Verbesserung (Forts.):

→ Auch die Komposition lässt sich direkt implementieren:

$$\begin{array}{rcll}
(M_1 \circ M_2)\; x & = & b' \sqcup \bigsqcup_{y \in I'} y & \text{mit} \\
b' & = & b \sqcup \bigsqcup_{z \in I} b_z & \\
I' & = & \bigcup_{z \in I} I_z & \text{sofern} \\
M_1\; x & = & b \sqcup \bigsqcup_{y \in I} y & \\
M_2\; z & = & b_z \sqcup \bigsqcup_{y \in I_z} y &
\end{array}$$

→ Die Effekte von Zuweisungen sehen dann so aus:

$$[\![x = e;]\!]^\sharp = \begin{cases} \mathsf{Id}_{\mathit{Vars}} \oplus \{x \mapsto c\} & \text{falls} \quad e = c \in \mathbb{Z} \\ \mathsf{Id}_{\mathit{Vars}} \oplus \{x \mapsto y\} & \text{falls} \quad e = y \in \mathit{Vars} \\ \mathsf{Id}_{\mathit{Vars}} \oplus \{x \mapsto \top\} & \text{sonst} \end{cases}$$

... im Beispiel:

$$\begin{array}{lcl} [\![t = 0;]\!]^\sharp & = & \{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto \mathsf{ret}, \boxed{t \mapsto 0}\} \\ [\![b_1 = a_1;]\!]^\sharp & = & \{a_1 \mapsto a_1, \boxed{b_1 \mapsto a_1}, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\} \end{array}$$

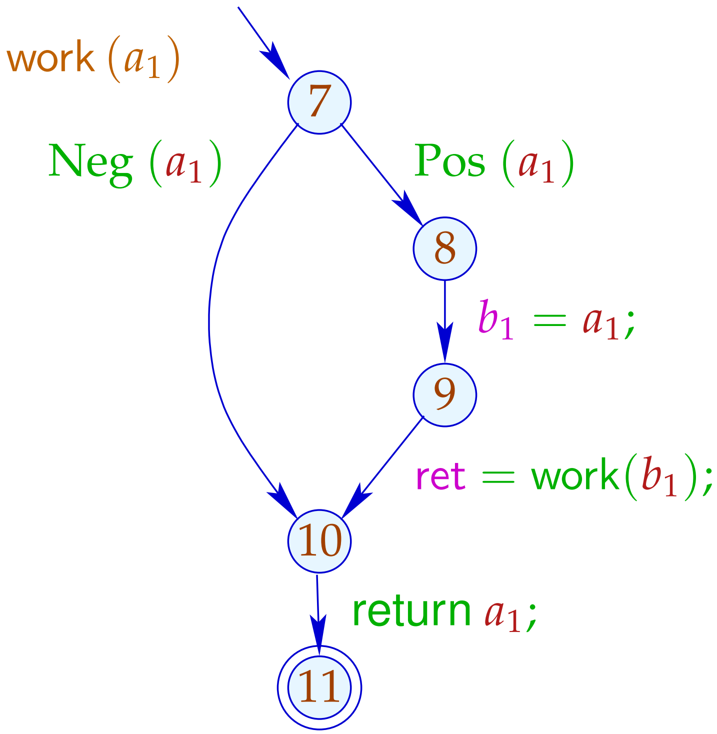
Um die Analyse zu implementieren, müssen wir nur noch den Effekt eines Aufrufs $k = (\_, \mathsf{ret} = f\,(b_1, \ldots, b_k);, \_)$ aus dem Effekt der Funktion f ermitteln:

$$\begin{array}{lcl} [\![k]\!]^\sharp & = & H_f\,([\![f]\!]^\sharp) \qquad\qquad \text{wobei:} \\ H_f\,(M) & = & \mathsf{Id} \oplus \{\mathsf{ret} \mapsto ((M \circ E_f)\ \mathsf{ret})\} \\ E_f\ x & = & \begin{cases} b_i & \text{falls} \quad x = a_i \\ 0 & \text{sonst} \end{cases} \end{array}$$

... im Beispiel:

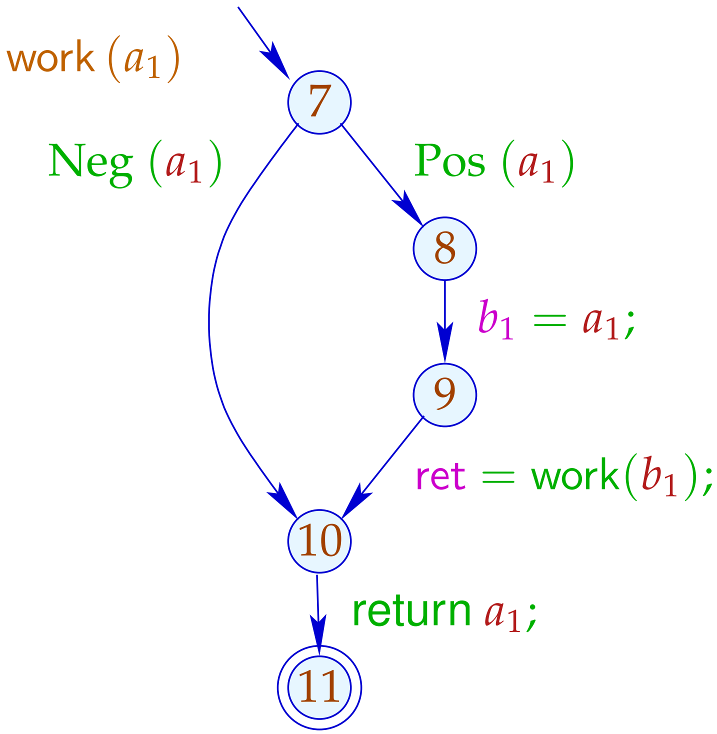
$$\begin{array}{llcl} \text{Falls} & [\![\mathsf{work}]\!]^\sharp & = & \{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto a_1\} \\ \text{dann} & H_{\mathsf{work}}\,[\![\mathsf{work}]\!]^\sharp & = & \mathsf{Id} \oplus \{\mathsf{ret} \mapsto (([\![\mathsf{work}]\!]^\sharp \circ E_f)\ \mathsf{ret})\} \\ & & = & \mathsf{Id} \oplus \{\mathsf{ret} \mapsto (\boxed{\{a_1 \mapsto b_1, b_1 \mapsto 0, \mathsf{ret} \mapsto b_1\}}\ \mathsf{ret})\} \\ & & = & \{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto b_1\} \end{array}$$

Damit können wir die Fixpunkt-Iteration durchführen :-)



	1
7	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto \mathsf{ret}\}$
10	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto \mathsf{ret}\}$
11	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto a_1\}$
8	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto \mathsf{ret}\}$
9	$\{a_1 \mapsto a_1, b_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}\}$

$$\begin{aligned}
[\![(9,\ldots,10)]\!]^\sharp \circ [\![9]\!]^\sharp &= \{a_1 \mapsto a_1, b_1 \mapsto b_1, \mathsf{ret} \mapsto b_1\} \circ \\
&\ \{a_1 \mapsto a_1, b_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}\} \\
&= \{a_1 \mapsto a_1, b_1 \mapsto a_1, \mathsf{ret} \mapsto a_1\}
\end{aligned}$$



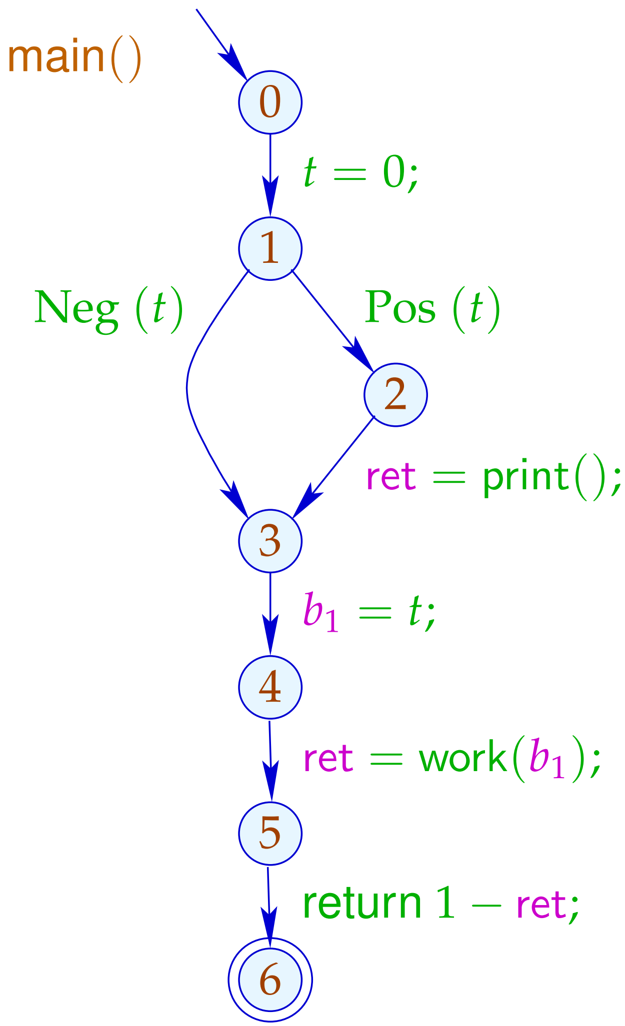
	2
7	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \text{ret} \mapsto \text{ret}\}$
10	$\boxed{\{a_1 \mapsto a_1, b_1 \mapsto a_1 \sqcup b_1, \text{ret} \mapsto a_1 \sqcup \text{ret}\}}$
11	$\boxed{\{a_1 \mapsto a_1, b_1 \mapsto a_1 \sqcup b_1, \text{ret} \mapsto a_1\}}$
8	$\{a_1 \mapsto a_1, b_1 \mapsto b_1, \text{ret} \mapsto \text{ret}\}$
9	$\{a_1 \mapsto a_1, b_1 \mapsto a_1, \text{ret} \mapsto \text{ret}\}$

