1.6 Pointer-Analyse

Fragen:

- $\longrightarrow$ Sind zwei Adressen möglicherweise gleich?
- $\longrightarrow$ 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:

$$
\begin{array}{lll}
[\![x = e;]\!]^\sharp\, A & = & (A \cup \{e\}) \backslash Expr_x \\
[\![x = M[e];]\!]^\sharp\, A & = & (A \cup \{e, M[e]\}) \backslash Expr_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A & = & (A \cup \{e_1, e_2\}) \backslash Loads
\end{array}
$$

(2) Werte von Variablen:

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

- Erweitere die Kanten-Effekte:

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

$$[\![M[e_1] = e_2;]\!]^\sharp\, V\, e' \quad = \begin{cases} \emptyset & \text{falls} \quad e' \in \{e_1, e_2\} \\ V\, e' & \text{sonst} \end{cases}$$

(3) Konstantenpropagation:

- Erweitere den abstrakten Zustand um einen abstrakten Speicher M

- Führe Speicher-Operationen mit bekannten Adressen aus!

$$\llbracket x = M[e]; \rrbracket^\sharp (D, M) = \begin{cases} (D \oplus \{x \mapsto M\, a\}, M) & \text{falls} \\ & \llbracket e \rrbracket^\sharp\, D = a \sqsubset \top \\ (D \oplus \{x \mapsto \top\}, M) & \text{sonst} \end{cases}$$

$$\llbracket M[e_1] = e_2; \rrbracket^\sharp (D, M) = \begin{cases} (D, M \oplus \{a \mapsto \llbracket e_2 \rrbracket^\sharp D\}) & \text{falls} \\ & \llbracket e_1 \rrbracket^\sharp\, D = a \sqsubset \top \\ (D, \underline{\top}) & \text{sonst} \end{cases} \quad \text{wobei}$$

$$\underline{\top}\, a = \top \qquad (a \in \mathbb{N})$$

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:

$$x = \mathsf{new}(); \quad // \quad \text{Allokation eines neuen Paars}$$

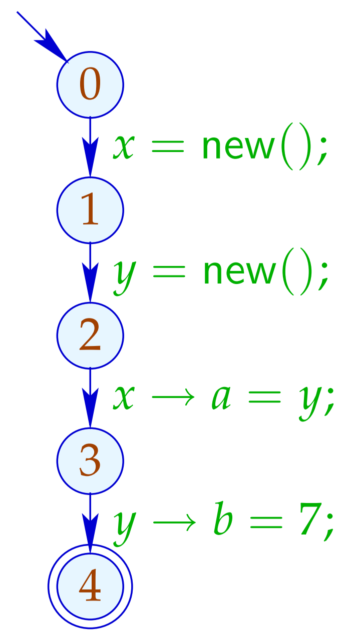
$$x = R \to a; \quad // \quad \text{Laden einer Komponente}$$

$$R \to a = x; \quad // \quad \text{Setzen einer Komponente}$$

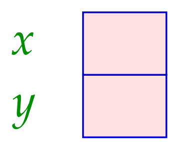
- Wir verzichten auf Pointer-Arithmetik :-)

Einfaches Beispiel:

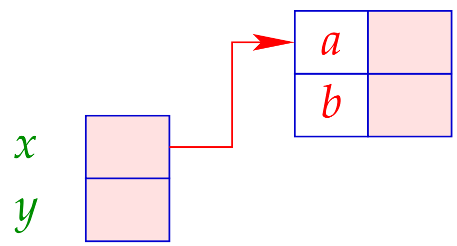
$$x = \text{new}();$$
$$y = \text{new}();$$
$$x \rightarrow a = y;$$
$$y \rightarrow b = 7;$$



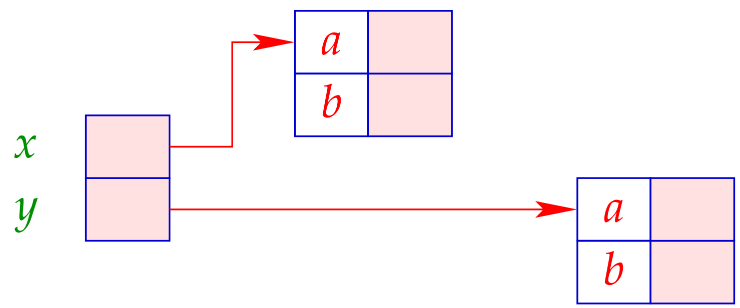
Die Semantik:



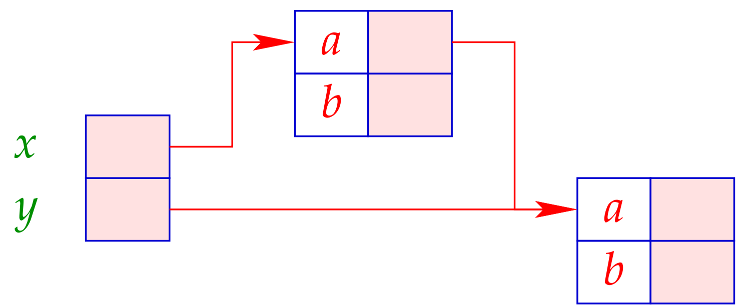
Die Semantik:



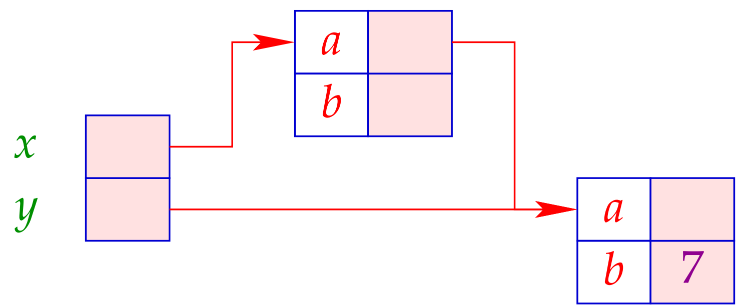
Die Semantik:



Die Semantik:



Die Semantik:



Schwierigeres Beispiel:

$r = \mathsf{Null};$

$\mathsf{while}\ (t \neq \mathsf{Null})\ \{$

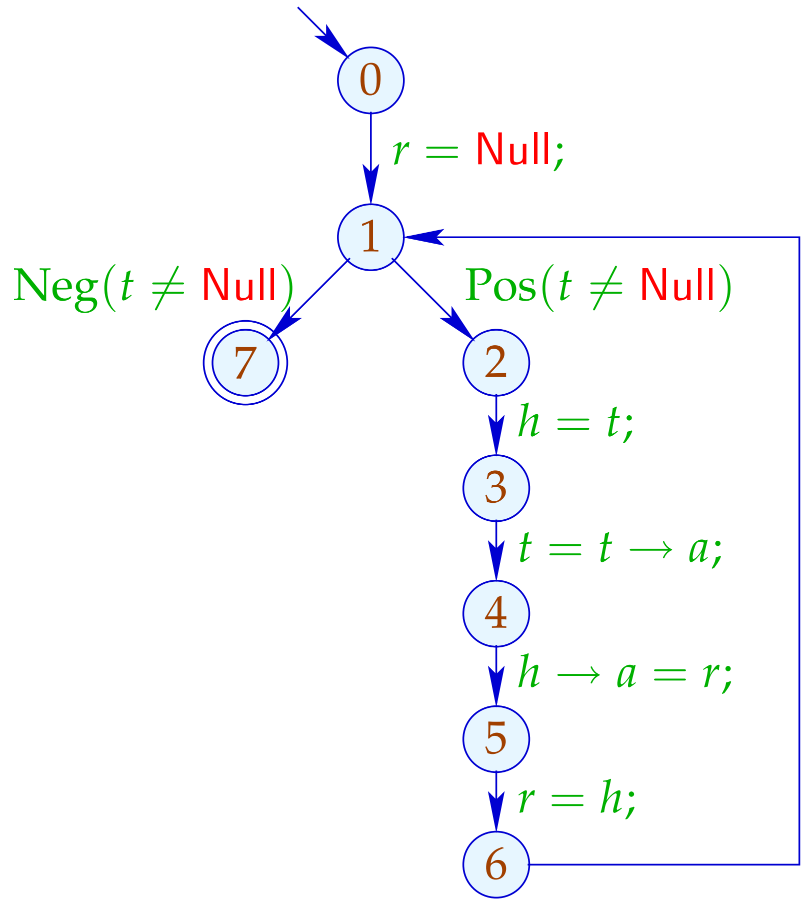
$\quad h = t;$

$\quad t = t \rightarrow a;$

$\quad h \rightarrow a = r;$

$\quad r = h;$

$\}$



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: $0 = \mathsf{Null}$

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

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

Achtung:

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

$$\begin{array}{lcl} x = \mathsf{new}(); & \qquad\qquad & y = \mathsf{new}(); \\ y = \mathsf{new}(); & & x = \mathsf{new}(); \end{array}$$

werden nicht als äquivalent betrachtet !!?

Ausweg:

Definiere Äquivalenz bis auf Permutation von Adressen :-)

Alias-Analyse 1. Idee:

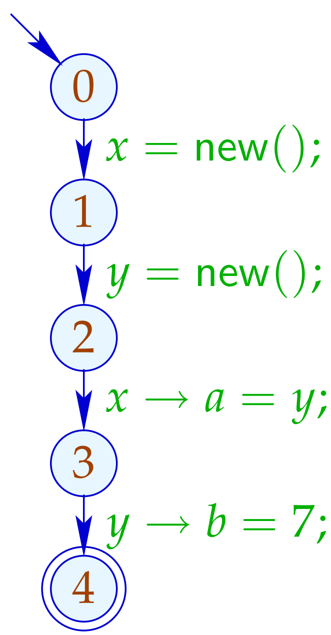
- Unterscheide endlich viele verschiedene Klassen von Objekten im Speicher.
- Benutze Mengen von Adressen als abstrakte Werte!

$\Longrightarrow$ 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:

$$
\begin{array}{lcl}
[\![(\_, ;, \_)]\!]^\sharp\,(D, M) & = & (D, M) \\
[\![(\_, \mathrm{Pos}(e), \_)]\!]^\sharp\,(D, M) & = & (D, M) \\
[\![(\_, x = y;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto D\,y\}, M) \\
[\![(\_, x = e;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \emptyset\}, M) \qquad , \qquad e \notin \mathit{Vars} \\
\\
[\![(u, x = \mathsf{new}();, v)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \{(u, v)\}\}, M) \\
[\![(\_, x = R \rightarrow a;, \_)]\!]^\sharp\,(D, M) & = & (D \oplus \{x \mapsto \bigcup\{M(f.a) \mid f \in D\,R\}\}, M) \\
[\![(\_, R \rightarrow a = x;, \_)]\!]^\sharp\,(D, M) & = & (D, M \oplus \{f.a \mapsto (M\,(f.a) \cup D\,x) \mid f \in D\,R\})
\end{array}
$$

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.

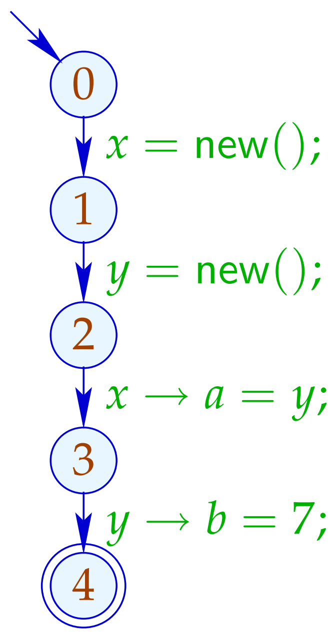
...

- 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 ??

2. Idee:

Berechne für jede Variable und jede Adresse einen Wert, der die Werte an sämtlichen Programmpunkten sicher approximiert!

... im einfachen Beispiel:



x	$\{(0,1)\}$
y	$\{(1,2)\}$
$(0,1).a$	$\{(1,2)\}$
$(0,1).b$	$\emptyset$

Jede Kante (u, lab, v) gibt Anlass zu Ungleichungen:

lab	*Ungleichung*
$x = y;$	$\mathcal{P}[x] \supseteq \mathcal{P}[y]$
$x = \mathsf{new}();$	$\mathcal{P}[x] \supseteq \{(u, v)\}$
$x = R \to a;$	$\mathcal{P}[x] \supseteq \bigcup\{\mathcal{P}[f.a] \mid f \in \mathcal{P}[R]\}$
$R \to a = x;$	$\mathcal{P}[f.a] \supseteq (f \in \mathcal{P}[R])\, ?\, \mathcal{P}[x] \;:\; \emptyset$ für alle $f \in Addr^\sharp$

Andere Kanten haben keinen Effekt :-)

Diskussion:

- Das resultierende Ungleichungssystem ist $\mathcal{O}(k \cdot n)$ bei k abstrakten Adressen und n Kanten :-(
- Die Anzahl eventuell notwendiger Iterationen ist $\mathcal{O}(k)$ ...
- Die berechnete Information ist möglicherweise immer noch zu präzise !!?
- Zur Korrektheit einer Lösung $s^\sharp \in \mathit{States}^\sharp$ des Ungleichungssystems zeigt man: