

$$\llbracket R_1 = M[R_2]; \rrbracket (\rho, \mu) = (\rho \oplus \{R_1 \mapsto \mu(\rho(R_2))\}, \mu)$$

$$\llbracket M[R_1] = R_2; \rrbracket (\rho, \mu) = (\rho, \mu \oplus \{\rho(R_1) \mapsto \rho(R_2)\})$$

Beispiel:

$$\llbracket x = x + 1; \rrbracket (\{x \mapsto 5\}, \mu) = (\rho, \mu) \quad \text{wobei:}$$

$$\begin{aligned} \rho &= \{x \mapsto 5\} \oplus \{x \mapsto \llbracket x + 1 \rrbracket \{x \mapsto 5\}\} \\ &= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\ &= \{x \mapsto 6\} \end{aligned}$$

Ein Pfad $\pi = k_1 k_2 \dots k_m$ ist eine **Berechnung** für den Zustand s falls:

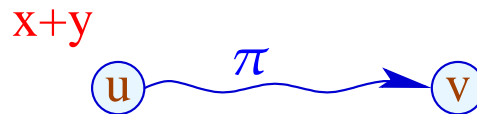
$$s \in \text{def} (\llbracket k_m \rrbracket \circ \dots \circ \llbracket k_1 \rrbracket)$$

Das **Ergebnis** der Berechnung ist:

$$\llbracket \pi \rrbracket s = (\llbracket k_m \rrbracket \circ \dots \circ \llbracket k_1 \rrbracket) s$$

Anwendung:

Nehmen wir an, wir hätten am Punkt u den Wert von $x + y$ berechnet:



Wir führen eine Berechnung entlang des Pfades π aus und erreichen v , wo wir erneut $x + y$ berechnen sollen ...

Idee:

Wenn x und y in π nicht verändert werden, dann muss $x + y$ in v den gleichen Wert liefern wie in u :-)

Diese Eigenschaft können wir an jeder Kante in π überprüfen :-}

Idee:

Wenn x und y in π nicht verändert werden, dann muss $x + y$ in v den gleichen Wert liefern wie in u :-)

Diese Eigenschaft können wir an jeder Kante in π überprüfen :-}

Allgemeiner:

Nehmen wir an, in u hätten wir die Werte der Ausdrücke aus $A = \{e_1, \dots, e_r\}$ zur Verfügung.

Idee:

Wenn x und y in π nicht verändert werden, dann muss $x + y$ in v den gleichen Wert liefern wie in u :-)

Diese Eigenschaft können wir an jeder Kante in π überprüfen :-}

Allgemeiner:

Nehmen wir an, in u hätten wir die Werte der Ausdrücke aus $A = \{e_1, \dots, e_r\}$ zur Verfügung.

Jede Kante k transformiert diese Menge in eine Menge $[[k]]^\# A$ von Ausdrücken, die nach Ausführung von k verfügbar sind ...

... die wir zur Ermittlung des **Effekts** eines Pfads $\pi = k_1 \dots k_r$ zusammen setzen können:

$$[[\pi]]^\# = [[k_r]]^\# \circ \dots \circ [[k_1]]^\#$$

... die wir zur Ermittlung des **Effekts** eines Pfads $\pi = k_1 \dots k_r$ zusammen setzen können:

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

Der Effekt $\llbracket k \rrbracket^\#$ einer Kante $k = (u, \text{lab}, v)$ hängt nur vom Label *lab* ab, d.h. $\llbracket k \rrbracket^\# = \llbracket \text{lab} \rrbracket^\#$

... die wir zur Ermittlung des **Effekts** eines Pfads $\pi = k_1 \dots k_r$ zusammensetzen können:

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

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

$$\llbracket ; \rrbracket^\# A = A$$

$$\llbracket \text{Pos}(e) \rrbracket^\# A = \llbracket \text{Neg}(e) \rrbracket^\# A = A \cup \{e\}$$

$$\llbracket R = e; \rrbracket^\# A = (A \cup \{e\}) \setminus \text{Expr}_R \quad \text{wobei}$$

Expr_R alle Ausdrücke sind, die R enthalten

$$\llbracket R_1 = M[R_2]; \rrbracket^\# A = A \setminus \text{Expr}_{R_1}$$

$$\llbracket M[R_1] = R_2; \rrbracket^\# A = A$$

$$\begin{aligned} \llbracket R_1 = M[R_2]; \rrbracket^\# A &= A \setminus \text{Expr}_{R_1} \\ \llbracket M[R_1] = R_2; \rrbracket^\# A &= A \end{aligned}$$

Damit können wir **jeden Pfad** untersuchen :-)

In einem Programm kann es **mehrere Pfade** geben :-)

Bei jeder Eingabe kann ein anderer gewählt werden :-((

$$\begin{aligned} \llbracket R_1 = M[R_2]; \rrbracket^\# A &= A \setminus \text{Expr}_{R_1} \\ \llbracket M[R_1] = R_2; \rrbracket^\# A &= A \end{aligned}$$

Damit können wir **jeden Pfad** untersuchen :-)

In einem Programm kann es **mehrere Pfade** geben :-)

Bei jeder Eingabe kann ein anderer gewählt werden :-((

⇒ Wir benötigen die Menge:

$$\mathcal{A}[v] = \bigcap \{ \llbracket \pi \rrbracket^\# \emptyset \mid \pi : \text{start} \rightarrow^* v \}$$

Im Klartext:

- Wir betrachten **sämtliche** Pfade, die v erreichen.
- Für jeden Pfad π bestimmen wir die Menge der entlang π verfügbaren Ausdrücke.
- Vor Programm-Ausführung ist **nichts** verfügbar :-)
- Wir bilden den **Durchschnitt** \implies **sichere Information**

