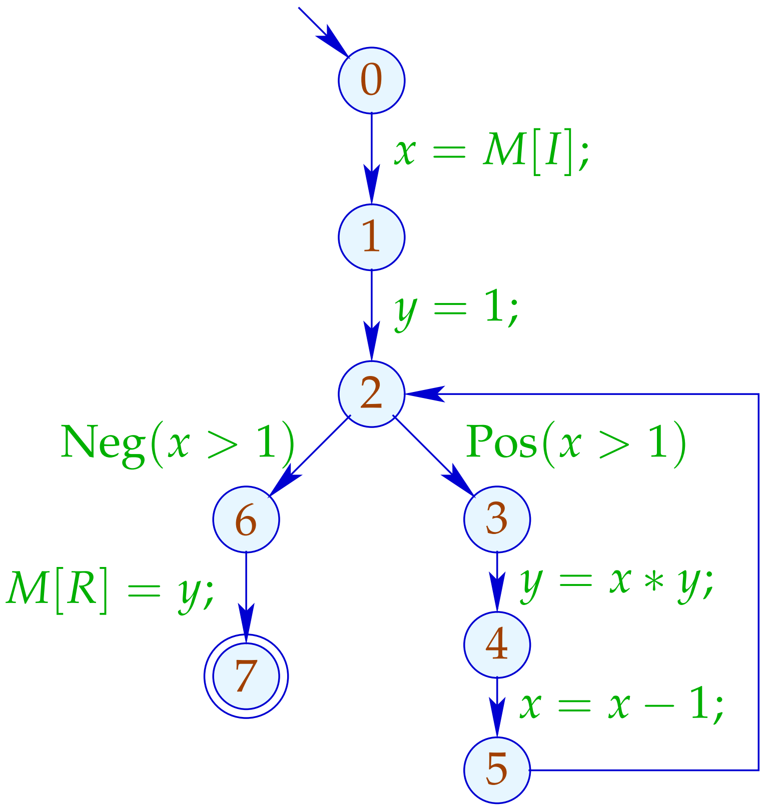
(2) Lösen des Constraint-Systems mittels RR-Iteration.

Da $\mathbb{L}$ endlich ist, terminiert die Iteration :-)

(3) Die kleinste Lösung $\mathcal{L}$ des Constraint-Systems ist gleich $\mathcal{L}^*$ da alle $[\![k]\!]^\sharp$ distributiv sind :-))

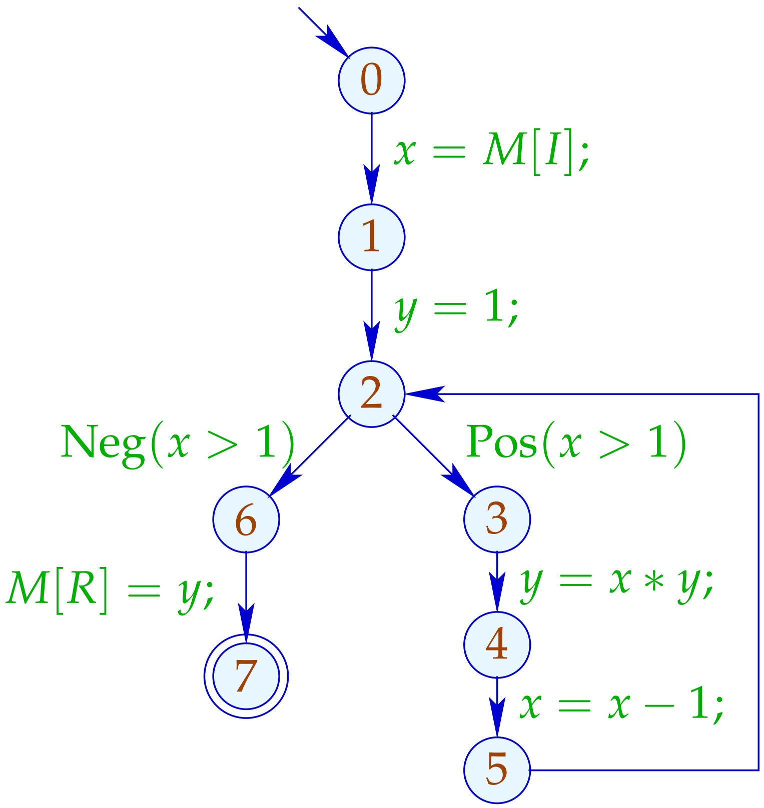
Achtung: Die Information wird rückwärts propagiert !!!

Beispiel:



$$
\begin{array}{lcl}
\mathcal{L}[0] & \supseteq & (\mathcal{L}[1] \backslash \{x\}) \cup \{I\} \\
\mathcal{L}[1] & \supseteq & \mathcal{L}[2] \backslash \{y\} \\
\mathcal{L}[2] & \supseteq & (\mathcal{L}[6] \cup \{x\}) \cup (\mathcal{L}[3] \cup \{x\}) \\
\mathcal{L}[3] & \supseteq & (\mathcal{L}[4] \backslash \{y\}) \cup \{x, y\} \\
\mathcal{L}[4] & \supseteq & (\mathcal{L}[5] \backslash \{x\}) \cup \{x\} \\
\mathcal{L}[5] & \supseteq & \mathcal{L}[2] \\
\mathcal{L}[6] & \supseteq & \mathcal{L}[7] \cup \{y, R\} \\
\mathcal{L}[7] & \supseteq & \emptyset
\end{array}
$$

Beispiel:

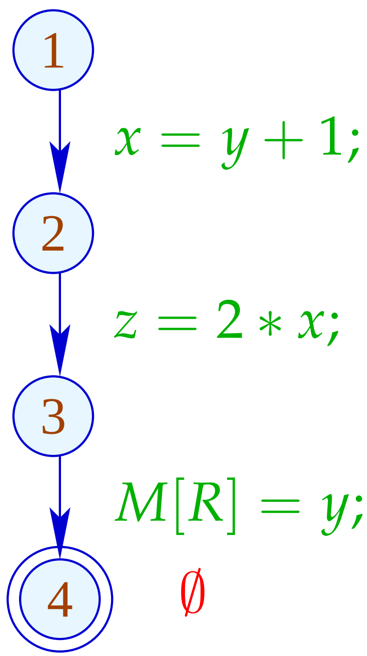


	1	2
7	$\emptyset$	
6	$\{y, R\}$	
2	$\{x, y, R\}$	dito
5	$\{x, y, R\}$	
4	$\{x, y, R\}$	
3	$\{x, y, R\}$	
1	$\{x, R\}$	
0	$\{I, R\}$	

Bei keiner Zuweisung ist die linke Variable tot :-)

Achtung:

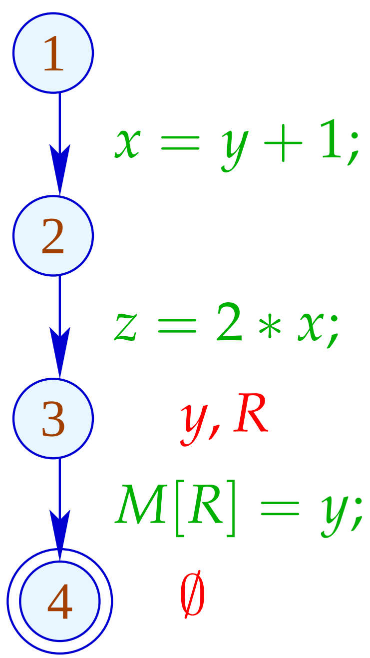
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot :-)

Achtung:

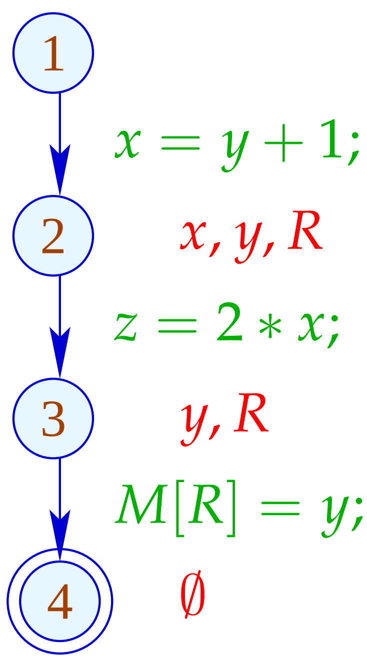
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot   :-)

Achtung:

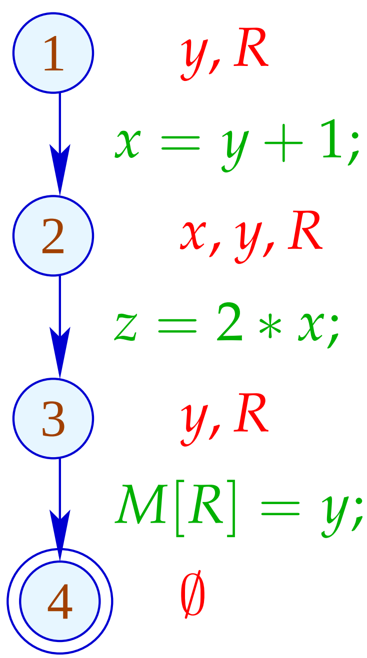
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot   :-)

Achtung:

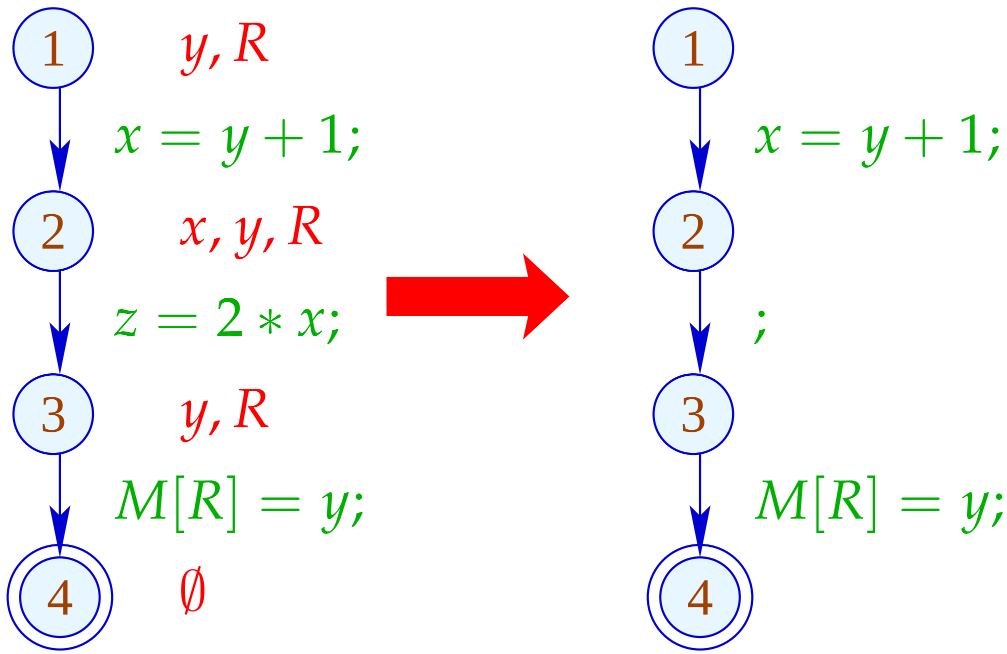
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot :-)

Achtung:

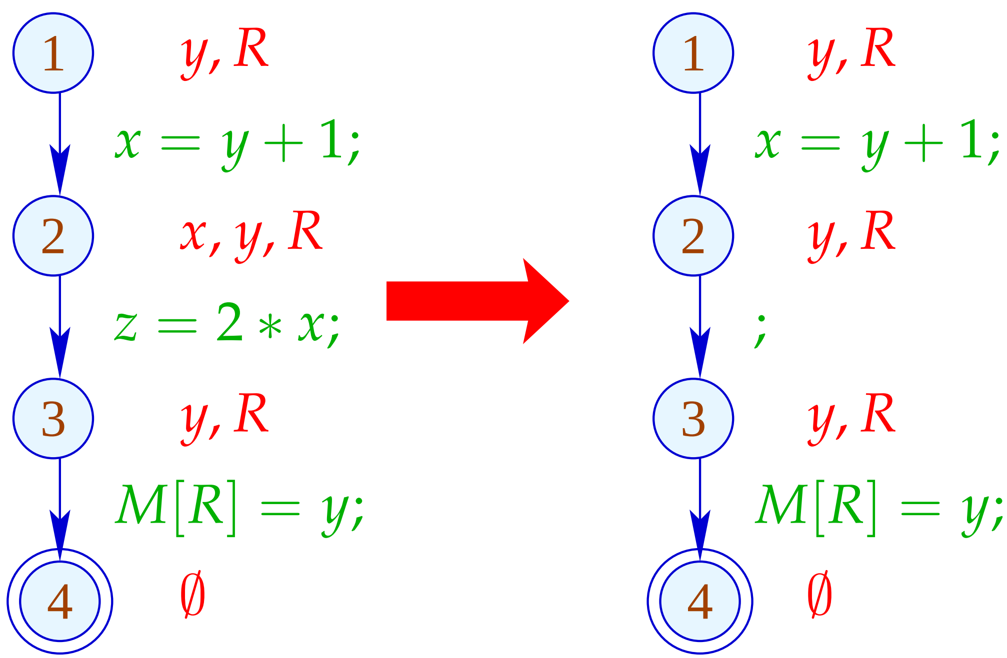
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot :-)

Achtung:

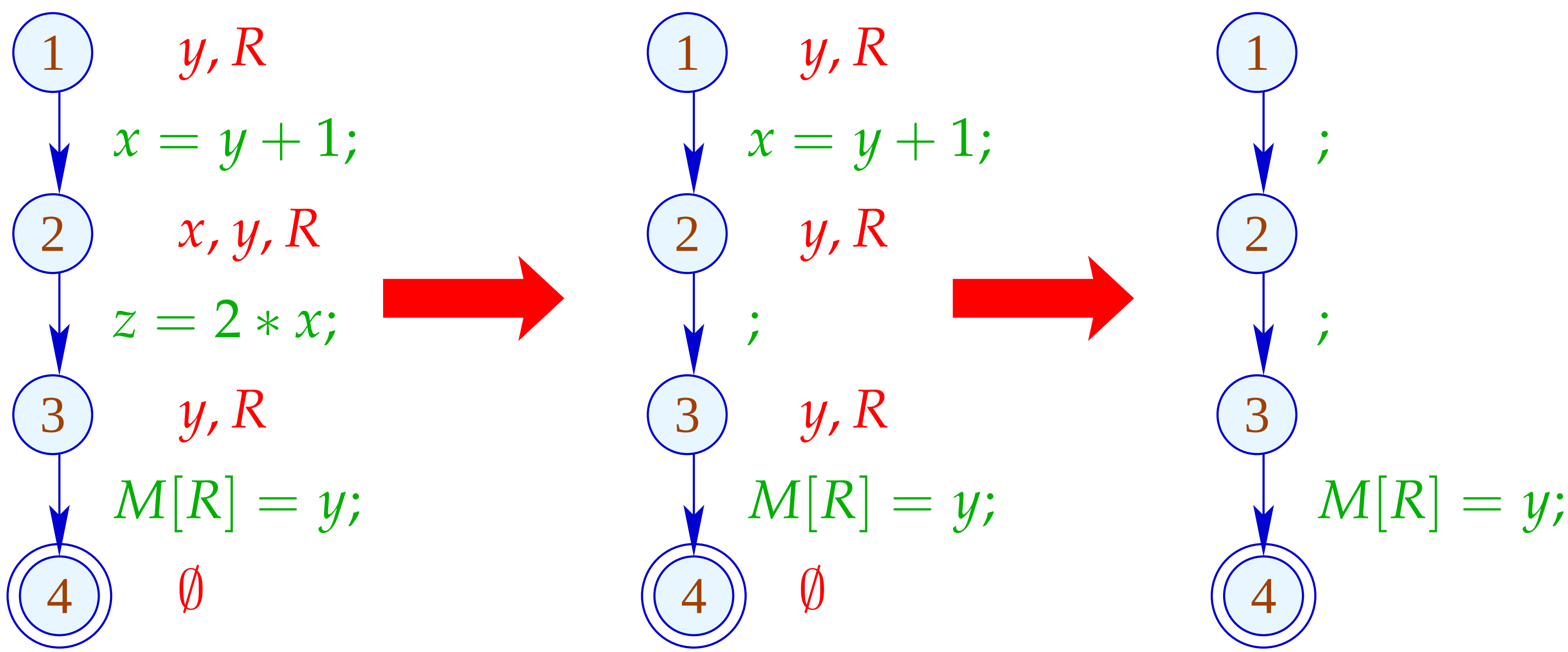
Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:



Bei keiner Zuweisung ist die linke Variable tot :-)

Achtung:

Beseitigung von Zuweisungen an tote Variablen kann weitere Variablen töten:

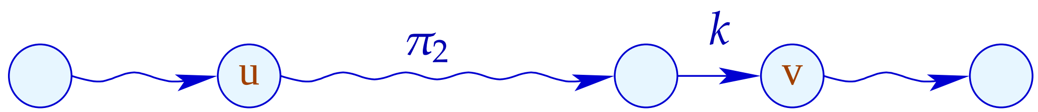


Das Programm mehrmals zu analysieren, ist hässlich :-(

Idee: Analysiere echte Lebendigkeit!

x heißt echt lebendig an u entlang eines Pfads π, falls sich π zerlegen lässt in $\pi = \pi_1\, \pi_2\, k\, \pi_3$ so dass gilt:

- π_1 erreicht u ;
- k ist eine echte Benutzung von x ;
- π_2 enthält keine Überschreibung von x.

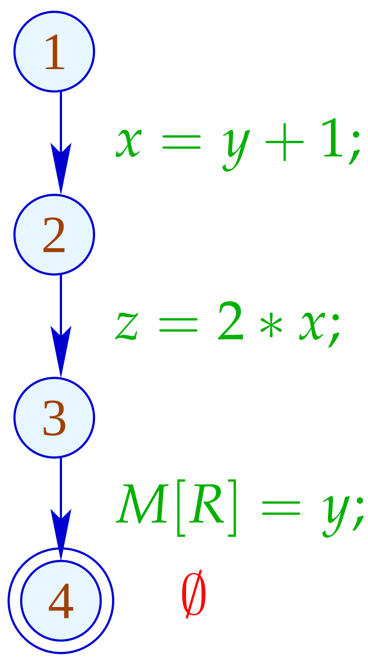


Die Menge der an einer Kante $k = (\_, lab, v)$ echt benutzten Variablen ist gegeben durch:

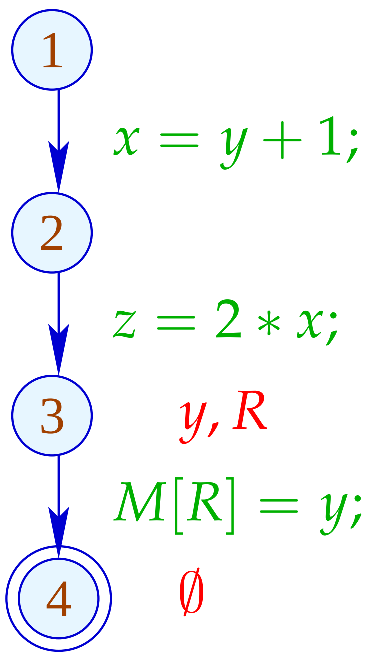
lab	echt benutzt
$;$	$\emptyset$
$\text{Pos}(e)$	$Vars(e)$
$\text{Neg}(e)$	$Vars(e)$
$x = e;$	$Vars(e)$ $(*)$
$x = M[R];$	$\{R\}$ $(*)$
$M[R_1] = R_2;$	$\{R_1, R_2\}$

$(*)$ – sofern x an v echt lebendig ist :-)

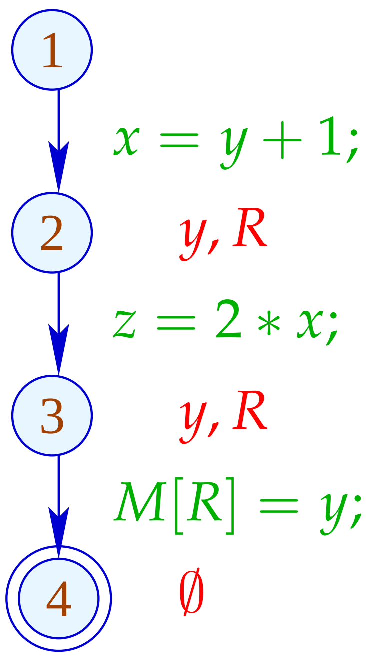
Beispiel:



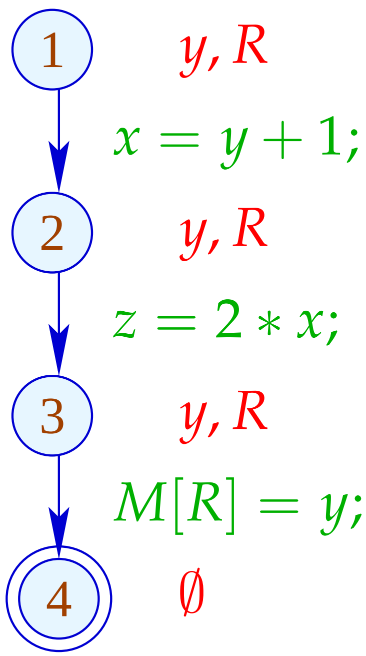
Beispiel:



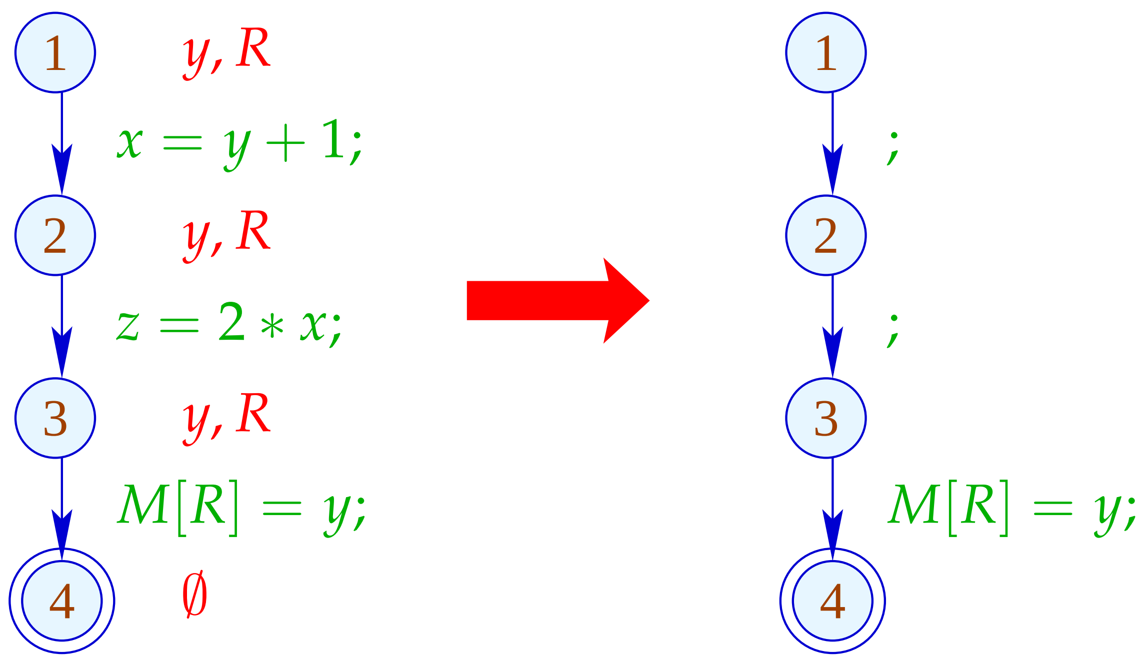
Beispiel:



Beispiel:



Beispiel:



Die Kanten-Effekte:

$$
\begin{array}{lclcl}
[\![;]\!]^\sharp\, L & = & L & & \\
[\![\mathrm{Pos}(e)]\!]^\sharp\, L & = & [\![\mathrm{Neg}(e)]\!]^\sharp\, L & = & L \cup \mathit{Vars}(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup & & \mathit{Vars}(e) \\
[\![R_1 = M[R_2];]\!]^\sharp\, L & = & (L \backslash \{R_1\}) \cup & & \{R_2\} \\
[\![M[R_1] = R_2;]\!]^\sharp\, L & = & L \cup \{R_1, R_2\} & &
\end{array}
$$

Die Kanten-Effekte:

$$
\begin{array}{lcl}
[\![;]\!]^\sharp\, L & = & L \\
[\![\mathrm{Pos}(e)]\!]^\sharp\, L & = & [\![\mathrm{Neg}(e)]\!]^\sharp\, L \quad = \quad L \cup \mathit{Vars}(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup \; (x \in L)\;?\;\mathit{Vars}(e) : \; \emptyset \\
[\![R_1 = M[R_2];]\!]^\sharp\, L & = & (L \backslash \{R_1\}) \cup \; (R_1 \in L)\;?\;\{R_2\} : \; \emptyset \\
[\![M[R_1] = R_2;]\!]^\sharp\, L & = & L \cup \{R_1, R_2\}
\end{array}
$$

Beachte:

- Die Kanten-Effekte für echt lebendige Variablen sind komplizierter als für lebendige Variablen :-)
- Sie sind aber immer noch distributiv !!