$$
\begin{aligned}
[\![(9, \ldots, 10)]\!]^\sharp \circ [\![9]\!]^\sharp &= \{a_1 \mapsto a_1, b_1 \mapsto b_1, \text{ret} \mapsto b_1\} \circ \\
& \{a_1 \mapsto a_1, b_1 \mapsto a_1, \text{ret} \mapsto \text{ret}\} \\
&= \{a_1 \mapsto a_1, b_1 \mapsto a_1, \text{ret} \mapsto a_1\}
\end{aligned}
$$

Wenn wir die Effekte von Funktionsaufrufen kennen, können wir ein Constraintsystem aufstellen, um den abstrakten Zustand bei Erreichen eines Punkts ermitteln:

$$
\begin{array}{lllll}
\mathcal{R}[\mathsf{main}] & \sqsupseteq & \mathsf{enter}_0\; d_0 & & \\
\mathcal{R}[f] & \sqsupseteq & \mathsf{enter}_f^\sharp\; (\mathcal{R}[u]) & k = (u, \mathsf{ret} = f\,(a_1, \ldots, a_k);, \_) & \text{Aufruf} \\
\mathcal{R}[v] & \sqsupseteq & \mathcal{R}[f] & v \quad \text{Anfangspunkt von} \quad f & \\
\mathcal{R}[v] & \sqsupseteq & [\![k]\!]^\sharp\; (\mathcal{R}[u]) & k = (u, \_, v) \quad \text{Kante} &
\end{array}
$$

... im Beispiel:



0	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto 0, t \mapsto 0\}$
1	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto 0, t \mapsto 0\}$
2	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto 0, t \mapsto 0\}$
3	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
4	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
5	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto 0, t \mapsto 0\}$
6	$\{b_1 \mapsto 0, \mathsf{ret} \mapsto \top, t \mapsto 0\}$

Diskussion:

- Zumindest Kopier-Konstanten lassen sich interprozedural ermitteln.
- Dazu mussten wir Bedingungen und kompliziertere Zuweisungen ignorieren :-(
- In der zweiten Phase hätten wir allerdings exakter rechnen können :-)
- Die weitere Abstrahierung war aus zwei Gründen notwendig:

  (1) Die Menge der auftretenden Transformer $\mathbb{M} \subseteq \mathbb{D} \to \mathbb{D}$ muss endlich sein;

  (2) Die Funktionen $M \in \mathbb{M}$ müssen effizient implementierbar sein :-)

- Auf die zweite Bedingung kann evt. verzichtet werden ...

Beobachtung: Sharir/Pnueli, Cousot

→ Oft werden Funktionen nur mit wenigen verschiedenen abstrakten Argumenten aufgerufen.

→ Man könnte dann doch jede Funktion für nur genau diese Aufrufe analysieren :-)

→ Stelle das folgende Constraint-System auf:

$$
\begin{array}{llll}
[\![v, a]\!]^\sharp & \sqsupseteq & a & v \quad \text{Eintrittspunkt} \\
[\![v, a]\!]^\sharp & \sqsupseteq & \mathsf{combine}^\sharp\,([\![u, a]\!], [\![f, \mathsf{enter}_f^\sharp\, [\![u, a]\!]^\sharp]\!]^\sharp) & \\
 & & & (u, \mathsf{ret} = f\,(a_1, \ldots, a_k);, v) \quad \text{Aufruf} \\
[\![v, a]\!]^\sharp & \sqsupseteq & [\![lab]\!]^\sharp\, [\![u, a]\!]^\sharp & k = (u, lab, v) \quad \text{Kante} \\
[\![f, a]\!]^\sharp & \sqsupseteq & [\![stop_f, a]\!]^\sharp & stop_f \quad \text{Endpunkt von} \quad f
\end{array}
$$

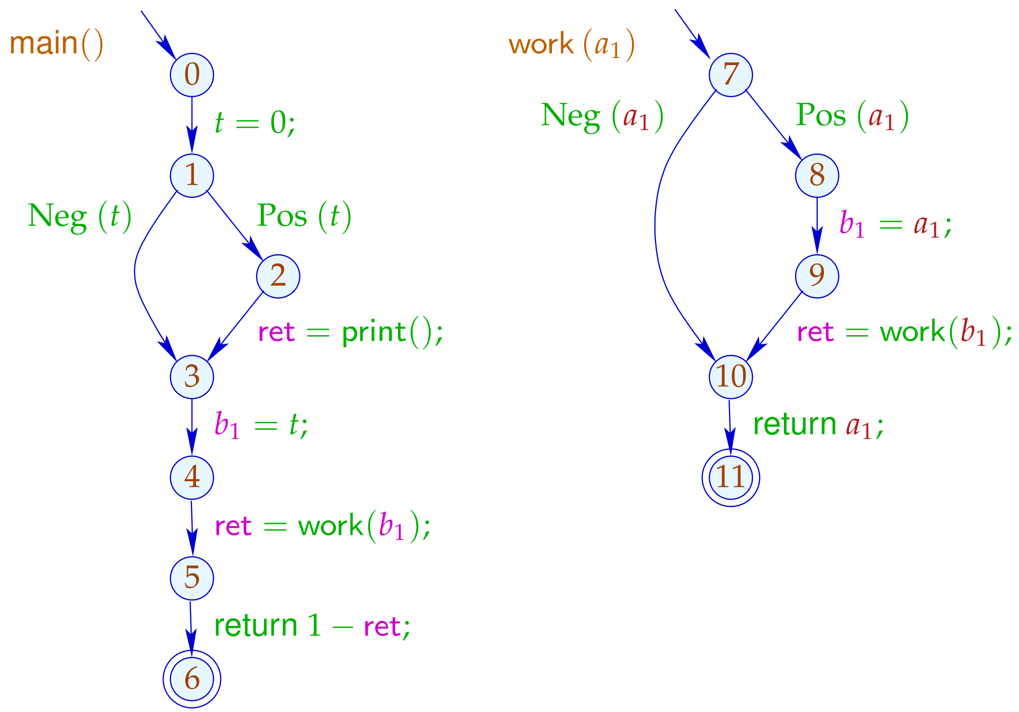
// $[\![v, a]\!]^\sharp$ $==$ Wert des Effekts für das Argument a .

Diskussion:

- Dieses Constraint-System ist i.a. riesengroß :-(
- Wir wollen es aber gar nicht komplett lösen !!!
- Uns reicht es, die korrekten Werte für alle Aufrufe zu ermitteln, die vorkommen, d.h. für die Berechnung des Werts $[\![\mathsf{main}(), a_0]\!]^\sharp$ benötigt werden $\Longrightarrow$ wir verwenden unseren lokalen Fixpunkt-Algorithmus :-))
- Der Fixpunkt-Algo liefert uns sogar noch die Menge der aktuellen Parameter $a \in \mathbb{D}$, für die eine Funktion (möglicherweise) aufgerufen wird sowie die Werte an allen ihren Programm-Punkten für jeden dieser Aufrufe :-)

... im Beispiel:

Versuchen wir einfach einmal eine volle Konstanten-Propagation ...



0	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
1	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
2	$\lambda\, x.\, 0$	$\bot$
3	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
4	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
7	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
8	$\lambda\, x.\, 0$	$\bot$
9	$\lambda\, x.\, 0$	$\bot$
10	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
11	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
5	$\lambda\, x.\, 0$	$\lambda\, x.\, 0$
6	$\lambda\, x.\, 0$	$(\lambda\, x.\, 0) \oplus \{\mathsf{ret} \mapsto 1\}$
main()	$\lambda\, x.\, 0$	$(\lambda\, x.\, 0) \oplus \{\mathsf{ret} \mapsto 1\}$

Diskussion:

- Im Beispiel terminiert die Analyse schnell :-)
- Falls $\mathbb{D}$ endliche Höhe hat, terminiert die Analyse, sofern nur jede Funktion während der Iteration nur mit endlich vielen verschiedenen Argumenten aufgerufen wird :-))
- Analoge Analyse-Algorithmen erwiesen sich bei der Analyse von Prolog als äußerst effizient und präzise :-)
- Zusammen mit einer Points-To-Analyse und Propagation selbst von negativer Konstanten-Information haben wir diesen Algorithmus äußerst erfolgreich zur Fehlersuche in C mit Posix-Threads eingesetzt :-)

(2) Der Call-String-Ansatz:

Idee:

$\rightarrow$ Berechne die Menge aller erreichbaren Aufrufkeller!

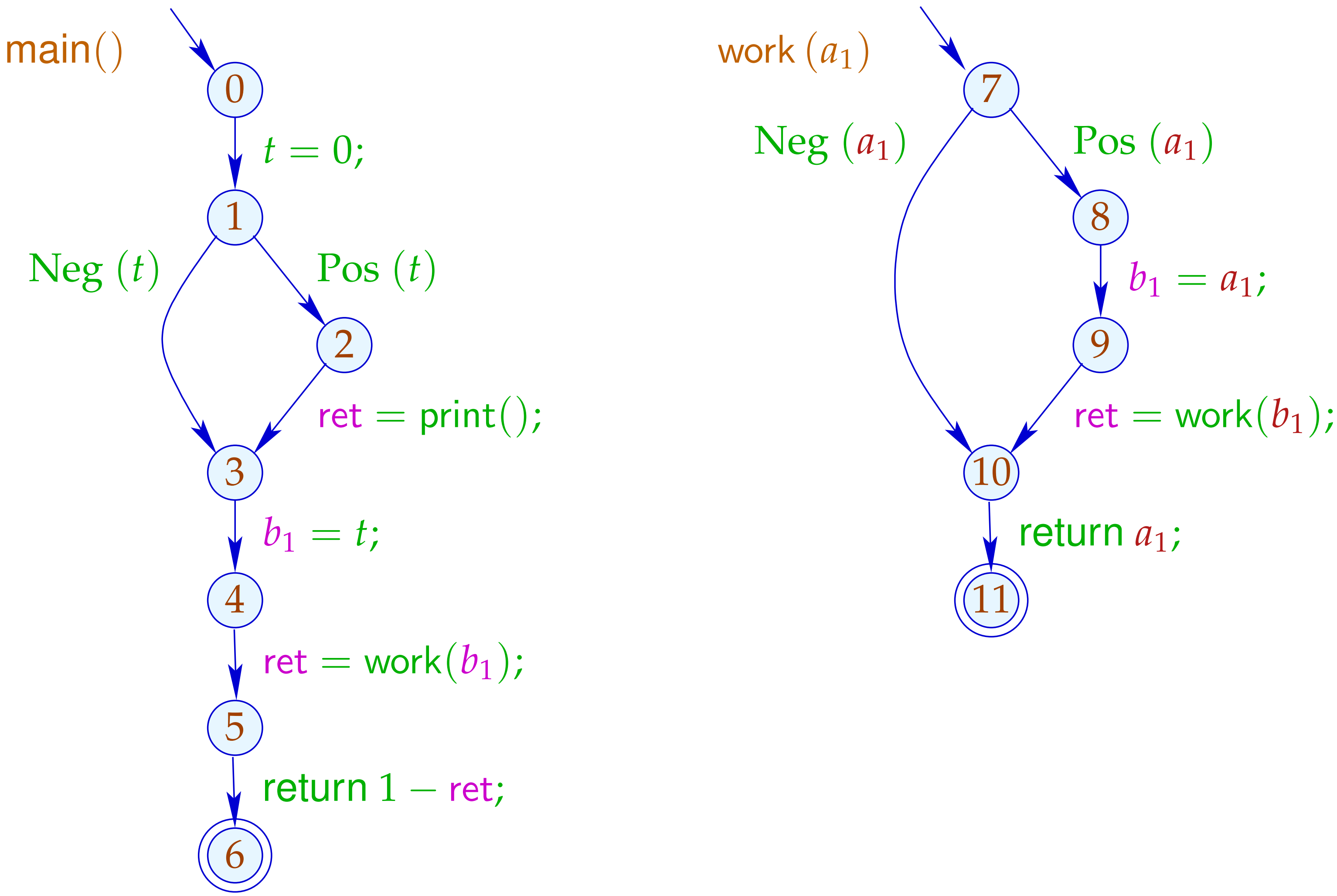
$\rightarrow$ Diese ist i.a. unendlich :-(

$\rightarrow$ Behandle Keller bis zu einer festen Tiefe d exakt! Behalte von längeren Kellern nur das obere Ende der Länge d :-)

$\rightarrow$ Wichtiger Spezialfall: $d = 0$.

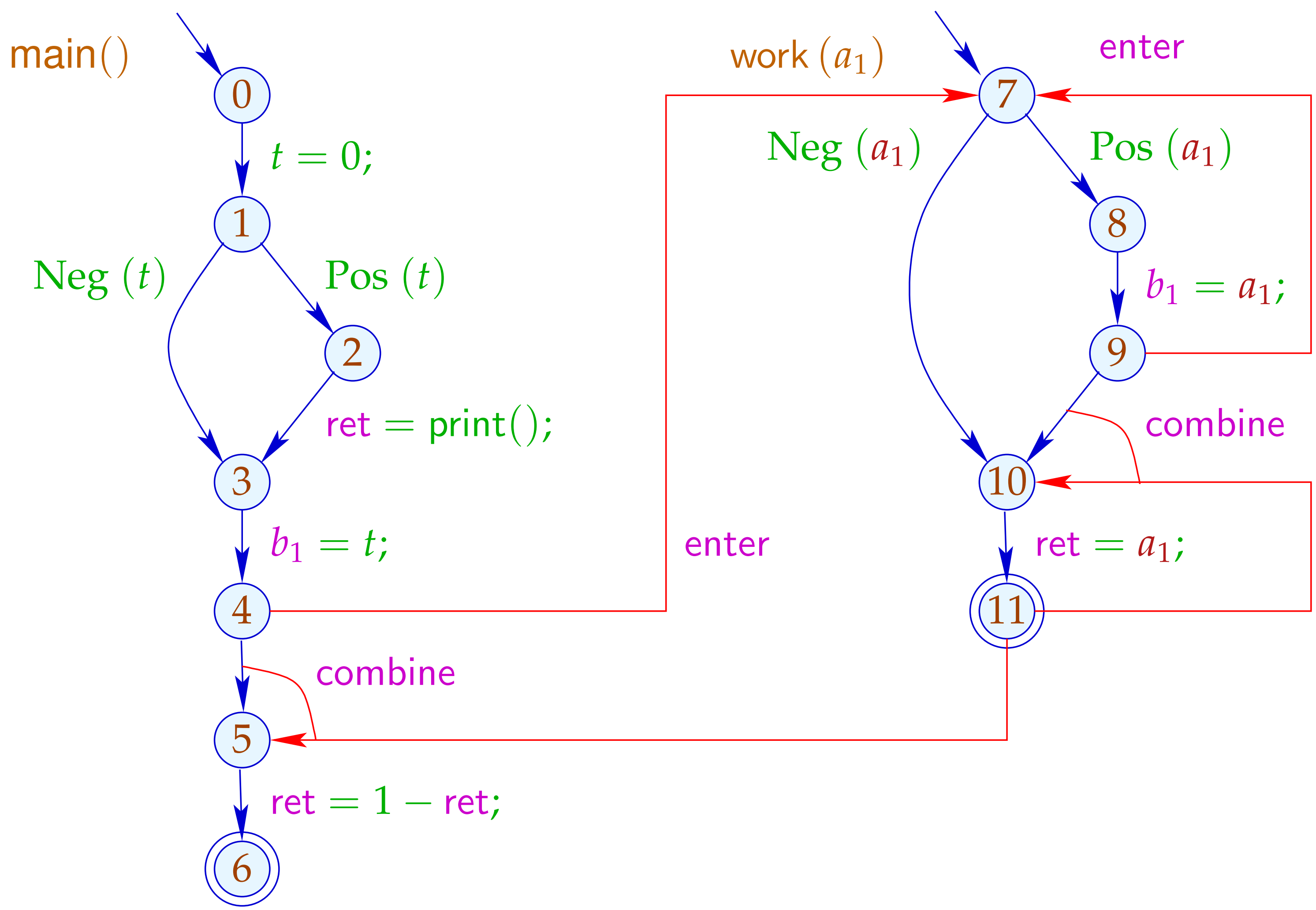
$\Longrightarrow$ Betrachte nur die obersten Kellerrahmen ...

... im Beispiel:



main()
0
$t = 0;$
1
Neg (t)
Pos (t)
2
ret = print();
3
$b_1 = t;$
4
ret = work(b_1);
5
return 1 − ret;
6
work (a_1)
7
Neg (a_1)
Pos (a_1)
8
$b_1 = a_1;$
9
ret = work(b_1);
10
return a_1;
11

... im Beispiel:


main()
0
$t = 0;$
1
Neg (t)
Pos (t)
2
ret = print();
3
$b_1 = t;$
4
combine
5
ret = 1 − ret;
6
enter
work (a_1)
7
enter
Neg (a_1)
Pos (a_1)
8
$b_1 = a_1;$
9
combine
10
ret = a_1;
11

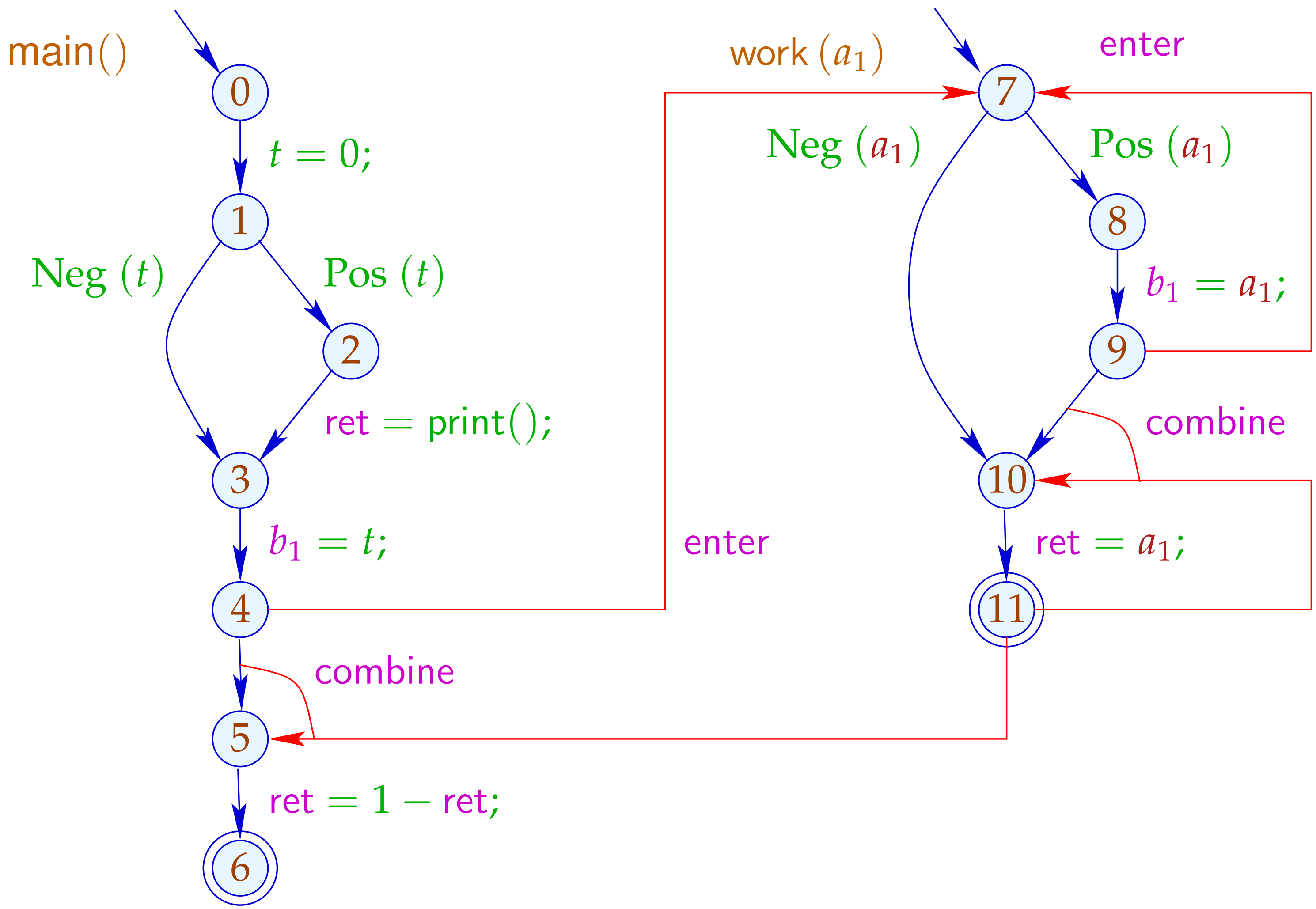
Die Bedingungen für $5, 7, 10$ sind dann etwa:

$$
\begin{array}{lll}
\mathcal{R}[5] & \sqsupseteq & \mathsf{combine}^\sharp\,(\mathcal{R}[4], \mathcal{R}[11]) \\
\\
\mathcal{R}[7] & \sqsupseteq & \mathsf{enter}_f^\sharp\,(\mathcal{R}[4]) \\
\mathcal{R}[7] & \sqsupseteq & \mathsf{enter}_f^\sharp\,(\mathcal{R}[9]) \\
\\
\mathcal{R}[10] & \sqsupseteq & \mathsf{combine}^\sharp\,(\mathcal{R}[9], \mathcal{R}[11])
\end{array}
$$

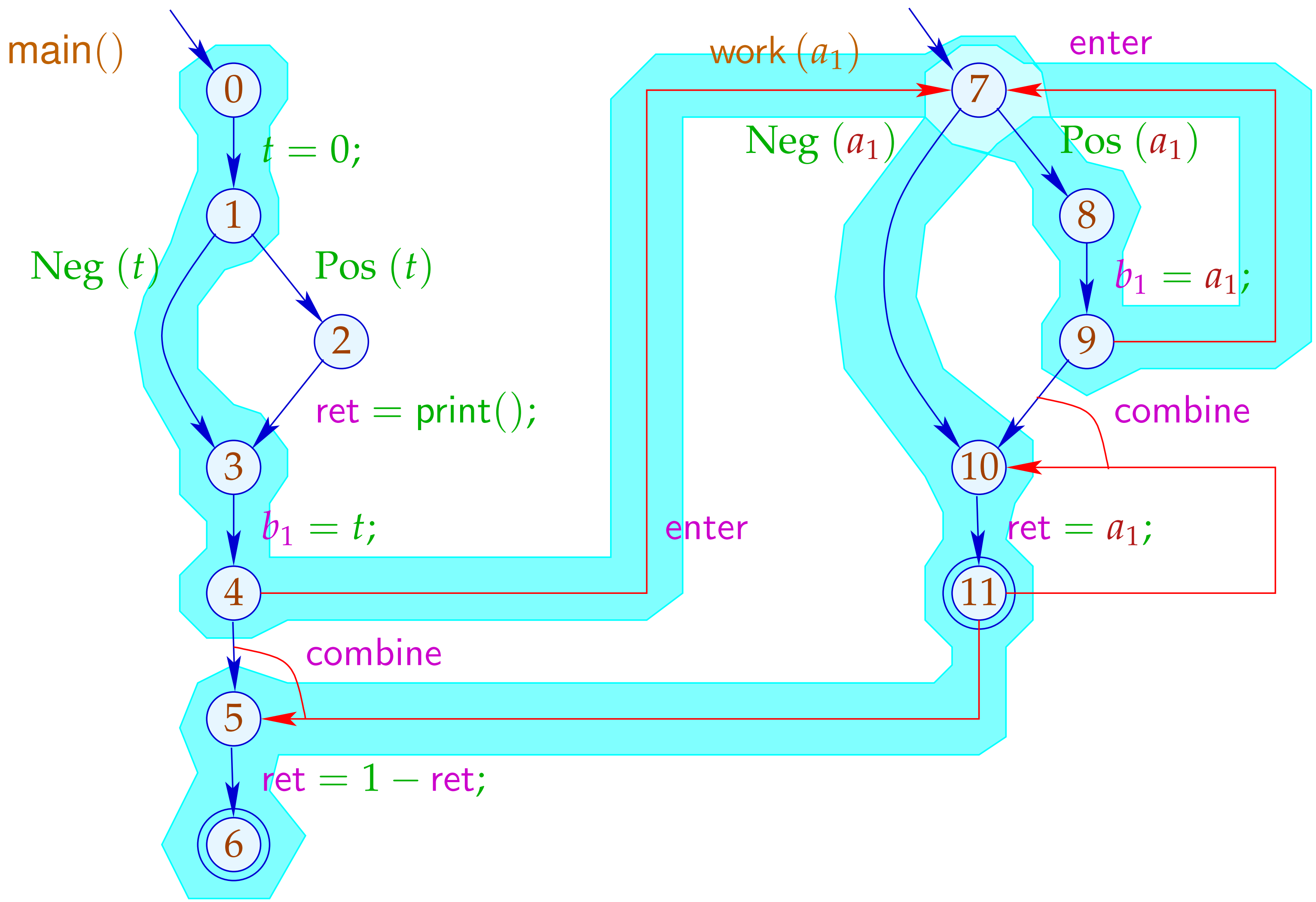
Achtung:

Der resultierende Supergraph enthält offensichtlich unmögliche Pfade ...

... im Beispiel ist das etwa:



... im Beispiel ist das etwa:



Beachte:

- → Im Beispiel finden wir zwar die gleichen Ergebnisse:
  Mehr Pfade machen die Ergebnissse evt. weniger präzise.

  Insbesondere analysieren wir jede Funktion nur für ein (evt. sehr nichtssagendes) Argument-Tupel :-(
- → Die Analyse terminiert — sofern nur $\mathbb{D}$ keine unendlichen echt aufsteigenden Ketten besitzt :-)
- → Die Korrektheit zeigt man relativ zur operationellen Semantik mit den Stacks.
- → Für die Korrektheit des funktionalen Ansatzes ist die Semantik über Berechnungswälder besser geeignet :-)