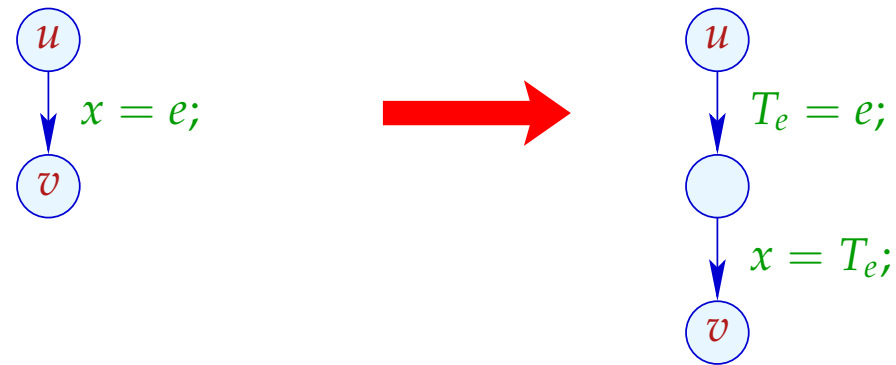
Im Klartext:

- Wir betrachten **sämtliche** Pfade, die v erreichen.
- Für jeden Pfad π bestimmen wir die Menge der entlang π verfügbaren Ausdrücke.
- Vor Programm-Ausführung ist **nichts** verfügbar :-)
- Wir bilden den **Durchschnitt** \implies **sichere Information**

Wie nutzen wir diese Information aus ???

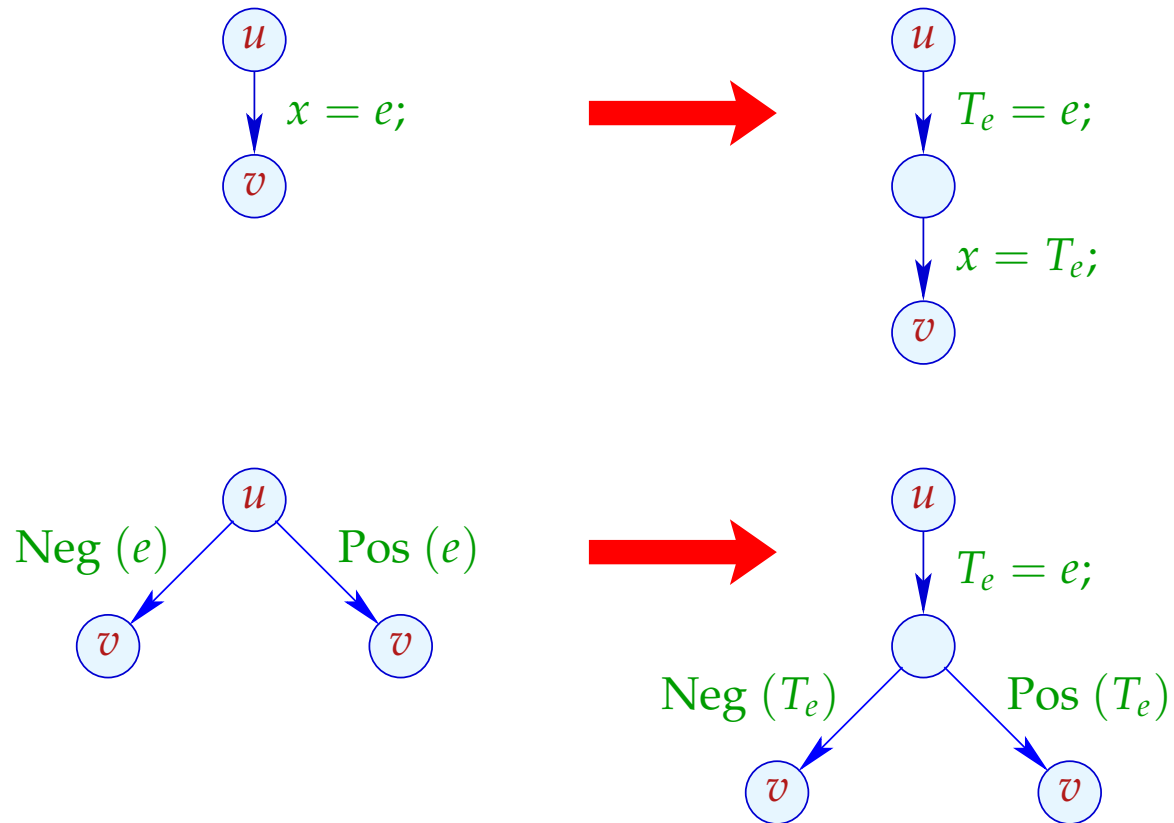
Transformation 1:

Wir stellen neue Register T_e als Speicherplatz für die e bereit:



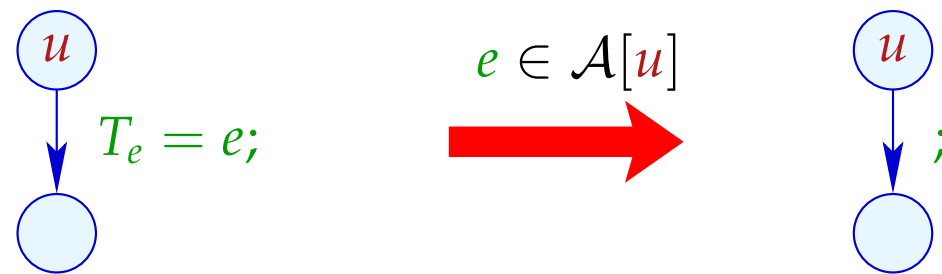
Transformation 1:

Wir stellen neue Register T_e als Speicherplatz für die e bereit:



Transformation 2:

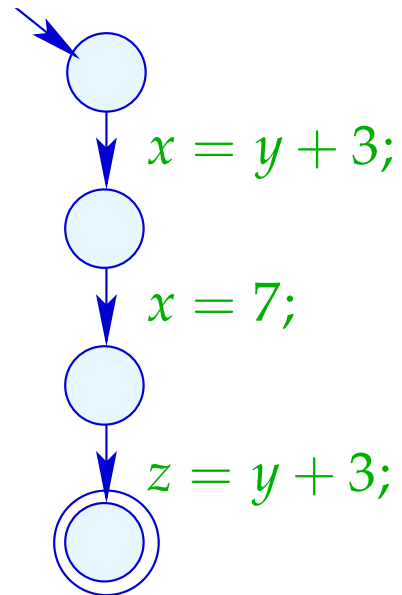
Falls e am Punkt u verfügbar ist, wird e nicht neu berechnet:



Wir ersetzen dann die Zuweisung durch *Nop* :-)

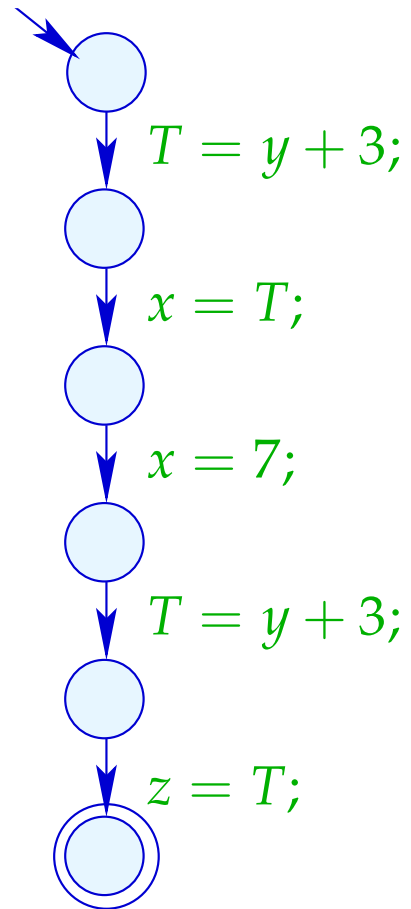
Beispiel:

$x = y + 3;$
 $x = 7;$
 $z = y + 3;$



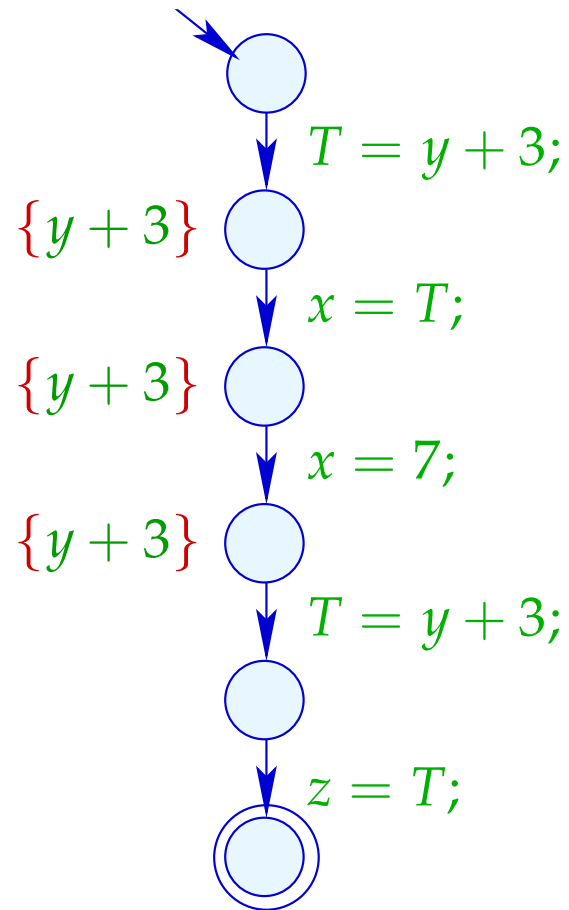
Beispiel:

$x = y + 3;$
 $x = 7;$
 $z = y + 3;$



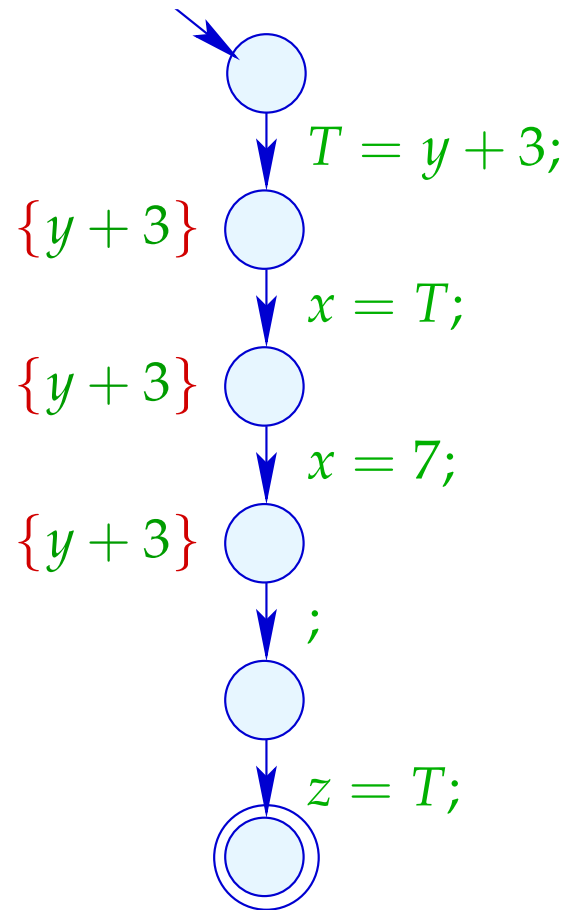
Beispiel:

$x = y + 3;$
 $x = 7;$
 $z = y + 3;$



Beispiel:

$x = y + 3;$
 $x = 7;$
 $z = y + 3;$



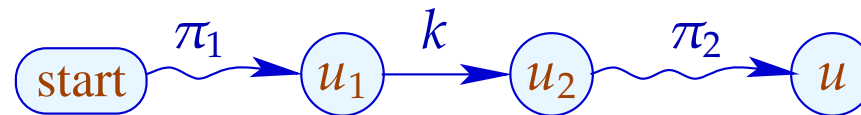
Korrektheit: (Idee)

Transformation 1 erhält offenbar die Bedeutung und $\mathcal{A}[u]$ für alle Knoten u :-)

Sei $\pi : start \rightarrow^* u$ der Pfad, den eine Berechnung nimmt.

Ist $e \in \mathcal{A}[u]$, dann auch $e \in \llbracket \pi \rrbracket^\# \emptyset$.

Dann muss es eine Zerlegung von π geben:



mit den folgenden Eigenschaften:

- Der Ausdruck e wird an der Kante k berechnet;
- Der Ausdruck e wird an keiner Kante in π_2 aus der Menge der verfügbaren Ausdrücke entfernt, d.h. keine Variable von e erhält einen neuen Wert :-)

- Der Ausdruck e wird an der Kante k berechnet;
- Der Ausdruck e wird an keiner Kante in π_2 aus der Menge der verfügbaren Ausdrücke entfernt, d.h. keine Variable von e erhält einen neuen Wert :-)



Wird u erreicht, enthält das Register T_e den Wert von e :-))

Achtung:

Die Transformation 1 ist nur sinnvoll an Zuweisungen $x = e;$, wobei:

- $x \notin \text{Vars}(e);$
- $e \notin \text{Vars};$
- sich die Berechnung von e lohnt :-}

Achtung:

Die Transformation 1 ist nur sinnvoll an Zuweisungen $x = e;$,
wobei:

- $x \notin \text{Vars}(e);$
- $e \notin \text{Vars};$
- sich die Berechnung von e lohnt :-}

Bleibt die Preisfrage ...

Preisfrage:

Wie berechnen wir $\mathcal{A}[u]$ für jeden Programmpunkt u ??

Preisfrage:

Wie berechnen wir $\mathcal{A}[u]$ für jeden Programmpunkt u ??

Idee:

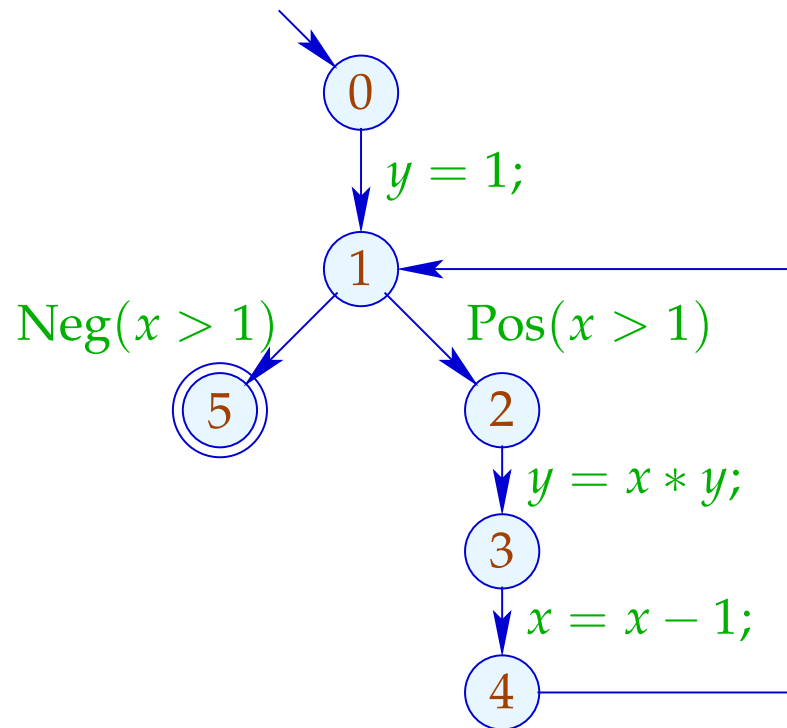
Wir stellen ein **Constraint-System** auf, das alle Bedingungen an die Werte $\mathcal{A}[u]$ sammelt:

$$\begin{aligned}\mathcal{A}[\textit{start}] &\subseteq \emptyset \\ \mathcal{A}[v] &\subseteq \llbracket k \rrbracket^\# (\mathcal{A}[u]) \quad k = (u, _, v) \text{ Kante}\end{aligned}$$

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:

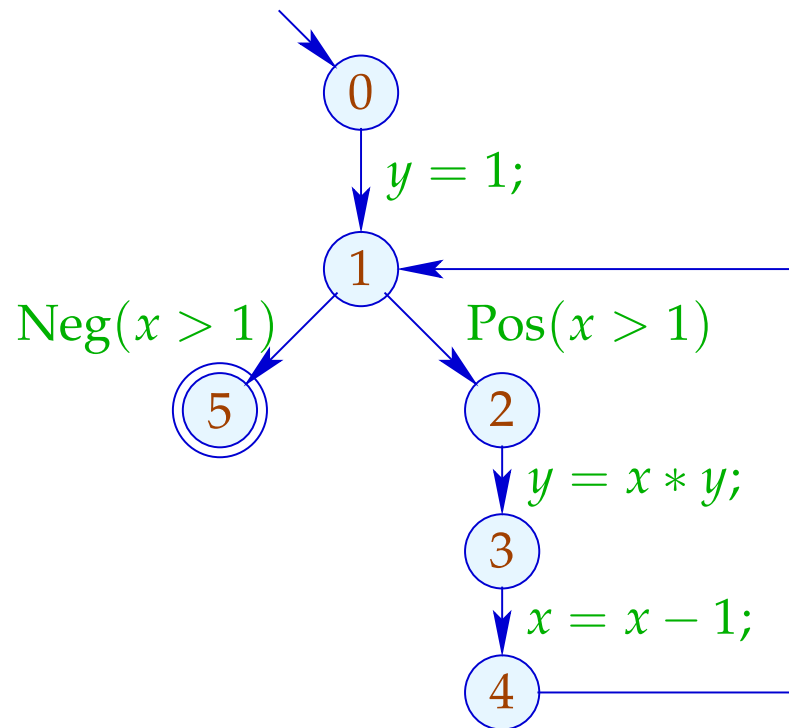


[]