Beachte:

- Die Kanten-Effekte für echt lebendige Variablen sind komplizierter als für lebendige Variablen :-)
- Sie sind aber immer noch distributiv !!

  Dazu betrachten wir für $\mathbb{D} = 2^U$, $f\,y = (u \in y)\,?\,b :\, \emptyset$

  Wir überprüfen:

$$
\begin{array}{lll}
f\,(y_1 \cup y_2) & = & (u \in y_1 \cup y_2)\,?\,b :\, \emptyset \\
 & = & (u \in y_1 \vee u \in y_2)\,?\,b :\, \emptyset \\
 & = & (u \in y_1)\,?\,b :\, \emptyset \cup (u \in y_2)\,?\,b :\, \emptyset \\
 & = & f\,y_1 \cup f\,y_2
\end{array}
$$

Beachte:

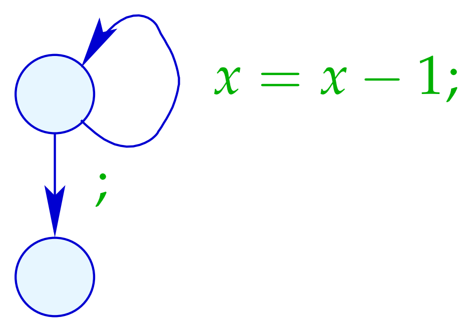
- Die Kanten-Effekte für echt lebendige Variablen sind komplizierter als für lebendige Variablen :-)
- Sie sind aber immer noch distributiv !!

  Dazu betrachten wir für $\mathbb{D} = 2^U$, $f\, y = (u \in y)\, ?\, b : \emptyset$
  Wir überprüfen:

$$
\begin{aligned}
f\,(y_1 \cup y_2) &= (u \in y_1 \cup y_2)\, ?\, b : \emptyset \\
&= (u \in y_1 \vee u \in y_2)\, ?\, b : \emptyset \\
&= (u \in y_1)\, ?\, b : \emptyset \cup (u \in y_2)\, ?\, b : \emptyset \\
&= f\, y_1 \cup f\, y_2
\end{aligned}
$$

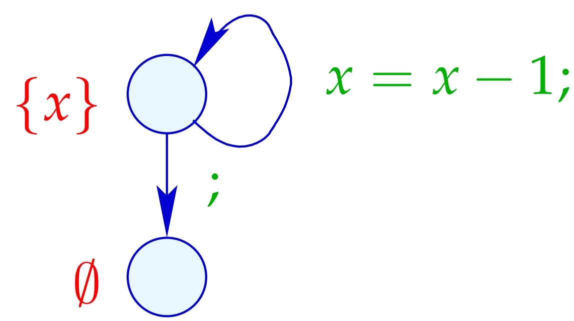
$\Longrightarrow$ Constraint-System liefert MOP :-))

- Echte Lebendigkeit findet mehr überflüssige Zuweisungen als wiederholte Lebendigkeit !!!



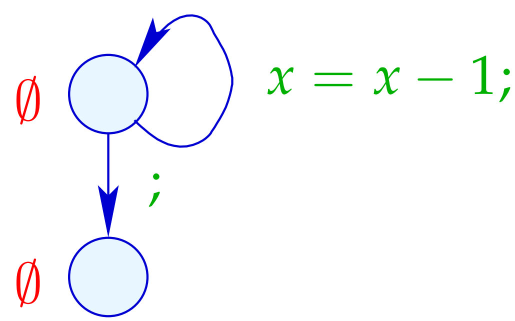
- Echte Lebendigkeit findet mehr überflüssige Zuweisungen als wiederholte Lebendigkeit !!!

Lebendigkeit:



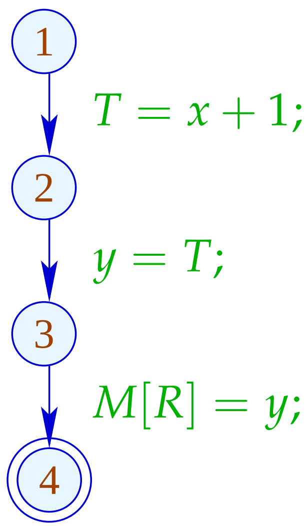
- Echte Lebendigkeit findet mehr überflüssige Zuweisungen als wiederholte Lebendigkeit !!!

Echte Lebendigkeit:



1.3 Beseitigung überflüssiger Umspeicherungen

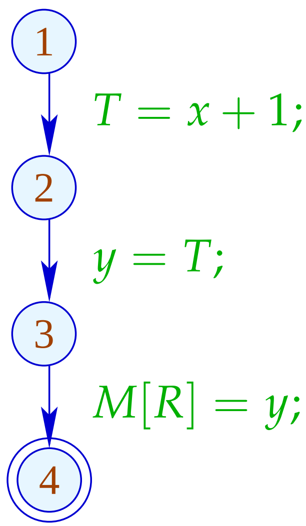
Beispiel:



Offenbar ist die Umspeicherung nutzlos :-(

1.3 Beseitigung überflüssiger Umspeicherungen

Beispiel:

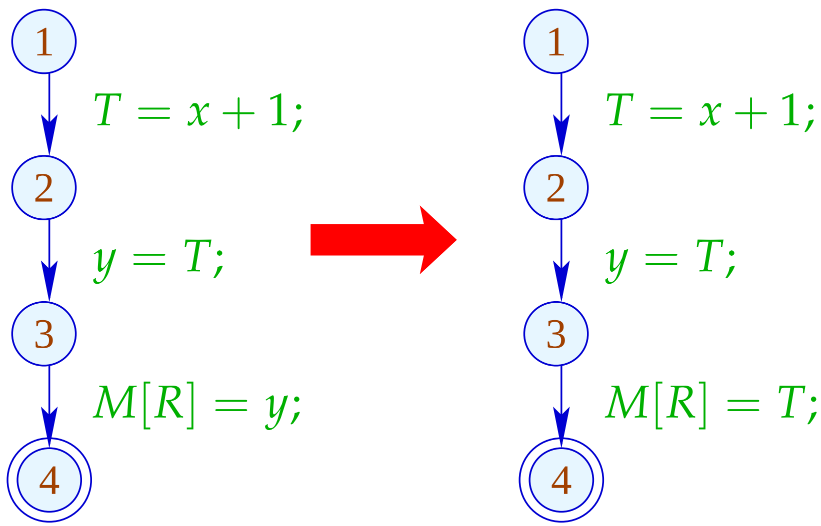


Offenbar ist die Umspeicherung nutzlos :-(

Statt y könnten wir auch T abspeichern :-)

1.3 Beseitigung überflüssiger Umspeicherungen

Beispiel:

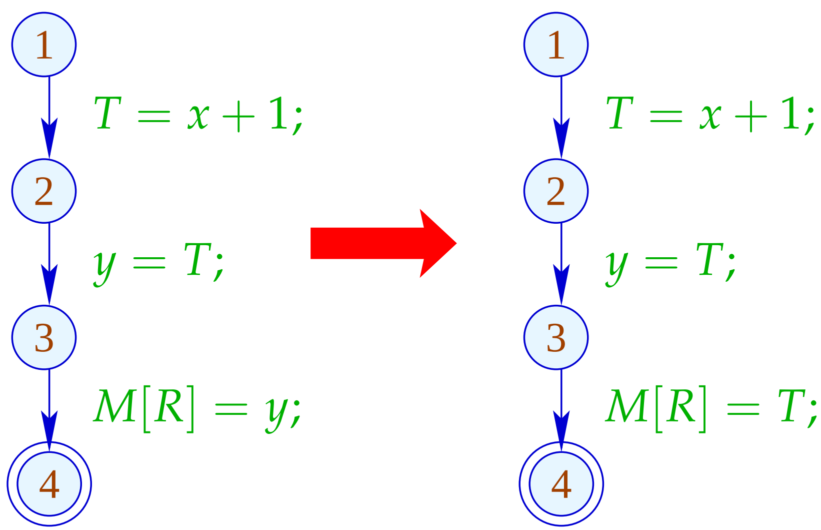


Offenbar ist die Umspeicherung nutzlos :-(

Statt y könnten wir auch T abspeichern :-)

1.3 Beseitigung überflüssiger Umspeicherungen

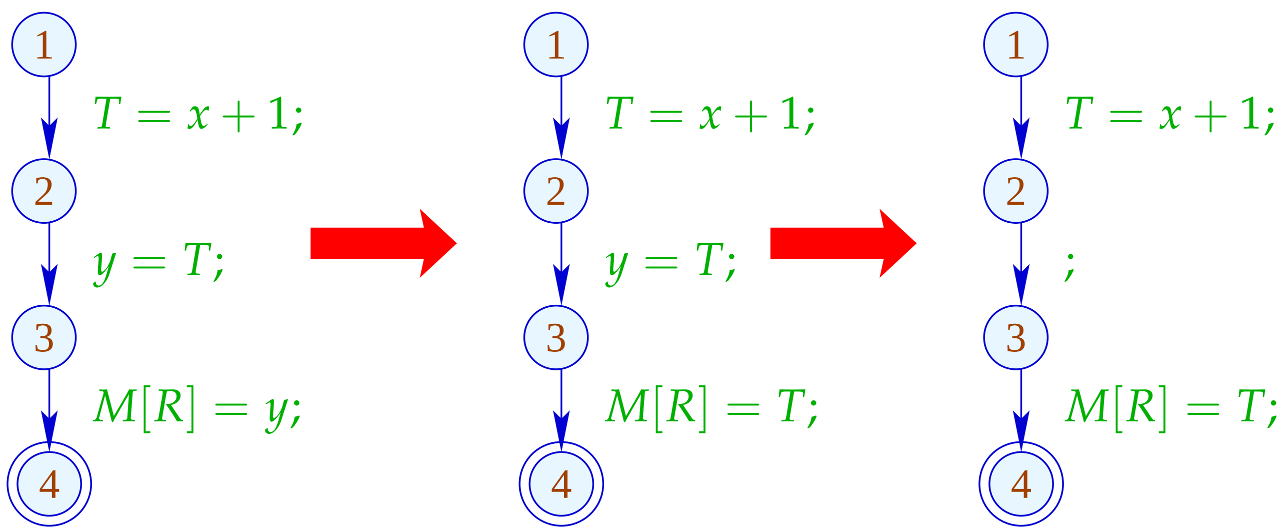
Beispiel:



Vorteil: Jetzt ist y tot :-))

1.3 Beseitigung überflüssiger Umspeicherungen

Beispiel:



Vorteil: Jetzt ist y tot :-))

Idee:

Für jeden Ausdruck merken wir uns die Variablen, die gegenwärtig seinen Wert enthalten :-)

Wir benutzen: $\mathbb{V} = \mathit{Expr} \to 2^{\mathit{Vars}}$ ...

