Der Algorithmus:

$$
\begin{array}{l}
W = \{x_1, \ldots, x_n\}; \\
\textsf{while } (W \neq \emptyset) \ \{ \\
\qquad \begin{array}{lll} x_i & = & \textsf{extract}\, W; \\ t & = & f_i\, \textsf{eval}; \end{array} \\
\qquad \textsf{if } !(t \sqsubseteq D[x_i]) \ \{ \\
\qquad\qquad \begin{array}{lll} D[x_i] & = & D[x_i] \sqcup t; \\ W & = & W \cup I[x_i]; \end{array} \\
\qquad \} \\
\}
\end{array}
$$

wobei :

$$
eval \;\; x_1 \quad = \quad D[x_i]
$$

Beispiel:

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

	I
x_1	$\{x_3\}$
x_2	$\emptyset$
x_3	$\{x_1, x_2\}$

Beispiel:

$$\begin{aligned} x_1 &\supseteq \{a\} \cup x_3 \\ x_2 &\supseteq x_3 \cap \{a, b\} \\ x_3 &\supseteq x_1 \cup \{c\} \end{aligned}$$

	I
x_1	$\{x_3\}$
x_2	$\emptyset$
x_3	$\{x_1, x_2\}$

$D[x_1]$	$D[x_2]$	$D[x_3]$	W
$\emptyset$	$\emptyset$	$\emptyset$	$\boxed{x_1}, x_2, x_3$
$\{a\}$	$\emptyset$	$\emptyset$	$\boxed{x_2}, x_3$
$\{a\}$	$\emptyset$	$\emptyset$	$\boxed{x_3}$
$\{a\}$	$\emptyset$	$\{a, c\}$	$\boxed{x_1}, x_2$
$\{a, c\}$	$\emptyset$	$\{a, c\}$	$\boxed{x_3}, x_2$
$\{a, c\}$	$\emptyset$	$\{a, c\}$	$\boxed{x_2}$
$\{a, c\}$	$\{a\}$	$\{a, c\}$	$\emptyset$

Theorem

Sei $x_i \sqsupseteq f_i\,(x_1, \ldots, x_n)\,,\quad i = 1, \ldots, n$ ein Constraint-System über dem vollständigen Verband $\mathbb{D}$ der Höhe $h > 0$.

(1) Der Algorithmus terminiert nach maximal $h \cdot N$ Auswertungen rechter Seiten, wobei

$$N = \sum_{i=1}^{n} (1 + \#\,(\mathit{Dep}\; f_i)) \qquad // \quad \text{Größe des Systems} \quad \text{:-)}$$

(2) Der Algorithmus liefert eine Lösung.
Sind alle f_i monoton, liefert er die kleinste.

Beweis:

Zu (1):

Jede Variable x_i kann nur h mal ihren Wert ändern :-)

Dann wird die Menge $I[x_i]$ zu W hinzu gefügt.

Damit ist die Anzahl an Auswertungen:

$$\begin{aligned}
&\leq n + \sum_{i=1}^{n} (h \cdot \#\,(I[x_i])) \\
&= n + h \cdot \sum_{i=1}^{n} \#\,(I[x_i]) \\
&= n + h \cdot \sum_{i=1}^{n} \#\,(Dep\ f_i) \\
&\leq h \cdot \sum_{i=1}^{n} (1 + \#\,(Dep\ f_i)) \\
&= h \cdot N
\end{aligned}$$

Zu (2):

wir betrachten nur die Aussage für monotone f_i .

Sei D_0 die kleinste Lösung. Man zeigt:

- $D_0[x_i] \sqsupseteq D[x_i]$ (zu jedem Zeitpunkt)
- $D[x_i] \not\sqsupseteq f_i\,\textsf{eval} \implies x_i \in W$ (am Ende des Rumpfs)
- Bei Terminierung liefert der Algo eine Lösung :-))

Diskussion:

- Im Beispiel werden tatsächlich weniger Auswertungen rechter Seiten benötigt als bei RR-Iteration :-)
- Der Algo funktioniert auch für nicht-monotone f_i :-)
- Für monotone f_i kann man den Algo vereinfachen:

$$\boxed{D[x_i] = D[x_i] \sqcup t;} \implies \boxed{D[x_i] = \qquad t;}$$

- Für Widening ersetzt man:

$$\boxed{D[x_i] = D[x_i] \sqcup t;} \implies \boxed{D[x_i] = D[x_i] \sqcup\!\!\!\!\underline{\;\;} \; t;}$$

- Für Narrowing ersetzt man:

$$\boxed{D[x_i] = D[x_i] \sqcup t;} \implies \boxed{D[x_i] = D[x_i] \overline{\sqcap} \; t;}$$

Achtung:

- Der Algorithmus benötigt die Variablen-Abhängigkeiten *Dep* f_i .

  In unseren bisherigen Anwendungen waren die offensichtlich. Das muss nicht immer so sein :-(

- Wir benötigen eine Strategie für `extract` , die festlegt, welche Variable als nächstes auszuwerten ist.

