

Idee 3: Narrowing

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

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

Dann gilt für monotone f_i ,

$$\underline{x} \sqsupseteq F \underline{x} \sqsupseteq F^2 \underline{x} \sqsupseteq \dots \sqsupseteq F^k \underline{x} \sqsupseteq \dots$$

// Narrowing Iteration

Idee 3: Narrowing

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

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

Dann gilt für monotone f_i ,

$$\underline{x} \sqsupseteq F \underline{x} \sqsupseteq F^2 \underline{x} \sqsupseteq \dots \sqsupseteq F^k \underline{x} \sqsupseteq \dots$$

// Narrowing Iteration

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

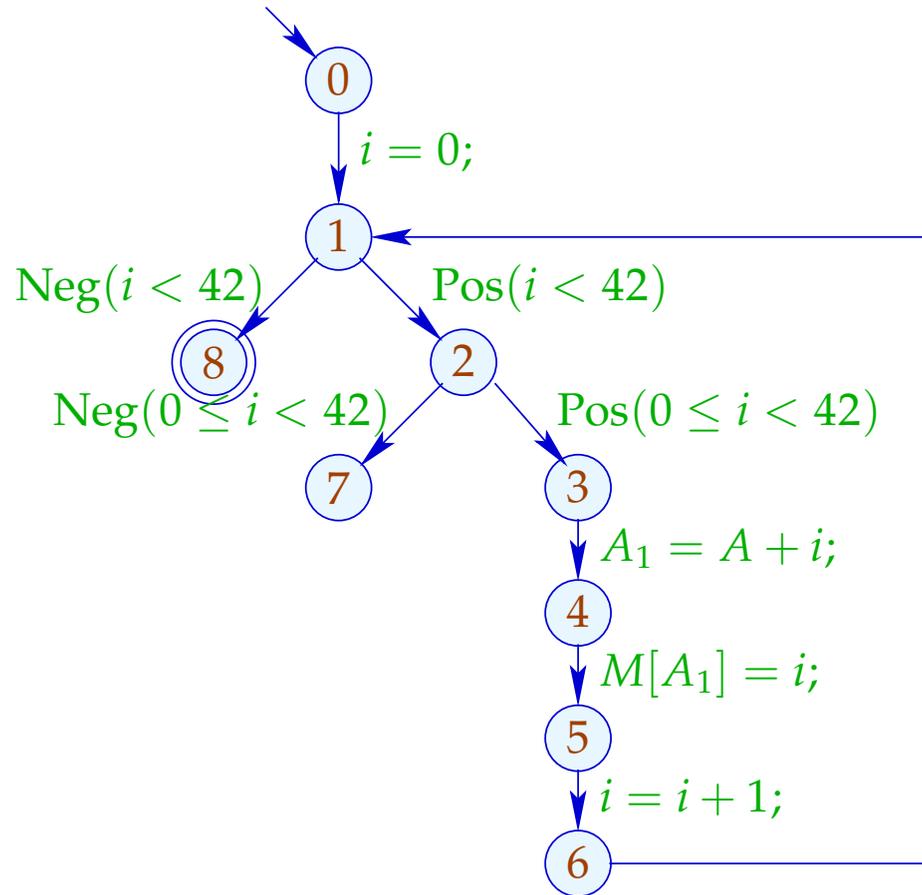


Terminierung ist kein Problem mehr:

wir stoppen, wenn wir keine Lust mehr haben :-))

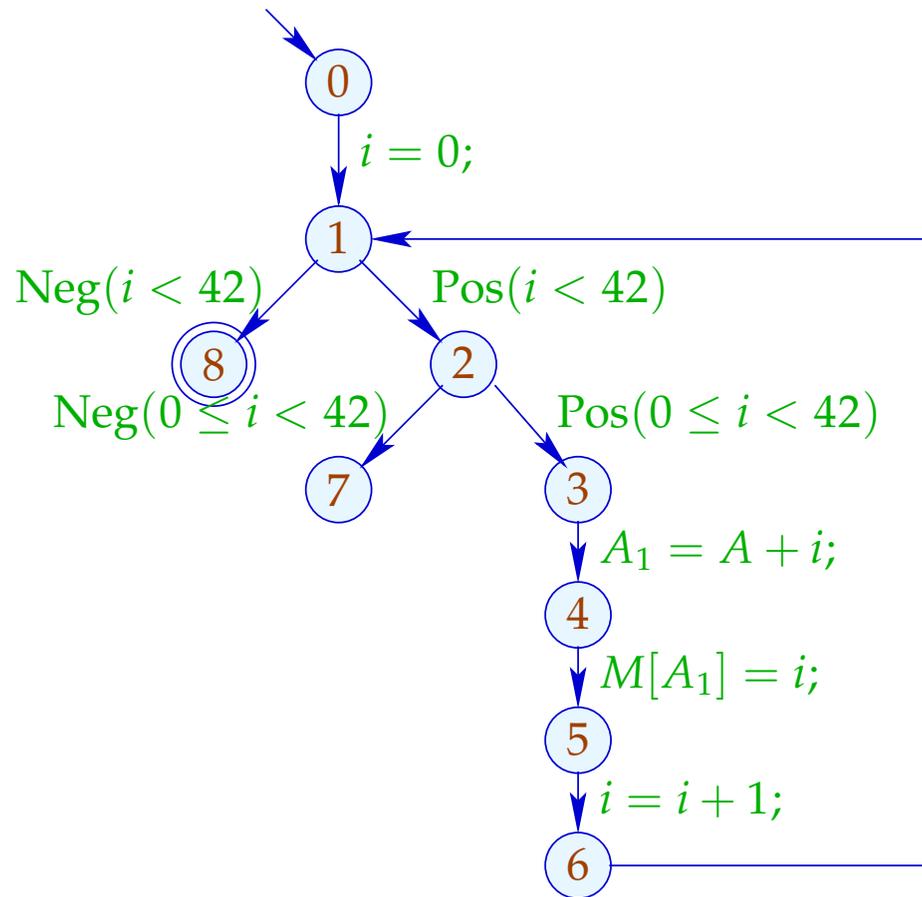
// Analoges gilt für RR-Iteration.

Narrowing Iteration im Beispiel:



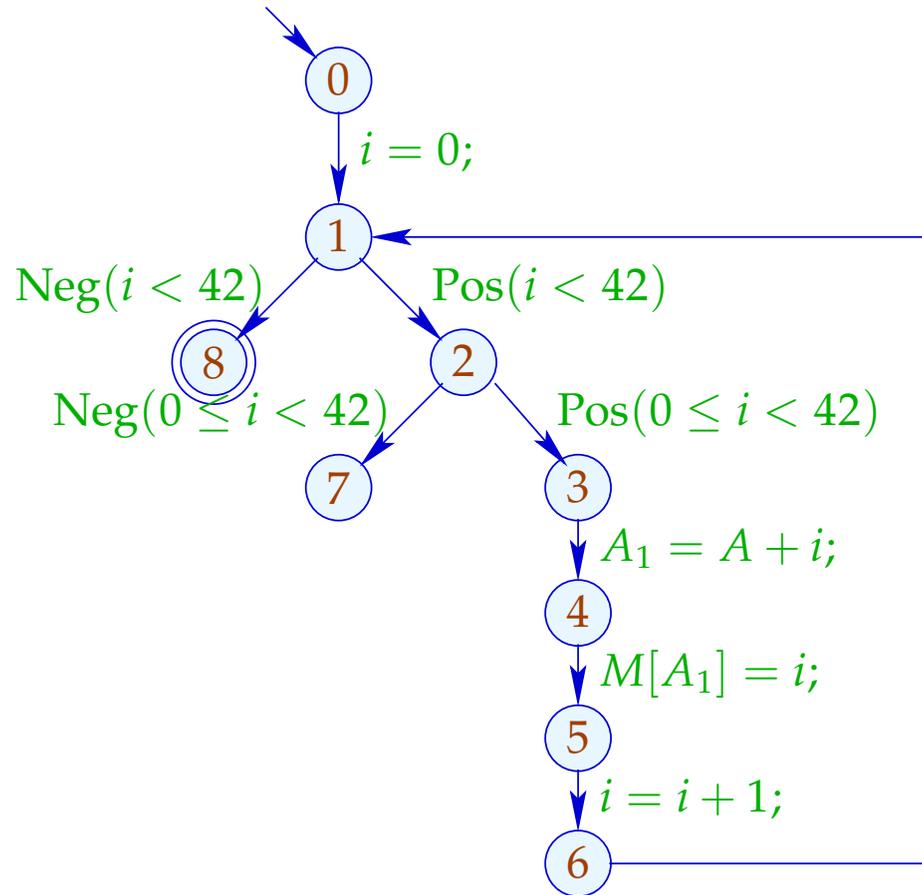
	0	
	<i>l</i>	<i>u</i>
0	$-\infty$	$+\infty$
1	0	$+\infty$
2	0	$+\infty$
3	0	$+\infty$
4	0	$+\infty$
5	0	$+\infty$
6	1	$+\infty$
7	42	$+\infty$
8	42	$+\infty$

Narrowing Iteration im Beispiel:



	0		1	
	<i>l</i>	<i>u</i>	<i>l</i>	<i>u</i>
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	
	<i>l</i>	<i>u</i>	<i>l</i>	<i>u</i>	<i>l</i>	<i>u</i>
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:

- Wir beginnen mit einer sicheren Approximation.
- Wir finden, dass die innere Abfrage redundant ist :-)
- Wir finden, dass nach der Iteration gilt: $i = 42$:-))
- 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 $\underline{x} = (x_1, \dots, x_n)$ des Constraint-Systems:

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

Dann schreiben betrachten wir das Gleichungssystem:

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

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

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

In (4) ersetzen wir \sqcap durch den neuen Operator \sqbar mit:

$$a_1 \sqcap a_2 \sqsubseteq a_1 \sqbar a_2 \sqsubseteq a_1$$

... für die Intervall-Analyse:

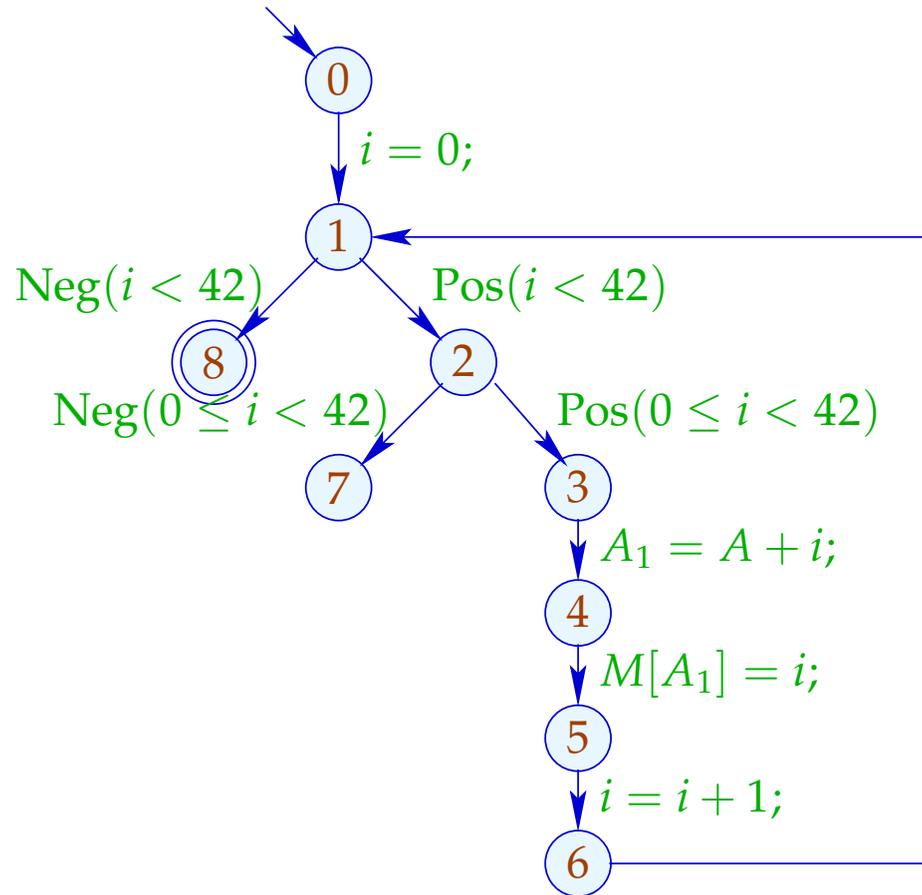
Wir konservieren endliche Intervall-Grenzen :-)

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

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

$\implies \sqcap$ ist nicht kommutativ !!!

Beschleunigtes Narrowing im Beispiel:



	0		1		2	
	<i>l</i>	<i>u</i>	<i>l</i>	<i>u</i>	<i>l</i>	<i>u</i>
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:

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

$$\#Punkte \cdot (1 + 2 \cdot \#Vars)$$

1.6 Pointer-Analyse

Fragen:

- Sind zwei Adressen **möglicherweise** gleich?
- Sind zwei Adressen **definitiv** gleich?

1.6 Pointer-Analyse

Fragen:

- Sind zwei Adressen **möglicherweise** gleich? **May Alias**
- Sind zwei Adressen **definitiv** gleich? **Must Alias**

⇒ **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:

$$\llbracket x = e; \rrbracket^\# A = (A \cup \{e\}) \setminus Expr_x$$

$$\llbracket x = M[R]; \rrbracket^\# A = (A \cup \{M[R]\}) \setminus Expr_x$$

$$\llbracket M[R] = x; \rrbracket^\# A = A \setminus Loads$$

(2) Werte von Variablen:

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

$$\llbracket x = M[R]; \rrbracket^\# V e = \begin{cases} \{x\} & \text{falls } e = M[R] \\ V e \setminus \{x\} & \text{sonst} \end{cases}$$

$$\llbracket M[R] = x; \rrbracket^\# V = V$$

(3) Konstantenpropagation:

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

$$\begin{aligned}
 \llbracket x = M[R]; \rrbracket^\# (D, M) &= \begin{cases} (D \oplus \{x \mapsto M a\}, M) & \text{falls } D R = a \sqsubset \top \\ (D \oplus \{x \mapsto \top\}, M) & \text{sonst} \end{cases} \\
 \llbracket M[R] = x; \rrbracket^\# (D, M) &= \begin{cases} (D, M \oplus \{a \mapsto D x\}) & \text{falls } D R = a \sqsubset \top \\ (D, \perp) & \text{sonst} \end{cases} \quad \text{wobei} \\
 \perp a &= \top \quad (a \in \mathbb{N})
 \end{aligned}$$

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 :-(

⇒⇒ Konstanten-Propagation versagt :-(
⇒⇒ Speicherzugriffe/Pointer **zerstören Präzision** :-)

Vereinfachung:

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

$x = \text{new}();$ // Allokation eines neuen Paares

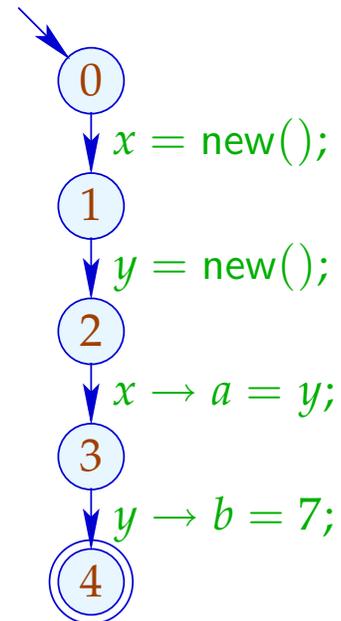
$x = R \rightarrow a;$ // Laden einer Komponente

$R \rightarrow a = x;$ // Setzen einer Komponente

- Wir verzichten auf **Pointer-Arithmetik** :-)

Einfaches Beispiel:

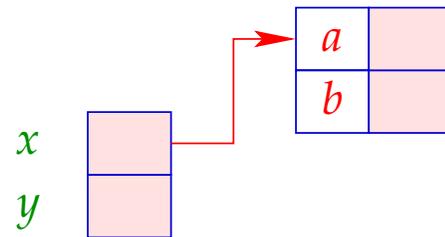
```
 $x = \text{new}();$   
 $y = \text{new}();$   
 $x \rightarrow a = y;$   
 $y \rightarrow b = 7;$ 
```



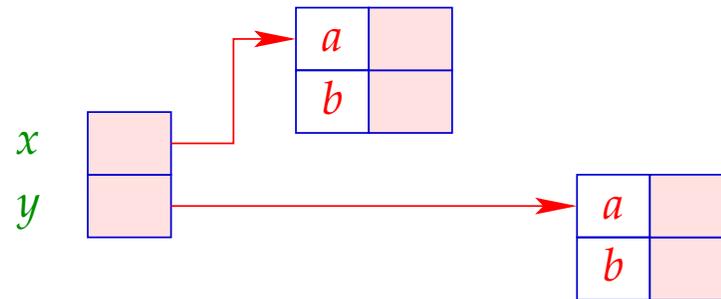
Die Semantik:

x	
y	

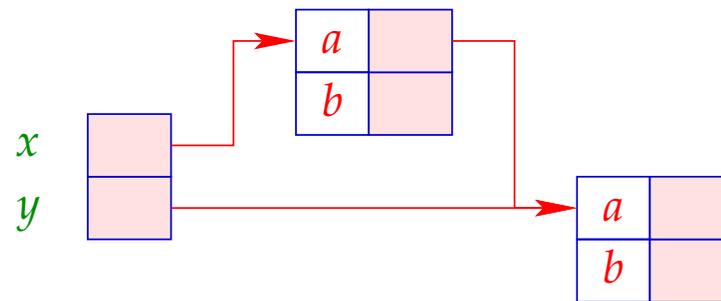
Die Semantik:



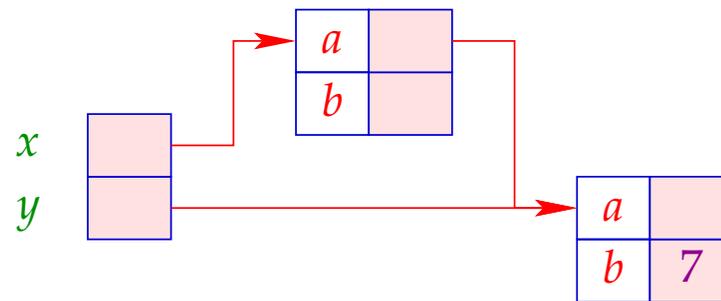
Die Semantik:



Die Semantik:

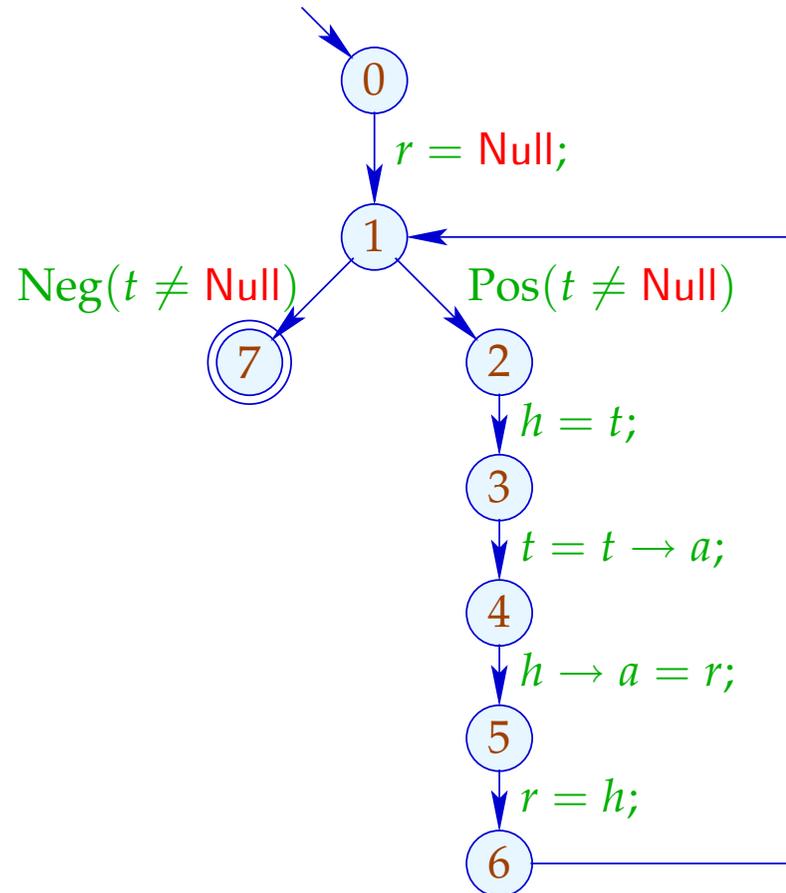


Die Semantik:



Schwierigeres Beispiel:

```
r = Null;  
while (t ≠ Null) {  
    h = t;  
    t = t → a;  
    h → a = r;  
    r = h;  
}
```



Konkrete Semantik:

Ein Speicher ist jetzt eine **endliche** Ansammlung von Paaren.

Nach h new-Operationen haben wir:

$$Addr_h = \{\text{ref } a \mid 0 \leq a < h\} \quad // \text{ Adressen}$$

$$Val_h = Addr_h \cup \mathbb{Z} \quad // \text{ Werte}$$

$$Store_h = (Addr_h \times \{a, b\}) \rightarrow Val_h \quad // \text{ Speicher}$$

$$State_h = (Vars \rightarrow Val_h) \times Store_h \quad // \text{ Zustände}$$

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

Sei $(\rho, \mu) \in State_h$. Dann erhalten wir für die neuen Kanten:

$$\begin{aligned} \llbracket x = \text{new}(); \rrbracket (\rho, \mu) &= (\rho \oplus \{x \mapsto \text{ref } h\}, \\ &\quad \mu \oplus \{(\text{ref } h).a \mapsto 0, (\text{ref } h).b \mapsto 0\}) \\ \llbracket x = R \rightarrow a; \rrbracket (\rho, \mu) &= (\rho \oplus \{x \mapsto \mu((\rho R).a)\}, \mu) \\ \llbracket R \rightarrow a = x; \rrbracket (\rho, \mu) &= (\rho, \mu \oplus \{(\rho R).a \mapsto \rho x\}) \end{aligned}$$

Achtung:

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

$x = \text{new}();$	$y = \text{new}();$
$y = \text{new}();$	$x = \text{new}();$

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!

⇒ Points-to-Analyse

$Addr^\# = Edges$ // Erzeugungs-Kanten

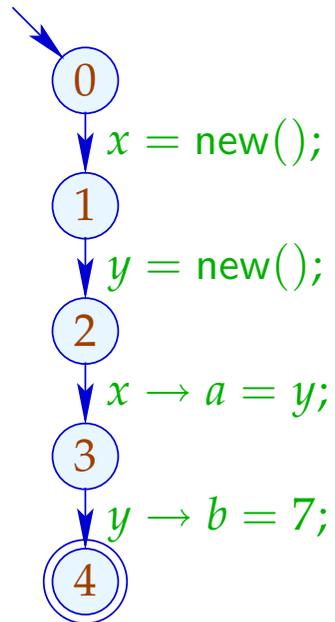
$Val^\# = 2^{Addr^\#}$ // Abstrakte Werte

$Store^\# = (Addr^\# \times \{a, b\}) \rightarrow Val^\#$ // abstrakter Speicher

$State^\# = (Vars \rightarrow Val^\#) \times Store^\#$ // Zustände

// 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:

$$\llbracket (_, ;, _) \rrbracket^\# (D, M) = (D, M)$$

$$\llbracket (_, \text{Pos}(e), _) \rrbracket^\# (D, M) = (D, M)$$

$$\llbracket (_, x = y; , _) \rrbracket^\# (D, M) = (D \oplus \{x \mapsto D y\}, M)$$

$$\llbracket (_, x = e; , _) \rrbracket^\# (D, M) = (D \oplus \{x \mapsto \emptyset\}, M) \quad , \quad e \notin \text{Vars}$$

$$\llbracket (u, x = \text{new}(); , v) \rrbracket^\# (D, M) = (D \oplus \{x \mapsto \{(u, v)\}\}, M)$$

$$\llbracket (_, x = R \rightarrow a; , _) \rrbracket^\# (D, M) = (D \oplus \{x \mapsto \cup \{M(f.a) \mid f \in D R\}\}, M)$$

$$\llbracket (_, R \rightarrow a = x; , _) \rrbracket^\# (D, M) = (D, M \oplus \{f.a \mapsto (M(f.a) \cup D x) \mid f \in D R\})$$

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!

⇒ 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.

...

- Wir berechnen mögliche Points-to-Information.
- Daraus können wir May-Alias-Information gewinnen.
- Die Analyse kann jedoch ziemlich aufwendig sein (ohne viel raus zu kriegen :-)
- Separate Information für jeden Programmpunkt ist möglicherweise nicht nötig ??