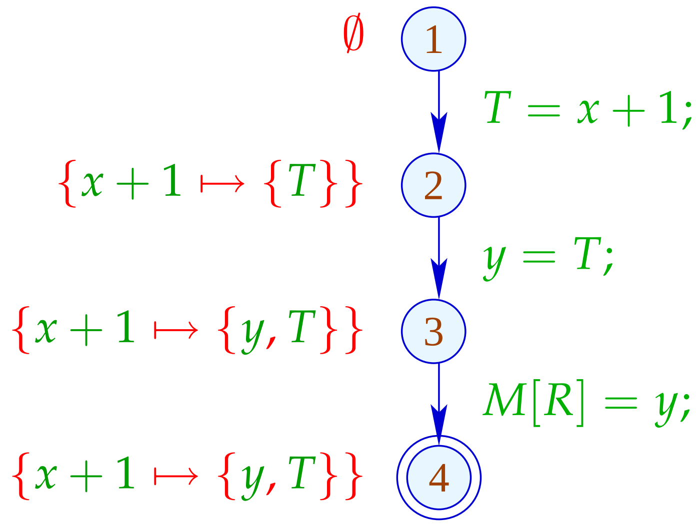
Idee:

Für jeden Ausdruck merken wir uns die Variablen, die gegenwärtig seinen Wert enthalten :-)

Wir benutzen: $\mathbb{V} = \mathit{Expr} \to 2^{\mathit{Vars}}$ und definieren:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, V & = & V \\
[\![\mathrm{Pos}(e)]\!]^\sharp\, V & = & [\![\mathrm{Neg}(e)]\!]^\sharp\, V \quad = \quad V \\
[\![x = e;]\!]^\sharp\, V\, e' & = & \left\{ \begin{array}{ll} \{x\} & \text{falls} \quad e' = e \\ (V\, e')\backslash\{x\} & \text{sonst} \end{array} \right. \\
[\![x = y;]\!]^\sharp\, V\, e & = & \left\{ \begin{array}{ll} (V\, e) \cup \{x\} & \text{falls} \quad y \in V\, e \\ (V\, e)\backslash\{x\} & \text{sonst} \end{array} \right. \\
[\![x = M[R];]\!]^\sharp\, V\, e & = & (V\, e)\backslash\{x\} \\
[\![M[R_1] = R_2;]\!]^\sharp\, V & = & V
\end{array}
$$

Im Beispiel:



Im Beispiel:

$$\begin{tikzcd} \emptyset & 1 \arrow[d, "T = x+1;"] \\ \{x+1 \mapsto \{T\}\} & 2 \arrow[d, "y = T;"] \\ \{x+1 \mapsto \{y, T\}\} & 3 \arrow[d, "M[R] = y;"] \\ \{x+1 \mapsto \{y, T\}\} & 4 \end{tikzcd}$$

$\longrightarrow$ Wir propagieren die Information vorwärts :-)

An *start* haben wir $V_0\, e = \emptyset$ für alle e

$\longrightarrow$ $\sqsubseteq\, \subseteq\, \mathbb{V} \times \mathbb{V}$ definieren wir durch:

$$V_1 \sqsubseteq V_2 \quad \text{gdw.} \quad V_1\, e \quad \supseteq \quad V_2\, e \qquad \text{für alle} \quad e$$

Beobachtung:

Die neuen Kanten-Effekte sind distributiv:

Dazu zeigen wir, dass die folgenden Funktionen distributiv sind:

(1) $\quad f_1\, V\, e = (V\, e)\backslash\{x\}$

(2) $\quad f_2\, V = V \oplus \{e \mapsto \{x\}\}$

(3) $\quad f_3\, V\, e = (y \in V\, e)\; ?\; (V\, e \cup \{x\}) : ((V\, e)\backslash\{x\})$

Offenbar gilt:

$$
\begin{array}{lcl}
[\![x = e;]\!]^\sharp & = & f_2 \circ f_1 \\
[\![x = y;]\!]^\sharp & = & f_3 \\
[\![x = M[R];]\!]^\sharp & = & f_1
\end{array}
$$

Distributivität ist unter Komposition abgeschlossen. Damit folgt die Behauptung :-))

(1) Für $f\,V\,e = (V\,e)\backslash\{x\}$ gilt:

$$\begin{aligned}
f\,(V_1 \sqcup V_2)\,e &= ((V_1 \sqcup V_2)\,e)\backslash\{x\} \\
&= ((V_1\,e) \cap (V_2\,e))\backslash\{x\} \\
&= ((V_1\,e)\backslash\{x\}) \cap ((V_2\,e)\backslash\{x\}) \\
&= (f\,V_1\,e) \cap (f\,V_2\,e) \\
&= (f\,V_1 \sqcup f\,V_2)\,e \qquad \text{:-)}
\end{aligned}$$