... offenbar ist das Ergebnis enttäuschend :-(

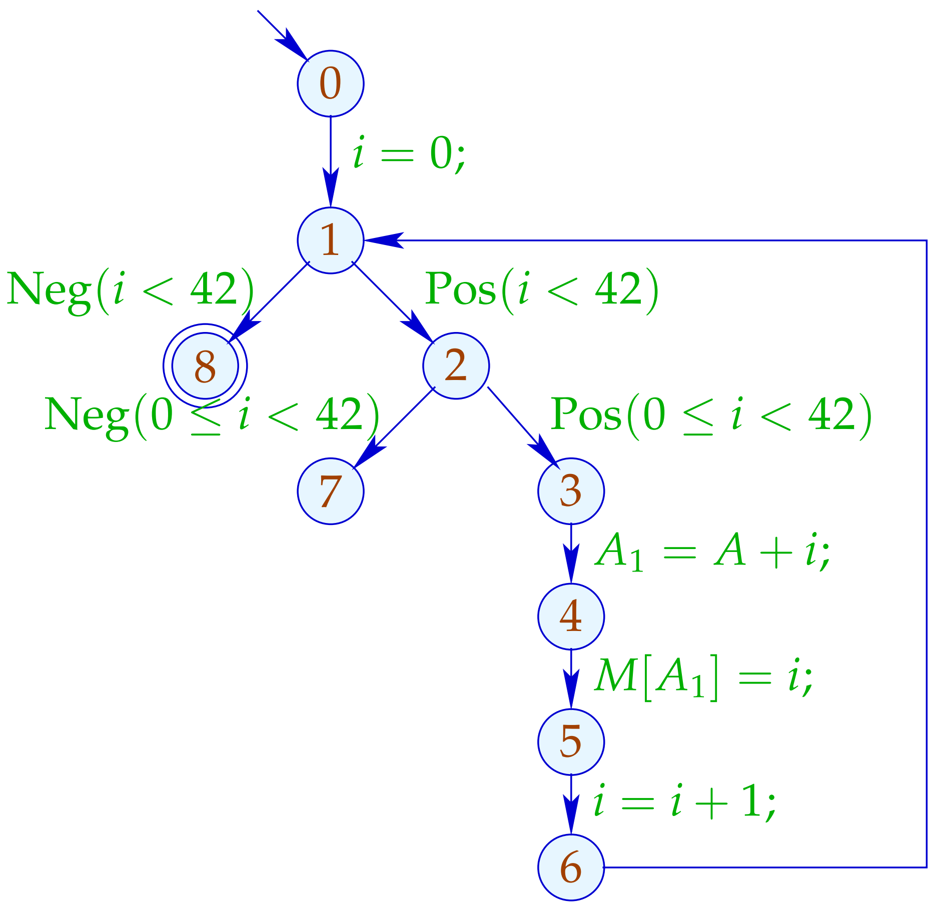
## Idee 2:

Eigentlich reicht es, die Beschleunigung mittels $\sqcup$ nur an genügend vielen Stellen anzuwenden!

Eine Menge $I$ heißt Loop Separator (Kreis-Trenner), falls jeder Kreis mindestens einen Punkt aus $I$ enthält :-)

Wenden wir Widening nur an den Punkten aus einer solchen Menge $I$ , terminiert RR-Iteration immer noch !!!

# In unserem Beispiel:

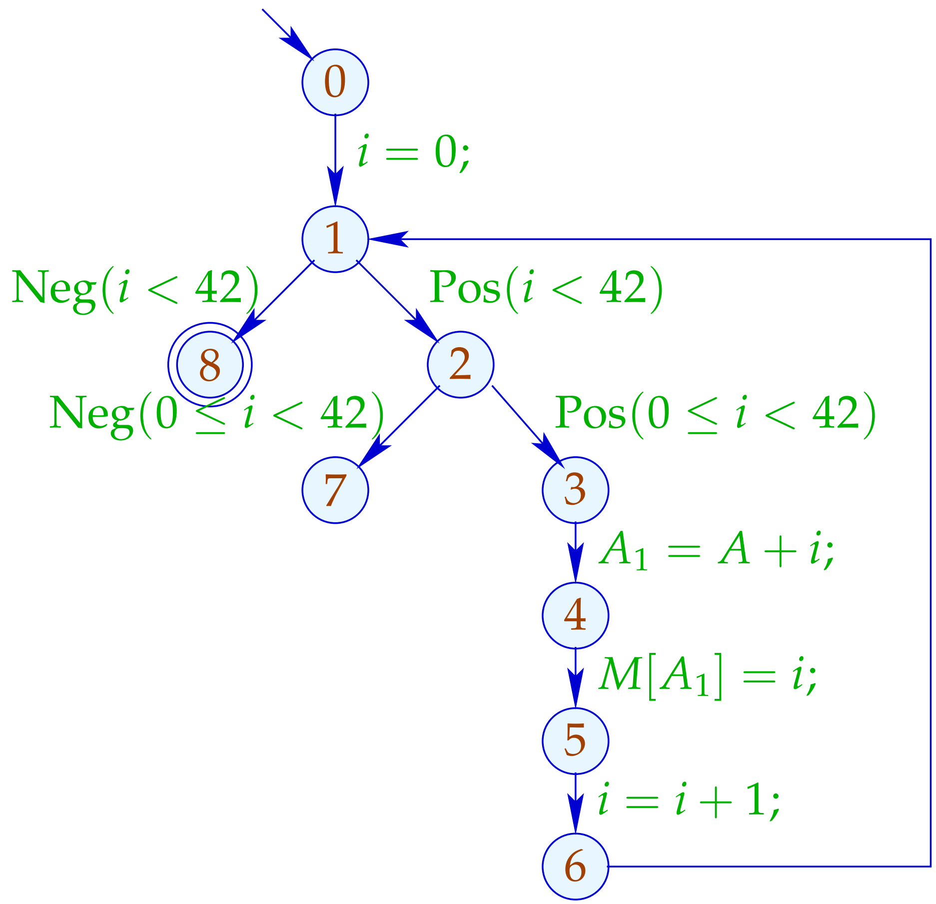



$I_1 = \{1\}$ oder auch:

$I_2 = \{2\}$ oder auch:

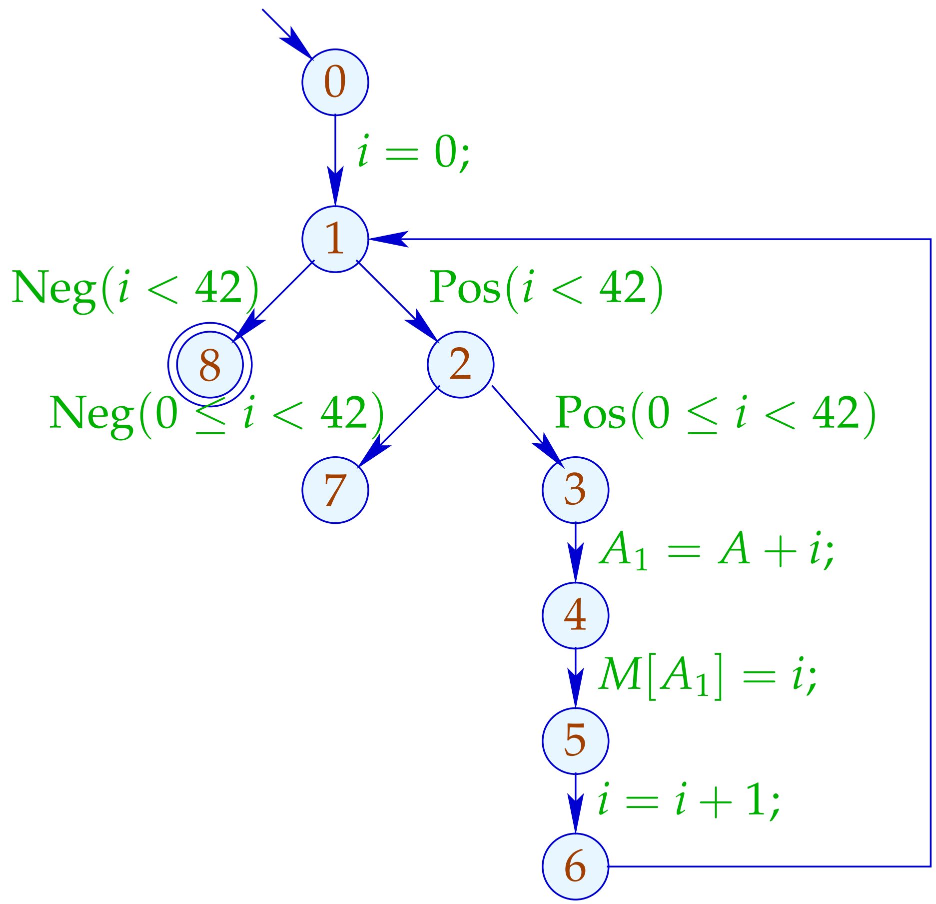
$I_3 = \{3\}$

## Die Analyse mit $I = \{1\}$ :




| | 1 | | 2 | | 3 | |
|---|---|---|---|---|---|---|
| | $l$ | $u$ | $l$ | $u$ | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ | dito | |
| 1 | 0 | 0 | 0 | $+\infty$ | | |
| 2 | 0 | 0 | 0 | 41 | | |
| 3 | 0 | 0 | 0 | 41 | | |
| 4 | 0 | 0 | 0 | 41 | | |
| 5 | 0 | 0 | 0 | 41 | | |
| 6 | 1 | 1 | 1 | 42 | | |
| 7 | $\perp$ | | $\perp$ | | | |
| 8 | $\perp$ | | 42 | $+\infty$ | | |

## Die Analyse mit $I = \{2\}$ :




| | 1 | | 2 | | 3 | |
|---|---|---|---|---|---|---|
| | $l$ | $u$ | $l$ | $u$ | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ | | |
| 1 | 0 | 0 | 0 | 42 | | |
| 2 | 0 | 0 | 0 | $+\infty$ | | |
| 3 | 0 | 0 | 0 | 41 | | |
| 4 | 0 | 0 | 0 | 41 | dito | |
| 5 | 0 | 0 | 0 | 41 | | |
| 6 | 1 | 1 | 1 | 42 | | |
| 7 | $\perp$ | | 42 | $+\infty$ | | |
| 8 | $\perp$ | | 42 | 42 | | |

## Diskussion:

- Beide Analysen-Läufe berechnen interessante Informationen :-)
- Der Lauf mit $I = \{2\}$ belegt, dass nach Verlassen der Schleife stets $i = 42$ gilt.
- Nur der Lauf mit $I = \{1\}$ belegt aber, dass der äußere Test den inneren überflüssig macht :-(

Wie findet man einen geeigneten Loop Separator $I$ ???

## Idee 3: Narrowing

Sei $\underline{x}$ irgend eine Lösung von $(1)$ , d.h.

$$x_i \;\sqsupseteq\; f_i\,\underline{x} \;, \qquad i = 1, \ldots, n$$

Dann gilt für monotone $f_i$ ,

$$\underline{x} \;\sqsupseteq\; F\,\underline{x} \;\sqsupseteq\; F^2\,\underline{x} \;\sqsupseteq \ldots \sqsupseteq\; F^k\,\underline{x} \;\sqsupseteq \ldots$$

// Narrowing Iteration

## Idee 3: Narrowing

Sei $\underline{x}$ irgend eine Lösung von $(1)$, d.h.

$$x_i \sqsupseteq f_i \, \underline{x} \quad , \qquad i = 1, \ldots, n$$

Dann gilt für monotone $f_i$,

$$\underline{x} \sqsupseteq F \, \underline{x} \sqsupseteq F^2 \, \underline{x} \sqsupseteq \ldots \sqsupseteq F^k \, \underline{x} \sqsupseteq \ldots$$

// Narrowing Iteration

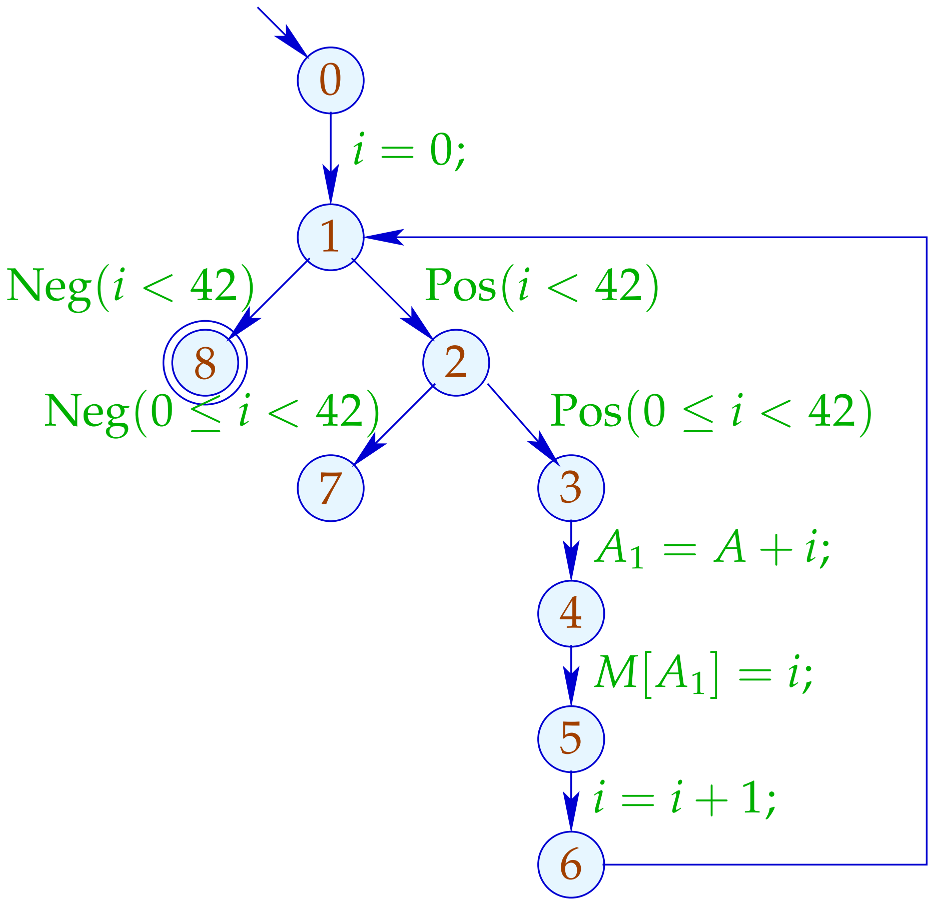
Jeder der Tupel $F^k \, \underline{x}$ ist eine Lösung von $(1)$ :-)

$\Longrightarrow$

Terminierung ist kein Problem mehr:
wir stoppen, wenn wir keine Lust mehr haben :-))

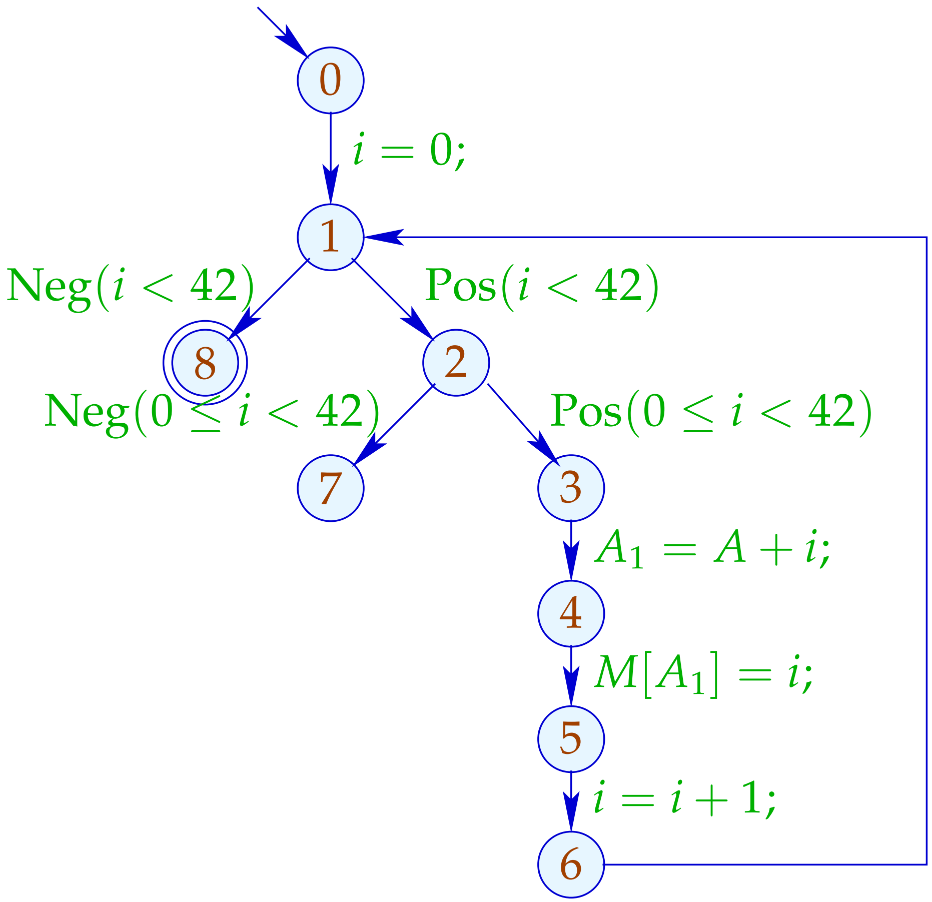
// Analoges gilt für RR-Iteration.

# Narrowing Iteration im Beispiel:



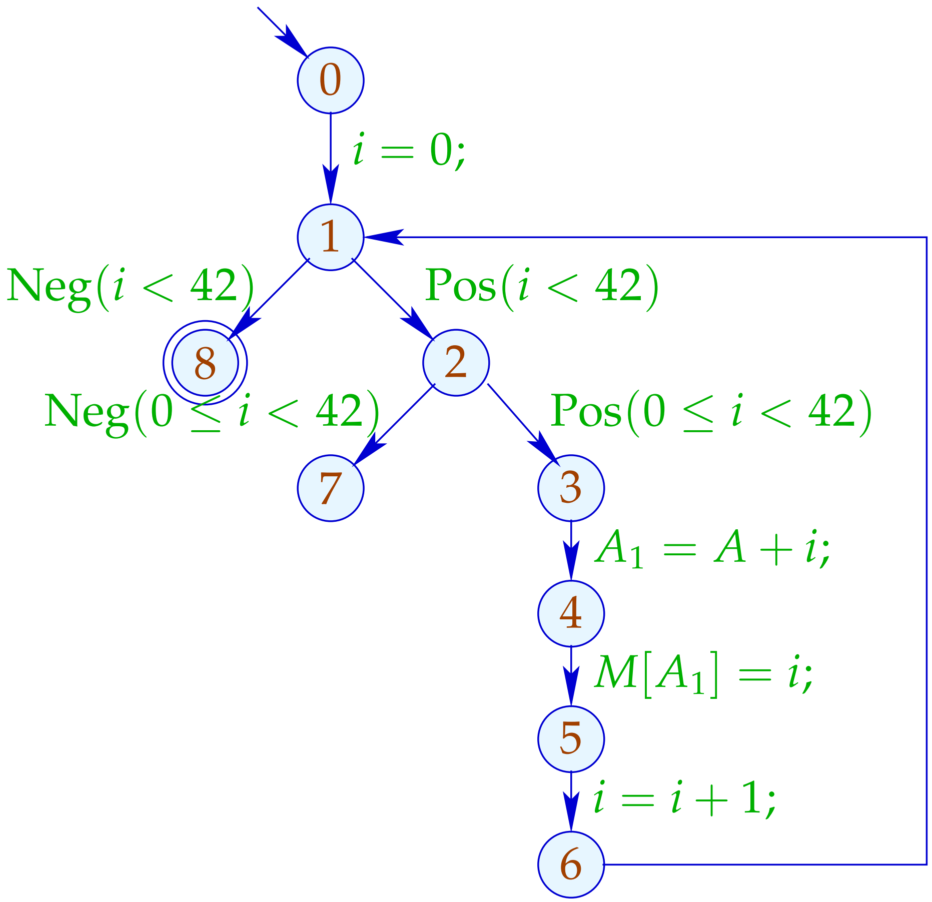

| | 0 | |
|---|---|---|
| | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ |
| 1 | 0 | $+\infty$ |
| 2 | 0 | $+\infty$ |
| 3 | 0 | $+\infty$ |
| 4 | 0 | $+\infty$ |
| 5 | 0 | $+\infty$ |
| 6 | 1 | $+\infty$ |
| 7 | 42 | $+\infty$ |
| 8 | 42 | $+\infty$ |

# Narrowing Iteration im Beispiel:




| | 0 | | 1 | |
|---|---|---|---|---|
| | $l$ | $u$ | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ |
| 1 | 0 | $+\infty$ | 0 | $+\infty$ |
| 2 | 0 | $+\infty$ | 0 | 41 |
| 3 | 0 | $+\infty$ | 0 | 41 |
| 4 | 0 | $+\infty$ | 0 | 41 |
| 5 | 0 | $+\infty$ | 0 | 41 |
| 6 | 1 | $+\infty$ | 1 | 42 |
| 7 | 42 | $+\infty$ | $\perp$ | |
| 8 | 42 | $+\infty$ | 42 | $+\infty$ |

# Narrowing Iteration im Beispiel:




| | 0 | | 1 | | 2 | |
|---|---|---|---|---|---|---|
| | $l$ | $u$ | $l$ | $u$ | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ |
| 1 | 0 | $+\infty$ | 0 | $+\infty$ | 0 | 42 |
| 2 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 3 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 4 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 5 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 6 | 1 | $+\infty$ | 1 | 42 | 1 | 42 |
| 7 | 42 | $+\infty$ | $\perp$ | | $\perp$ | |
| 8 | 42 | $+\infty$ | 42 | $+\infty$ | 42 | 42 |

## Diskussion:

$\longrightarrow$ Wir beginnen mit einer sicheren Approximation.

$\longrightarrow$ Wir finden, dass die innere Abfrage redundant ist :-)

$\longrightarrow$ Wir finden, dass nach der Iteration gilt: $i = 42$ :-))

$\longrightarrow$ Dazu war nicht erforderlich, einen optimalen Loop Separator zu berechnen :-)))

## Letzte Frage:

Müssen wir hinnehmen, dass Narrowing möglicherweise nicht terminiert ???

## 4. Idee: Beschleunigtes Narrowing

Nehmen wir an, wir hätten eine Lösung $\underline{x} = (x_1, \ldots, x_n)$ des Constraint-Systems:

$$x_i \sqsupseteq f_i\,(x_1, \ldots, x_n)\,, \quad i = 1, \ldots, n \qquad (1)$$

Dann schreiben betrachten wir das Gleichungssystem:

$$x_i = x_i \sqcap f_i\,(x_1, \ldots, x_n)\,, \quad i = 1, \ldots, n \qquad (4)$$

Offenbar gilt für monotone $f_i$ : $H^k\,\underline{x} = F^k\,\underline{x}$ :-)

wobei $H\,(x_1, \ldots, x_n) = (y_1, \ldots, y_n)\,, \quad y_i = x_i \sqcap f_i\,(x_1, \ldots, x_n)$.

In (4) ersetzen wir $\sqcap$ durch den neuen Operator $\overline{\sqcap}$ mit:

$$a_1 \sqcap a_2 \sqsubseteq a_1 \mathbin{\overline{\sqcap}} a_2 \sqsubseteq a_1$$

## ... für die Intervall-Analyse:

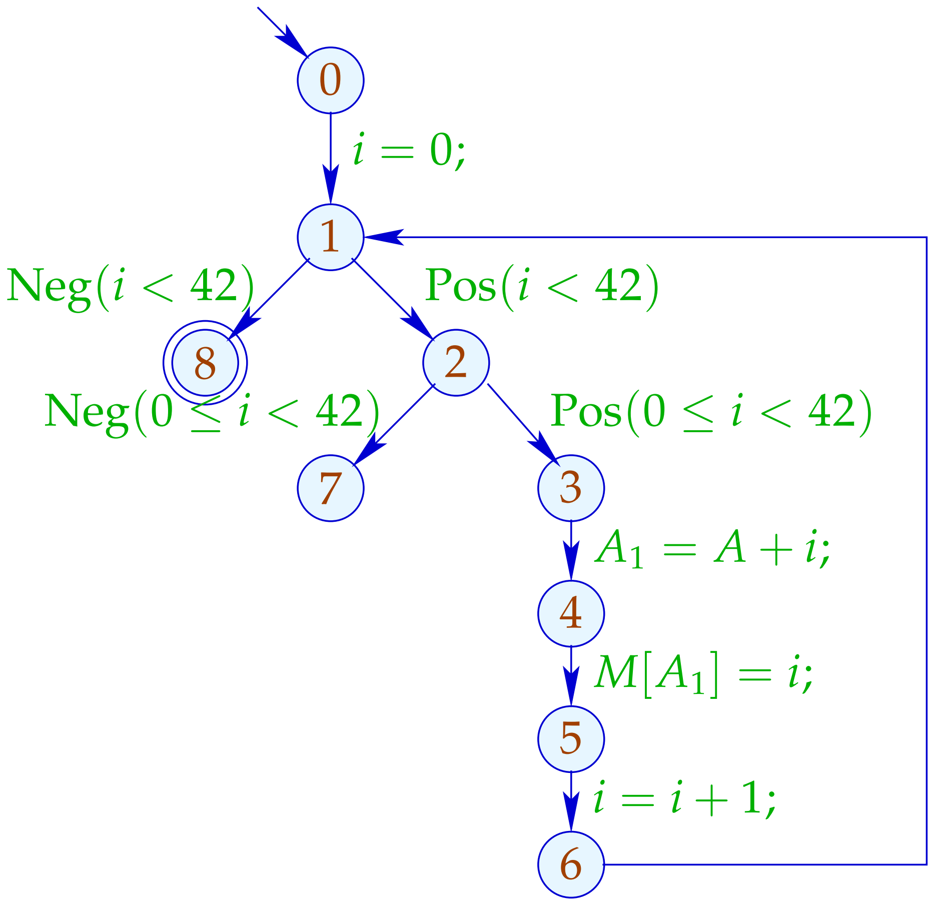
Wir konservieren endliche Intervall-Grenzen :-)

Deshalb $\quad \bot \sqcap D = D \sqcap \bot = \bot \quad$ und für $\quad D_1 \neq \bot \neq D_2$:

$$
\begin{array}{rcl}
(D_1 \sqcap D_2)\, x & = & (D_1\, x) \sqcap (D_2\, x) \qquad \text{wobei} \\
{[l_1, u_1]} \sqcap [l_2, u_2] & = & [l, u] \qquad \text{mit} \\
l & = & \begin{cases} l_2 & \text{falls} \quad l_1 = -\infty \\ l_1 & \text{sonst} \end{cases} \\
u & = & \begin{cases} u_2 & \text{falls} \quad u_1 = \infty \\ u_1 & \text{sonst} \end{cases}
\end{array}
$$

$\Longrightarrow \quad \sqcap \quad$ ist nicht kommutativ !!!

# Beschleunigtes Narrowing im Beispiel:




| | 0 | | 1 | | 2 | |
|---|---|---|---|---|---|---|
| | $l$ | $u$ | $l$ | $u$ | $l$ | $u$ |
| 0 | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ | $-\infty$ | $+\infty$ |
| 1 | 0 | $+\infty$ | 0 | $+\infty$ | 0 | 42 |
| 2 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 3 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 4 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 5 | 0 | $+\infty$ | 0 | 41 | 0 | 41 |
| 6 | 1 | $+\infty$ | 1 | 42 | 1 | 42 |
| 7 | 42 | $+\infty$ | $\perp$ | | $\perp$ | |
| 8 | 42 | $+\infty$ | 42 | $+\infty$ | 42 | 42 |

## Diskussion:

- $\longrightarrow$ Achtung: Widening liefert für nicht-monotone $f_i$ eine Lösung. Narrowing liefert dagegen nur für monotone $f_i$ eine Lösung!!
- $\longrightarrow$ Das beschleunigte Narrowing liefert (im Beispiel) das richtige Ergebnis :-)
- $\longrightarrow$ Erlaubt der neue Operator $\sqcap\!\!\!\!^{-}$ nur endlich viele Verbesserungen bei jedem Wert, kann Narrowing bis zur Stabilisierung durchgeführt werden.
- $\longrightarrow$ Für die Intervall-Analyse sind das maximal

$$\#\mathit{Punkte} \cdot (1 + 2 \cdot \#\mathit{Vars})$$

## 1.6 Pointer-Analyse

Fragen:

→ Sind zwei Adressen möglicherweise gleich?

→ Sind zwei Adressen definitiv gleich?

## 1.6 Pointer-Analyse

Fragen:

$\rightarrow$ Sind zwei Adressen möglicherweise gleich? May Alias

$\rightarrow$ Sind zwei Adressen definitiv gleich? Must Alias

$\Longrightarrow$ Alias-Analyse

## Die bisherigen Analysen ohne Alias-Information:

(1) Verfügbare Ausdrücke:

- Erweitere die Menge $Expr$ der Ausdrücke um die vorkommenden Loads $M[R]$ .

- Erweitere die Kanten-Effekte:

$$
\begin{array}{lll}
[\![x = e;]\!]^\sharp\, A & = & (A \cup \{e\})\backslash Expr_x \\
[\![x = M[R];]\!]^\sharp\, A & = & (A \cup \{M[R]\})\backslash Expr_x \\
[\![M[R] = x;]\!]^\sharp\, A & = & A\backslash Loads
\end{array}
$$

(2) Werte von Variablen:

- Erweitere die Menge $Expr$ der Ausdrücke um die vorkommenden Loads $M[R]$ .

- Erweitere die Kanten-Effekte:

$$\begin{aligned} [\![x = M[R];]\!]^\sharp \, V \, e &= \begin{cases} \{x\} & \text{falls} \quad e = M[R] \\ V \, e \backslash \{x\} & \text{sonst} \end{cases} \\ [\![M[R] = x;]\!]^\sharp \, V &= V \end{aligned}$$

## (3) Konstantenpropagation:

- Erweitere den abstrakten Zustand um einen abstrakten Speicher $M$
- Führe Speicher-Operationen mit bekannten Adressen aus!

$$\begin{array}{lll}
[\![x = M[R];]\!]^\sharp\,(D, M) & = & \begin{cases} (D \oplus \{x \mapsto M\,a\}, M) & \text{falls} \quad D\,R = a \sqsubset \top \\ (D \oplus \{x \mapsto \top\}, M) & \text{sonst} \end{cases} \\
[\![M[R] = x;]\!]^\sharp\,(D, M) & = & \begin{cases} (D, M \oplus \{a \mapsto D\,x\}) & \text{falls} \quad D\,R = a \sqsubset \top \\ (D, \underline{\top}) & \text{sonst} \end{cases} \qquad \text{wobei} \\
\underline{\top}\,a & = & \top \qquad\qquad (a \in \mathbb{N})
\end{array}$$

## Probleme:

- Adressen sind aus $\mathbb{N}$ :-(

  Es gibt zwar keine unendliche aufsteigende Ketten, aber ...

- Exakte Adressen sind zur Compilezeit selten bekannt :-(
- Am selben Programmpunkt wird i.a. auf mehrere Adressen zugegriffen ...
- Abspeichern an unbekannter Adresse zerstört alle Information $M$ :-(

$\Longrightarrow$ Konstanten-Propagation versagt :-(

$\Longrightarrow$ Speicherzugriffe/Pointer zerstören Präzision :-(

## Vereinfachung:

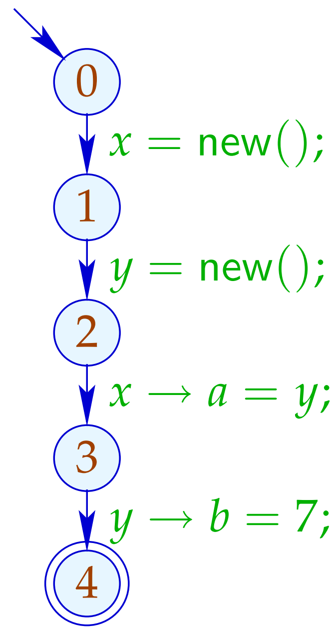
- Wir betrachten Pointer auf Strukturen mit Komponenten $a, b$ :-)
- Wir verzichten auf Wohl-Getyptheit dieser Komponenten.
- Neue Statements:

$$
\begin{array}{lll}
x = \mathsf{new}(); & // & \text{Allokation eines neuen Paars} \\
x = R \to a; & // & \text{Laden einer Komponente} \\
R \to a = x; & // & \text{Setzen einer Komponente}
\end{array}
$$

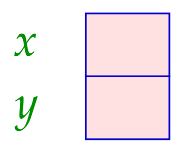
- Wir verzichten auf Pointer-Arithmetik :-)

# Einfaches Beispiel:

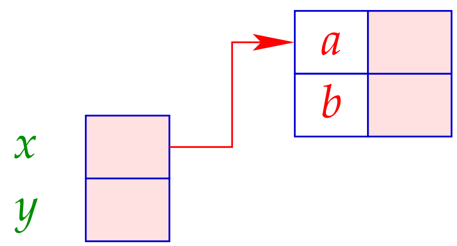
$$x = \mathsf{new}();$$
$$y = \mathsf{new}();$$
$$x \rightarrow a = y;$$
$$y \rightarrow b = 7;$$



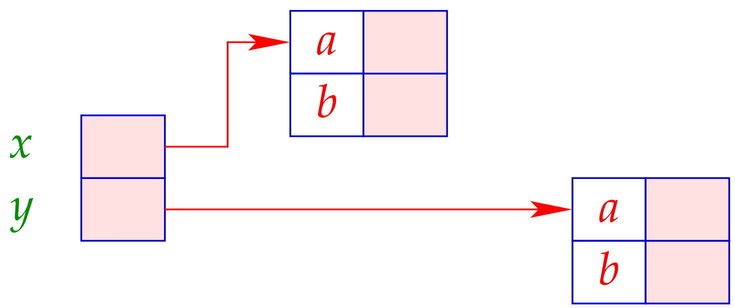
## Die Semantik:



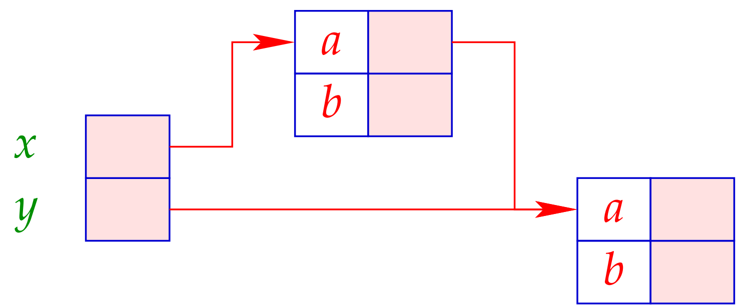
## Die Semantik:



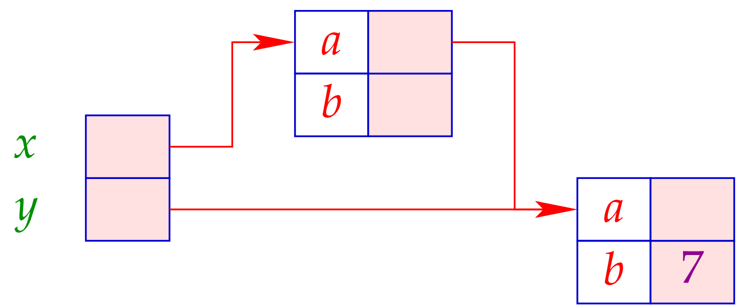
## Die Semantik:



## Die Semantik:



# Die Semantik:



# Schwierigeres Beispiel:

$r = \text{Null};$

$\text{while } (t \neq \text{Null}) \;\{$

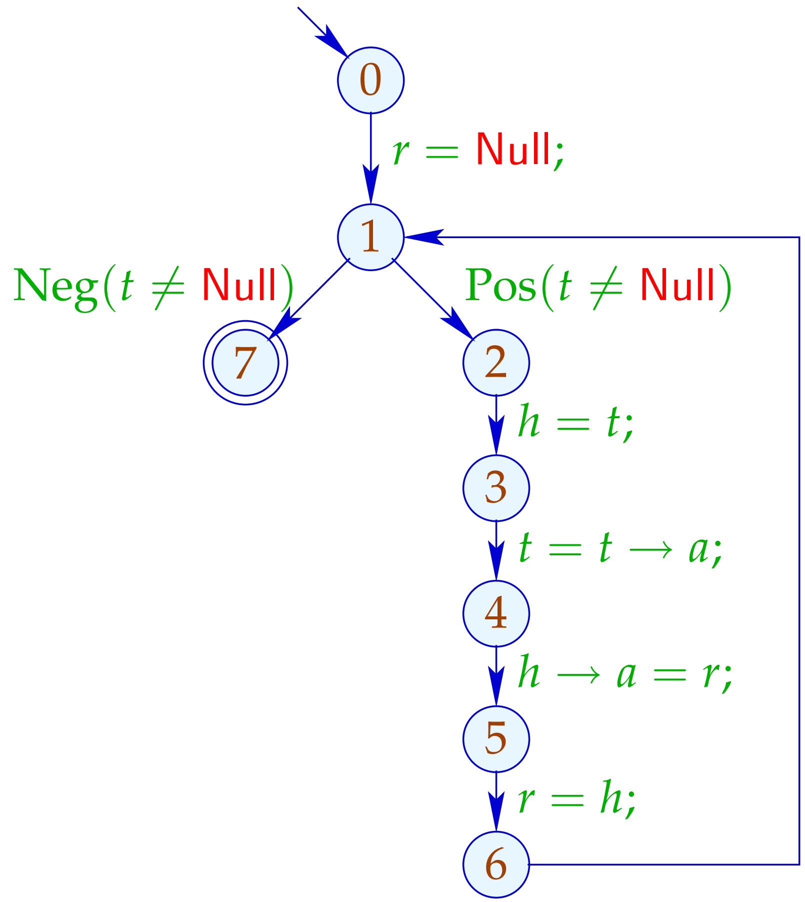
$\quad h = t;$

$\quad t = t \rightarrow a;$

$\quad h \rightarrow a = r;$

$\quad r = h;$

$\}$



## Konkrete Semantik:

Ein Speicher ist jetzt eine endliche Ansammlung von Paaren.

Nach $h$ new-Operationen haben wir:

$$
\begin{array}{lcll}
Addr_h & = & \{\mathsf{ref}\ a \mid 0 \leq a < h\} & // \quad \text{Adressen} \\
Val_h & = & Addr_h \cup \mathbb{Z} & // \quad \text{Werte} \\
Store_h & = & (Addr_h \times \{a, b\}) \to Val_h & // \quad \text{Speicher} \\
State_h & = & (Vars \to Val_h) \times Store_h & // \quad \text{Zustände}
\end{array}
$$

Der Einfachheit setzen wir: $\quad 0 = \mathsf{Null}$

Sei $\quad (\rho, \mu) \in \mathit{State}_h$ . Dann erhalten wir für die neuen Kanten:

$$\begin{array}{lll}
[\![x = \mathsf{new}();]\!]\,(\rho,\mu) & = & (\rho \oplus \{x \mapsto \mathsf{ref}\,h\}, \\
 & & \;\mu \oplus \{(\mathsf{ref}\,h).a \mapsto 0, (\mathsf{ref}\,h).b \mapsto 0\}) \\
[\![x = R \to a;]\!]\,(\rho,\mu) & = & (\rho \oplus \{x \mapsto \mu\,((\rho\,R).a)\}, \mu) \\
[\![R \to a = x;]\!]\,(\rho,\mu) & = & (\rho, \mu \oplus \{(\rho\,R).a \mapsto \rho\,x\})
\end{array}$$

## Achtung:

Diese Semantik ist zu detailliert, weil sie mit absoluten Adressen rechnet. Die beiden Programme:

$$x = \mathsf{new}(); \qquad\qquad y = \mathsf{new}();$$
$$y = \mathsf{new}(); \qquad\qquad x = \mathsf{new}();$$

werden nicht als äquivalent betrachtet !!?

## Ausweg:

Definiere Äquivalenz bis auf Permutation von Adressen :-)

## Alias-Analyse 1. Idee:

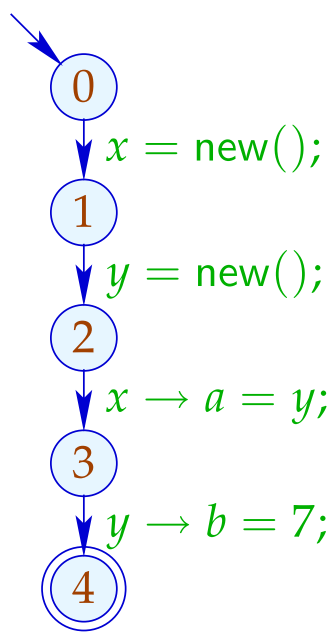
- Unterscheide endlich viele verschiedene Klassen von Objekten im Speicher.
- Benutze Mengen von Adressen als abstrakte Werte!

$\Longrightarrow$ Points-to-Analyse

$$
\begin{array}{lcll}
\mathit{Addr}^\sharp & = & \mathit{Edges} & // \quad \text{Erzeugungs-Kanten} \\
\mathit{Val}^\sharp & = & 2^{\mathit{Addr}^\sharp} & // \quad \text{Abstrakte Werte} \\
\mathit{Store}^\sharp & = & (\mathit{Addr}^\sharp \times \{a, b\}) \to \mathit{Val}^\sharp & // \quad \text{abstrakter Speicher} \\
\mathit{State}^\sharp & = & (\mathit{Vars} \to \mathit{Val}^\sharp) \times \mathit{Store}^\sharp & // \quad \text{Zustände}
\end{array}
$$

// vollständiger Verband !!!

... im einfachen Beispiel:




| | $x$ | $y$ | $(0,1).a$ |
|---|---|---|---|
| 0 | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 1 | $\{(0,1)\}$ | $\emptyset$ | $\emptyset$ |
| 2 | $\{(0,1)\}$ | $\{(1,2)\}$ | $\emptyset$ |
| 3 | $\{(0,1)\}$ | $\{(1,2)\}$ | $\{(1,2)\}$ |
| 4 | $\{(0,1)\}$ | $\{(1,2)\}$ | $\{(1,2)\}$ |

# Die Kanten-Effekte:

$$
\begin{array}{lll}
[\![(\_, ;, \_)]\!]^\sharp\,(D, M) & = & (D, M) \\
[\![(\_, \mathrm{Pos}(e), \_)]\!]^\sharp\,(D, M) & = & (D, M) \\
[\![(\_, x = y;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto D\,y\}, M) \\
[\![(\_, x = e;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \emptyset\}, M) \qquad , \qquad e \not\in \mathit{Vars} \\
\\
[\![(u, x = \mathsf{new}();, v)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \{(u, v)\}\}, M) \\
[\![(\_, x = R \to a;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \bigcup\{M(f.a) \mid f \in D\,R\}\}, M) \\
[\![(\_, R \to a = x;, \_)]\!]^\sharp\,(D, M) & = & (D, M \oplus \{f.a \mapsto (M\,(f.a) \cup D\,x) \mid f \in D\,R\})
\end{array}
$$

## Achtung:

- Den Wert Null haben wir nicht mit-modelliert. Dereferenzieren von Null kann darum nicht entdeckt werden :-(
- Destruktive Updates sind nur von Variablen möglich, nicht im Speicher!

  $\Longrightarrow$ keine Information, falls Speicher-Objekte nicht vorinitialisiert sind :-((
- Die Kanten-Effekte hängen jetzt von der ganzen Kante ab.

  Die Analyse lässt sich so nicht gegenüber der Referenz-Semantik als korrekt erweisen :-(

  Zur Korrektheit muss die konkrete Semantik mit zusätzlicher Information instrumentiert werden, die vermerkt, an welchem Programmpunkt eine Adresse erzeugt wurde.