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



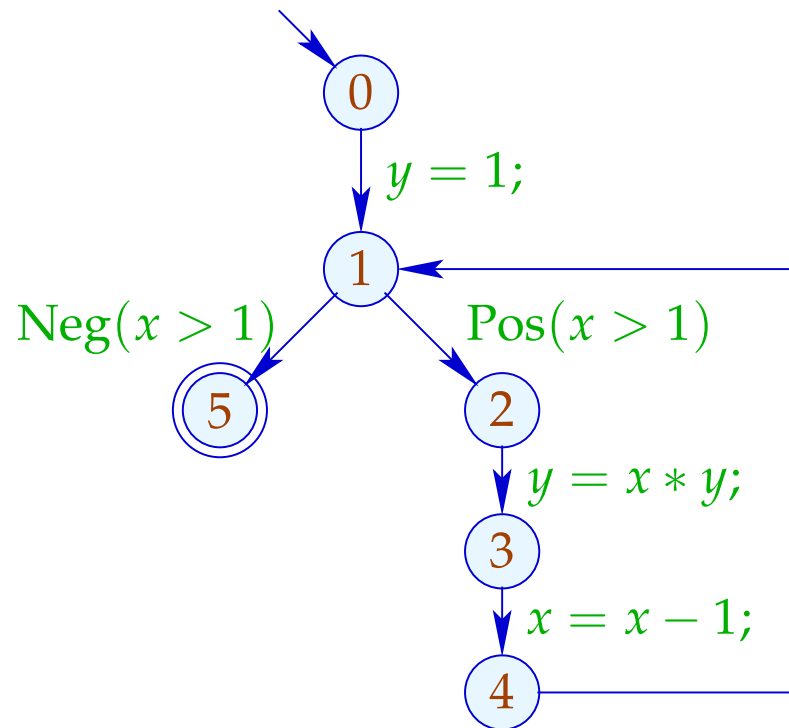
$\mathcal{A}[0] \subseteq \emptyset$

[]

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



$$\mathcal{A}[0] \subseteq \emptyset$$

$$\mathcal{A}[1] \subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y$$

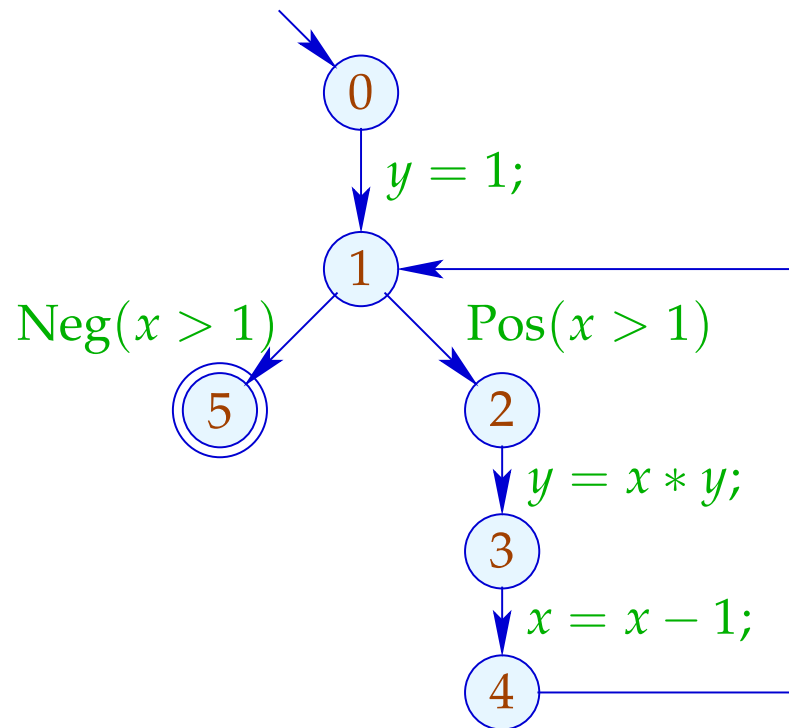
$$\mathcal{A}[1] \subseteq \mathcal{A}[4]$$

[]

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



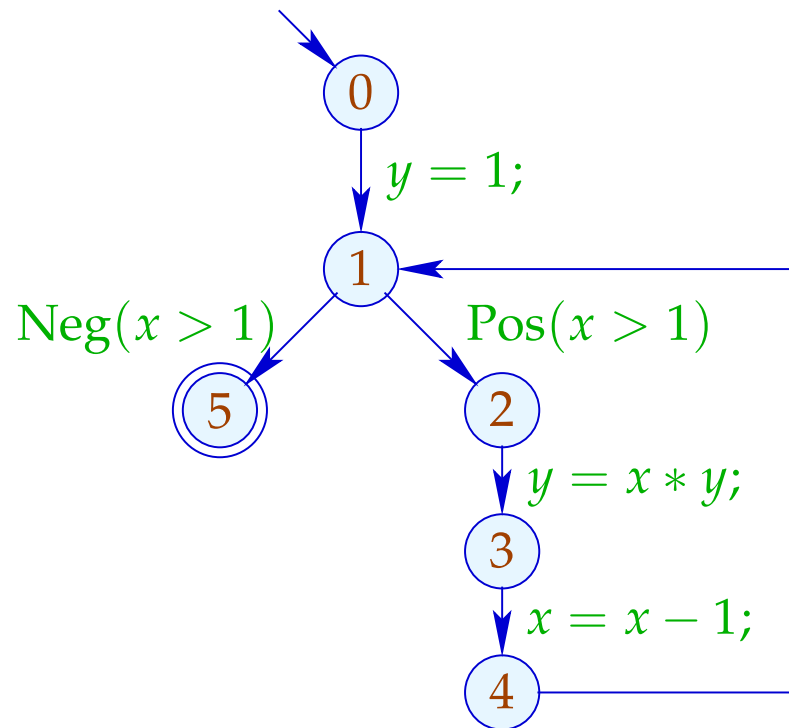
$$\begin{aligned}\mathcal{A}[0] &\subseteq \emptyset \\ \mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y \\ \mathcal{A}[1] &\subseteq \mathcal{A}[4] \\ \mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\}\end{aligned}$$

[]

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



$$\mathcal{A}[0] \subseteq \emptyset$$

$$\mathcal{A}[1] \subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y$$

$$\mathcal{A}[1] \subseteq \mathcal{A}[4]$$

$$\mathcal{A}[2] \subseteq \mathcal{A}[1] \cup \{x > 1\}$$

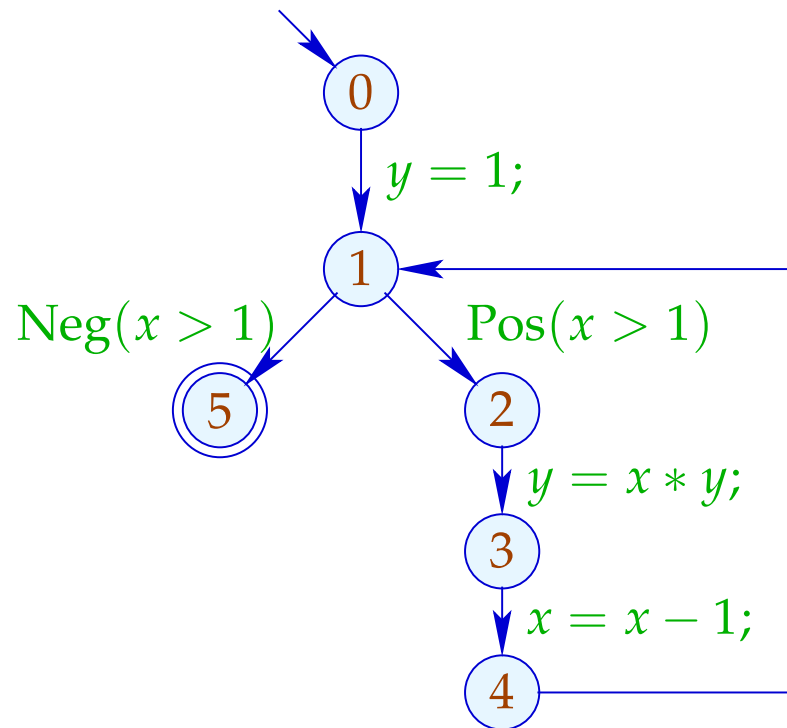
$$\mathcal{A}[3] \subseteq (\mathcal{A}[2] \cup \{x * y\}) \setminus \text{Expr}_y$$

[]

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:

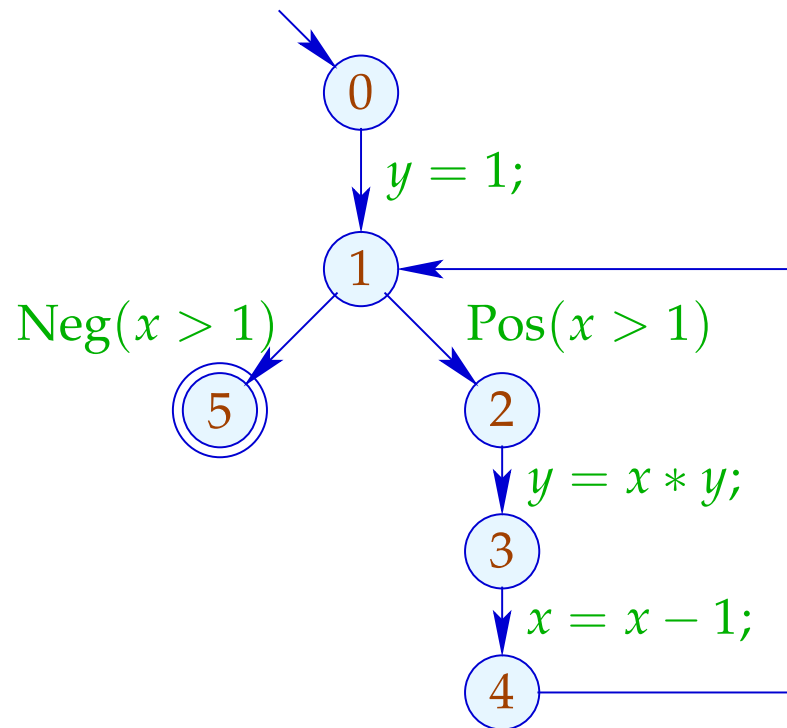


$$\begin{aligned}\mathcal{A}[0] &\subseteq \emptyset \\ \mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y \\ \mathcal{A}[1] &\subseteq \mathcal{A}[4] \\ \mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\} \\ \mathcal{A}[3] &\subseteq (\mathcal{A}[2] \cup \{x * y\}) \setminus \text{Expr}_y \\ \mathcal{A}[4] &\subseteq (\mathcal{A}[3] \cup \{x - 1\}) \setminus \text{Expr}_x \\ &[]\end{aligned}$$

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



$$\mathcal{A}[0] \subseteq \emptyset$$

$$\mathcal{A}[1] \subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y$$

$$\mathcal{A}[1] \subseteq \mathcal{A}[4]$$

$$\mathcal{A}[2] \subseteq \mathcal{A}[1] \cup \{x > 1\}$$

$$\mathcal{A}[3] \subseteq (\mathcal{A}[2] \cup \{x * y\}) \setminus \text{Expr}_y$$

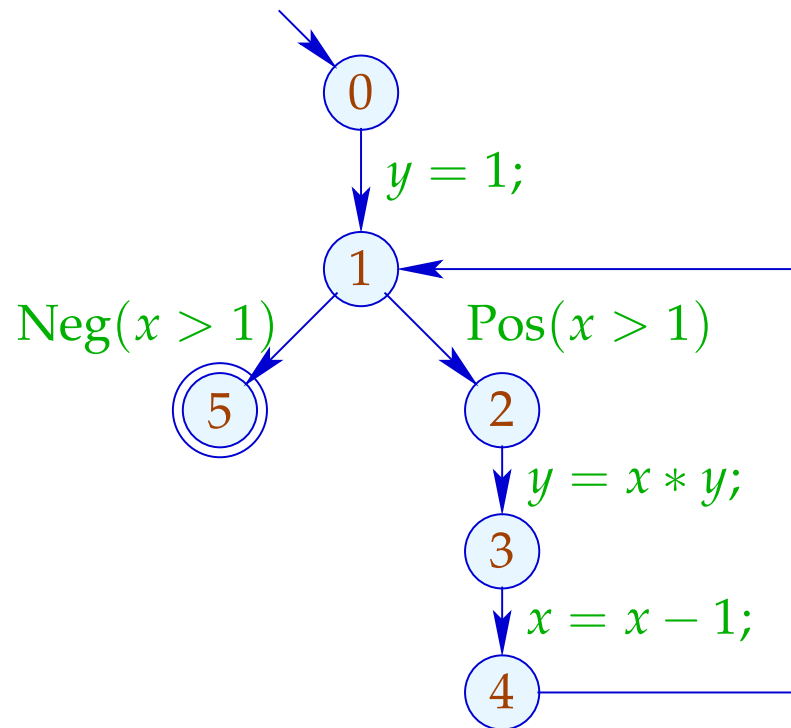
$$\mathcal{A}[4] \subseteq (\mathcal{A}[3] \cup \{x - 1\}) \setminus \text{Expr}_x$$

$$\mathcal{A}[5] \subseteq \mathcal{A}[1] \cup \{x > 1\}$$

Gesucht:

- möglichst **große** Lösung (??)
- Algorithmus, der diese berechnet :-)

Beispiel:



Lösung:

$$\begin{aligned}\mathcal{A}[0] &= \emptyset \\ \mathcal{A}[1] &= \{1\} \\ \mathcal{A}[2] &= \{1, x > 1\} \\ \mathcal{A}[3] &= \{1, x > 1\} \\ \mathcal{A}[4] &= \{1\} \\ \mathcal{A}[5] &= \{1, x > 1\}\end{aligned}$$

Beobachtung:

- Die möglichen Werte für $\mathcal{A}[u]$ bilden einen **vollständigen Verband**:

$$\mathbb{D} = 2^{Expr} \quad \text{mit} \quad B_1 \sqsubseteq B_2 \quad \text{gdw.} \quad B_1 \supseteq B_2$$

Beobachtung:

- Die möglichen Werte für $\mathcal{A}[u]$ bilden einen **vollständigen Verband**:

$$\mathbb{D} = 2^{Expr} \quad \text{mit} \quad B_1 \sqsubseteq B_2 \quad \text{gdw.} \quad B_1 \supseteq B_2$$

- Die Funktionen $\llbracket k \rrbracket^\# : \mathbb{D} \rightarrow \mathbb{D}$ sind **monoton**, d.h.

$$\llbracket k \rrbracket^\#(B_1) \sqsubseteq \llbracket k \rrbracket^\#(B_2) \quad \text{gdw.} \quad B_1 \sqsubseteq B_2$$

Exkurs 2: Vollständige Verbände

Eine Menge \mathbb{D} mit einer Relation $\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$ ist eine **partielle Ordnung** falls für alle $a, b, c \in \mathbb{D}$ gilt:

$$a \sqsubseteq a$$

Reflexivität

$$a \sqsubseteq b \wedge b \sqsubseteq a \implies a = b$$

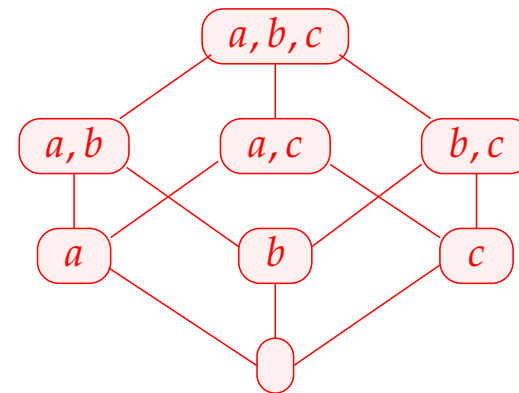
Anti – Symmetrie

$$a \sqsubseteq b \wedge b \sqsubseteq c \implies a \sqsubseteq c$$

Transitivität

Beispiele:

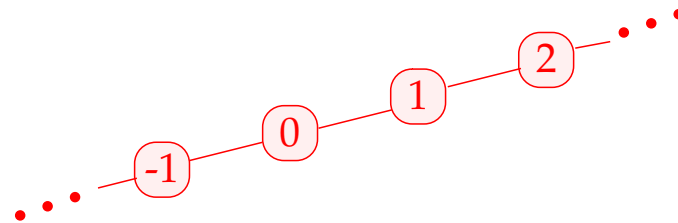
1. $\mathbb{D} = 2^{\{a,b,c\}}$ mit der Relation " \subseteq ":



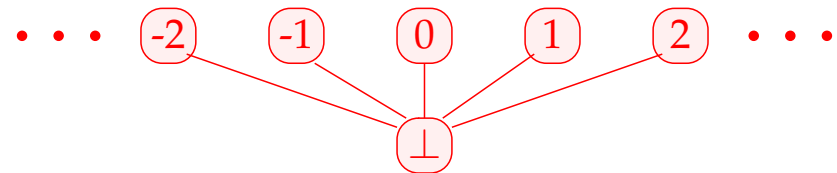
3. \mathbb{Z} mit der Relation “=” :



3. \mathbb{Z} mit der Relation “ \leq ” :



4. $\mathbb{Z}_{\perp} = \mathbb{Z} \cup \{\perp\}$ mit der Ordnung:



$d \in \mathbb{D}$ heißt **obere Schranke** für $X \subseteq \mathbb{D}$ falls

$$x \leq d \quad \text{für alle } x \in X$$

$d \in \mathbb{D}$ heißt **obere Schranke** für $X \subseteq \mathbb{D}$ falls

$$x \leq d \quad \text{für alle } x \in X$$

d heißt **kleinste obere Schranke (lub)** falls

1. d eine obere Schranke ist und
2. $d \leq y$ für jede obere Schranke y für X .

$d \in \mathbb{D}$ heißt **obere Schranke** für $X \subseteq \mathbb{D}$ falls

$$x \leq d \quad \text{für alle } x \in X$$

d heißt **kleinste obere Schranke (lub)** falls

1. d eine obere Schranke ist und
2. $d \leq y$ für jede obere Schranke y für X .

Achtung:

- $\{0, 2, 4, \dots\} \subseteq \mathbb{Z}$ besitzt **keine** obere Schranke!
- $\{0, 2, 4\} \subseteq \mathbb{Z}$ besitzt die oberen Schranken **4, 5, 6, ...**

Ein **vollständiger Verband (cl)** \mathbb{D} ist eine partielle Ordnung, in der **jede Teilmenge** $X \subseteq \mathbb{D}$ eine kleinste obere Schranke $\bigsqcup X \in \mathbb{D}$ besitzt.

Beachte:

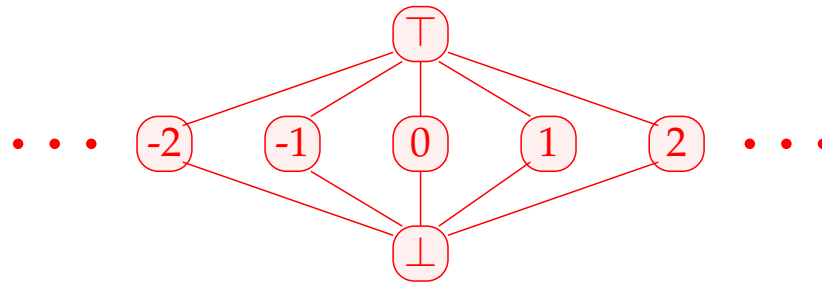
Jeder vollständige Verband besitzt

→ ein **kleinstes** Element $\perp = \bigsqcup \emptyset \in \mathbb{D}$;

→ ein **größtes** Element $\top = \bigsqcup \mathbb{D} \in \mathbb{D}$.

Beispiele:

1. $\mathbb{D} = 2^{\{a,b,c\}}$ ist ein cl :-)
2. $\mathbb{D} = \mathbb{Z}$ mit “=” ist keiner.
3. $\mathbb{D} = \mathbb{Z}$ mit “ \leq ” ebenfalls nicht.
4. $\mathbb{D} = \mathbb{Z}_{\perp}$ auch nicht :-)
5. Mit einem zusätzlichen Symbol \top erhalten wir den **flachen** Verband $\mathbb{Z}_{\perp}^{\top} = \mathbb{Z} \cup \{\perp, \top\}$:



Es gilt:

Satz:

In jedem vollständigen Verband \mathbb{D} besitzt jede Teilmenge $X \subseteq \mathbb{D}$ eine **größte untere Schranke** $\bigwedge X$.

Es gilt:

Satz:

In jedem vollständigen Verband \mathbb{D} besitzt jede Teilmenge $X \subseteq \mathbb{D}$ eine **größte untere Schranke** $\sqcap X$.

Beweis:

Konstruiere $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$.

// die Menge der unteren Schranken von X :-)

Es gilt:

Satz:

In jedem vollständigen Verband \mathbb{D} besitzt jede Teilmenge $X \subseteq \mathbb{D}$ eine **größte untere Schranke** $\sqcap X$.

Beweis:

Konstruiere $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$.

// die Menge der unteren Schranken von X :-)

Setze: $g := \sqcup U$

Behauptung: $g = \sqcap X$

(1) g ist eine **untere Schranke** von X :

Für $x \in X$ gilt:

$$u \sqsubseteq x \text{ für alle } u \in U$$

$\implies x$ ist obere Schranke von U

$\implies g \sqsubseteq x \quad :-)$

(1) g ist eine **untere Schranke** von X :

Für $x \in X$ gilt:

$$u \sqsubseteq x \text{ für alle } u \in U$$

$$\implies x \text{ ist obere Schranke von } U$$

$$\implies g \sqsubseteq x \quad :-)$$

(2) g ist **größte untere Schranke** von X :

Für jede untere Schranke u von X gilt:

$$u \in U$$

$$\implies u \sqsubseteq g \quad :-))$$