- Am besten wäre es, wenn wir erst auswerten, dann auf das Ergebnis zugreifen ... :-)

$\Longrightarrow$ rekursive Auswertung ...

Idee:

→ Greifen wir in f_i auf ein x_j zu, werten wir erst rekursiv aus. Dann fügen wir x_i zu $I[x_j]$ hinzu :-)

$$
\begin{array}{lll}
\mathsf{eval}\ x_i\ x_j & = & \mathsf{solve}\ x_j; \\
 & & I[x_j] = I[x_j] \cup \{x_i\}; \\
 & & D[x_j];
\end{array}
$$

→ Damit die Rekursion nicht unendlich absteigt, verwalten wir die Menge *Stable* von Variablen, für die solve den Wert nachschlägt :-)

Anfangs ist $\mathit{Stable} = \emptyset$ ...

Die Funktion solve :

$$
\begin{array}{lll}
\text{solve } x_i & = & \text{if } (x_i \notin \mathit{Stable}) \; \{ \\
& & \quad \mathit{Stable} = \mathit{Stable} \cup \{x_i\}; \\
& & \quad t = f_i \, (\text{eval } \; x_i); \\
& & \quad \text{if } (t \not\sqsubseteq D[x_i]) \; \{ \\
& & \qquad W = I[x_i]; \quad I[x_i] = \emptyset; \\
& & \qquad D[x_i] = D[x_i] \sqcup t; \\
& & \qquad \mathit{Stable} = \mathit{Stable} \backslash W; \\
& & \qquad \text{app } \; \text{solve } \; W; \\
& & \quad \} \\
& & \}
\end{array}
$$

Helmut Seidl, TU München ;-)

Beispiel:

Betrachte unser Standard-Beispiel:

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

Dann sieht ein Trace des Fixpunkt-Algorithmus etwa so aus:

solve x_2 eval x_2 x_3 solve x_3 eval x_3 x_1 solve x_1 eval x_1 x_3 solve x_3

stable!

$I[x_3] = \{x_1\}$

$\Rightarrow \quad \emptyset$

$\boxed{D[x_1] = \{a\}}$

$I[x_1] = \{x_3\}$

$\Rightarrow \quad \{a\}$

$\boxed{D[x_3] = \{a, c\}}$

$I[x_3] = \emptyset$

solve x_1 eval x_1 x_3 solve x_3

stable!

$I[x_3] = \{x_1\}$

$\Rightarrow \quad \{a, c\}$

$\boxed{D[x_1] = \{a, c\}}$

$I[x_1] = \emptyset$

solve x_3 eval x_3 x_1 solve x_1

stable!

$I[x_1] = \{x_3\}$

$\Rightarrow \quad \{a, c\}$

$\boxed{\text{ok}}$

$I[x_3] = \{x_1, x_2\}$

$\Rightarrow \quad \{a, c\}$

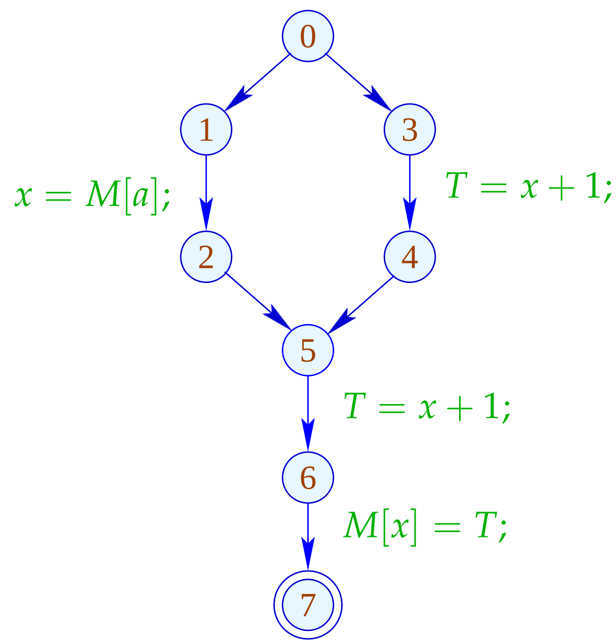
$\boxed{D[x_2] = \{a\}}$

- → Die Auswertung startet mit einer interessierenden Variable x_i (z.B. dem Wert für *stop*)
- → Es werden automatisch alle Variablen ausgewertet, die x_i beeinflussen :-)
- → Die Anzahl der Auswertungen ist i.a. kleiner als die bei normaler Iteration ;-)
- → Der Algorithmus ist komplizierter, benötigt aber keine Vorberechnung der Variablen-Abhängigkeiten :-))
- → Er funktioniert auch, wenn die Variablen-Abhängigkeiten sich während der Iteration ändern !!!

$\Longrightarrow$ interprozedurale Analyse

1.7 Beseitigung partieller Redundanzen

Beispiel:



// $\quad e = x + 1$ wird auf jedem Pfad ausgewertet ...

// leider auf einem Pfad sogar zweimal :-(

Ziel: