Zurück zu CM im allgemeinen

Traditionell

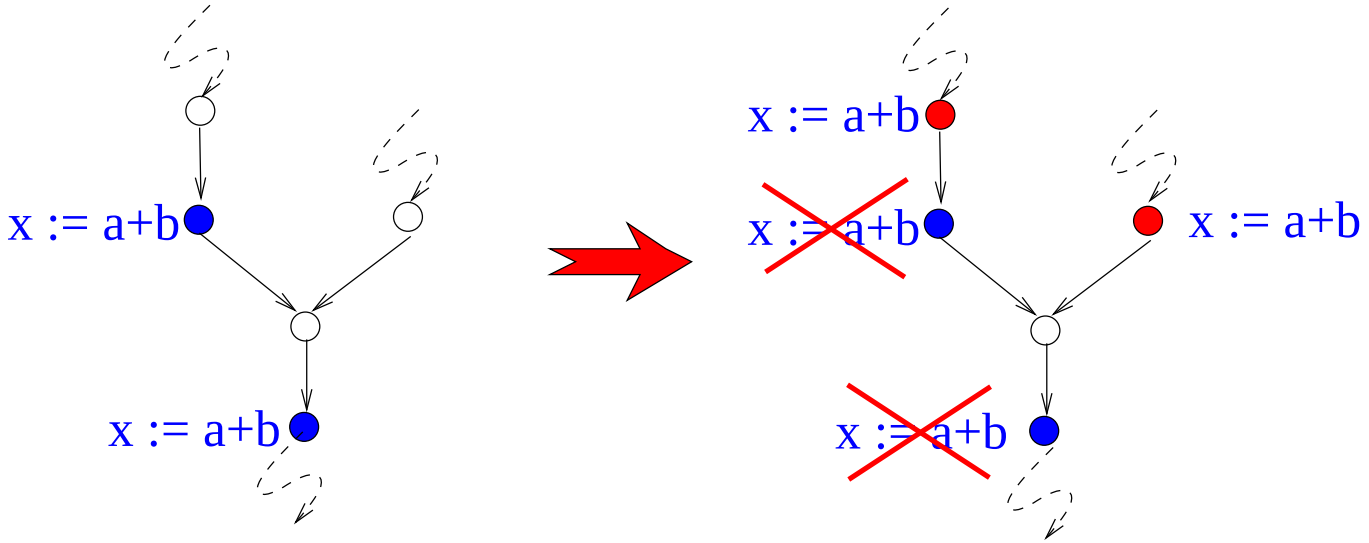
- Code (C) heißt Ausdrücke
- Motion (M) heißt vorziehen

Aber...

- CM ist mehr als vorziehen von Ausdrücken und PR(E)E!

Zum Beispiel: Auch...

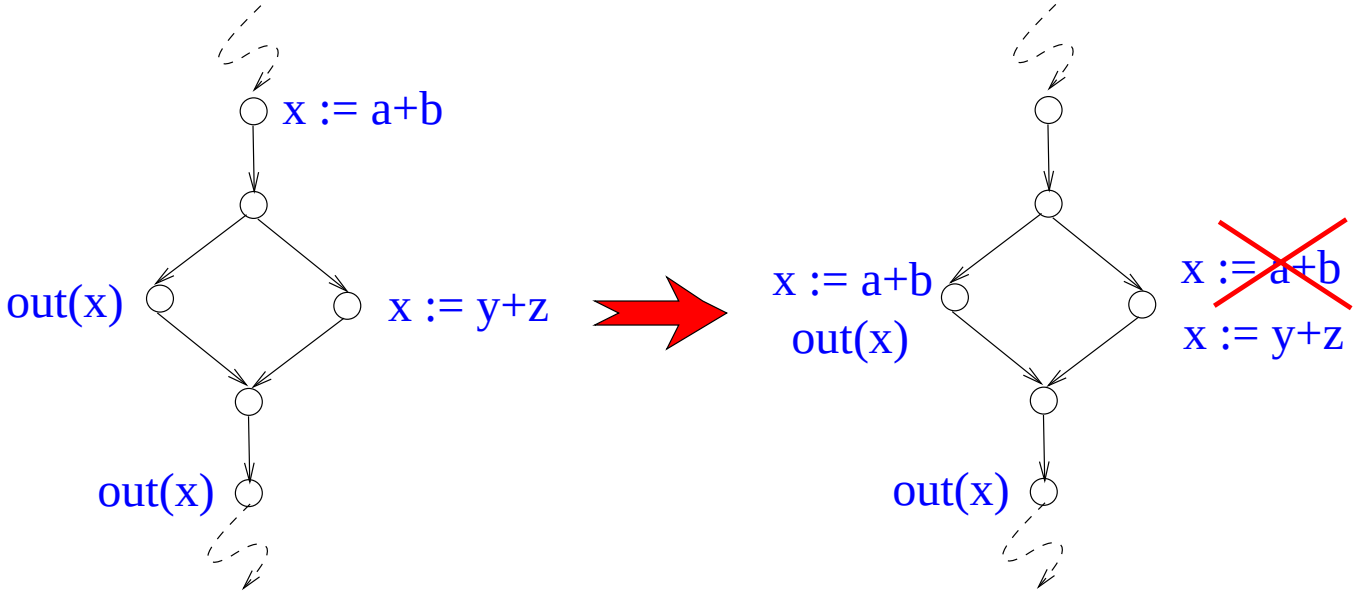
...Anweisungen sind Code.



- Hier heißt CM Elimination partiell redundanter Anweisungen (PRAE)

Im Unterschied zu Ausdrücken, können...

...Anweisungen auch verzögert werden.



- CM heißt jetzt Elimination partiell toten Codes (PDCE)

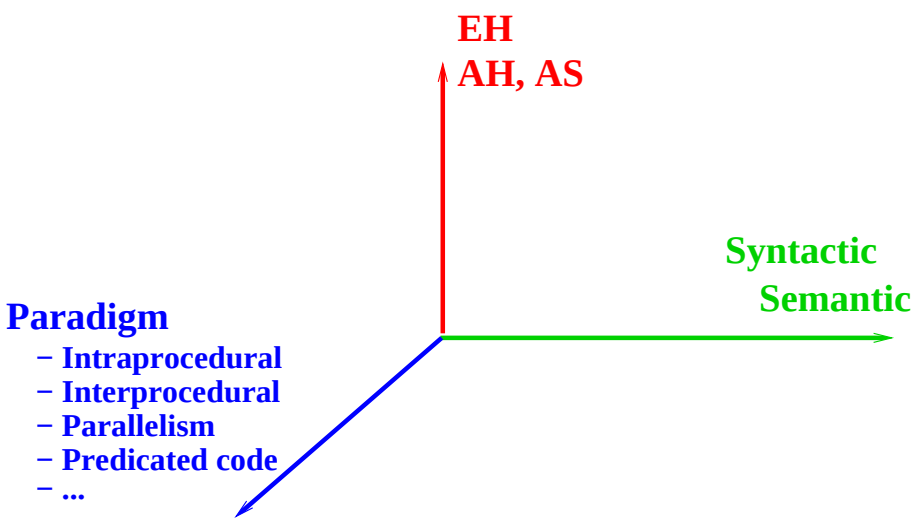
Über den Entwurfsraum von CM-Algorithmen...

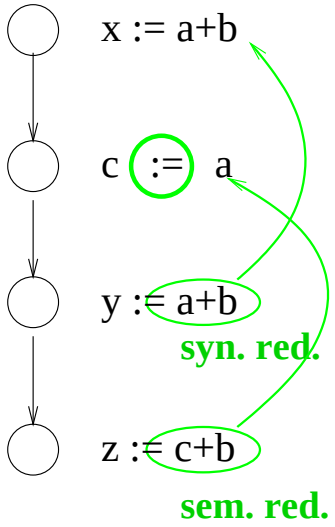
Allgemeiner...

- Code heißt Ausdrücke/Anweisungen
- Motion heißt vorziehen/verzögern

Code / Motion	Hoisting	Sinking
Expressions	EH	·/·
Assignments	AH	AS

Weitere Verfeinerung des Entwurfsraums von CM-Algorithmen...

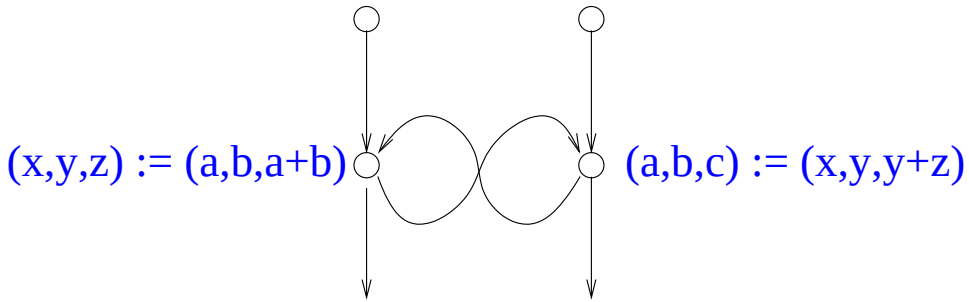


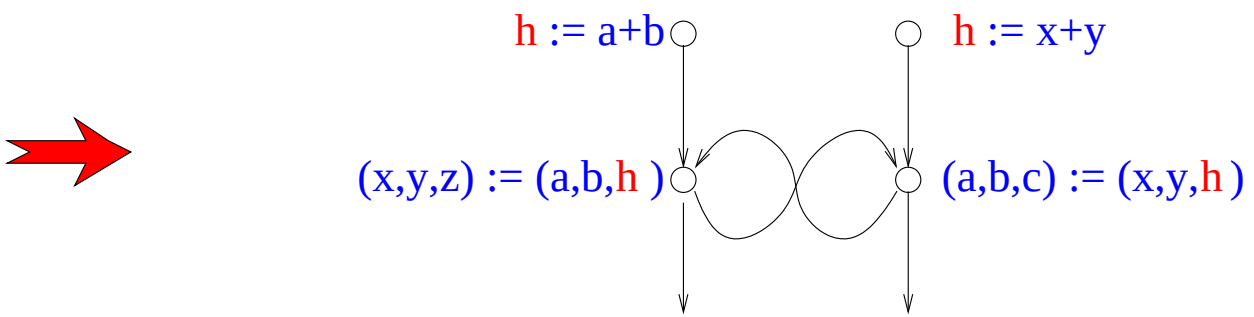


Introducing semantics... !

Semantisches Code Motion...

erlaubt mächtigere Optimierungen!





(Beispiel von B. Steffen, TAPSOFT'87)

Optimalitätsergebnisse sind sehr empfindlich!

Drei Beispiele sollen dies belegen...

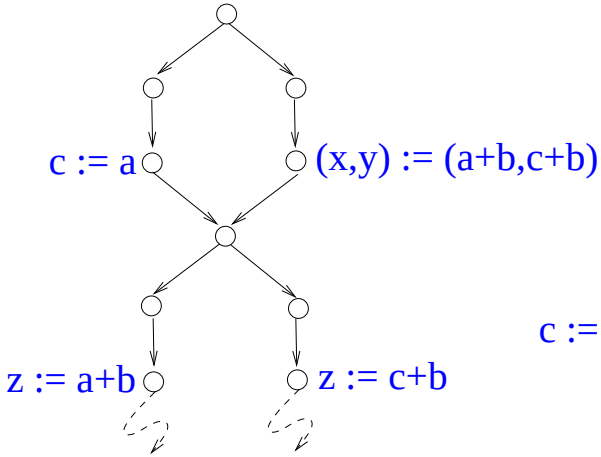
(I) Code motion vs. code placement

(II) Abhängigkeiten elementarer Transformationen

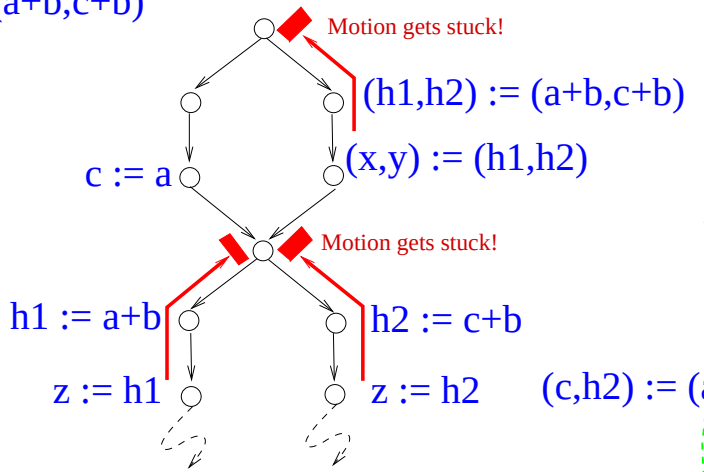
(III) Paradigmenabhängigkeiten

(I) Code Motion vs. Code Placement

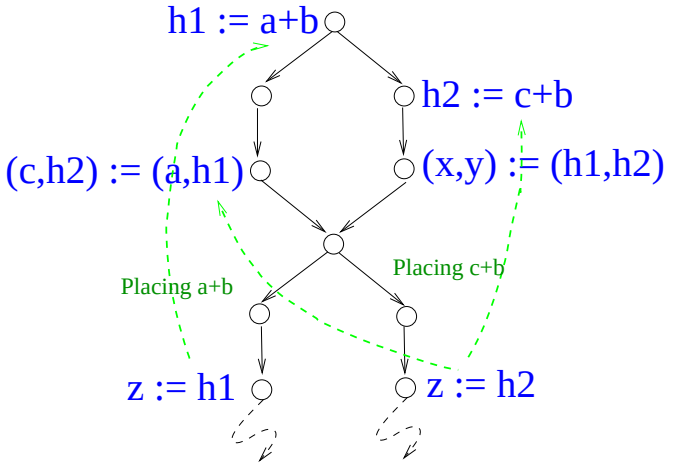
...sind keine Synonyme!



Original Program



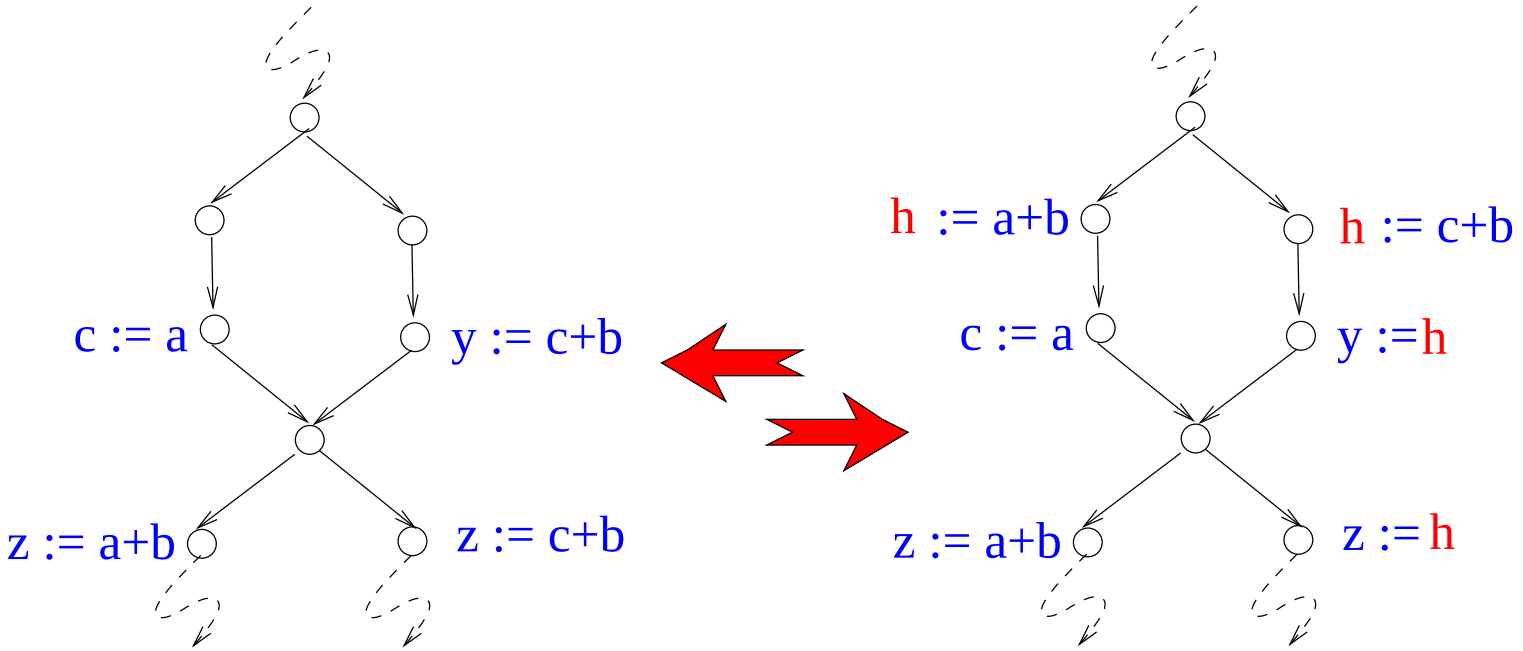
After Sem. Code Motion



After Sem. Code Placement

Schlechter noch...

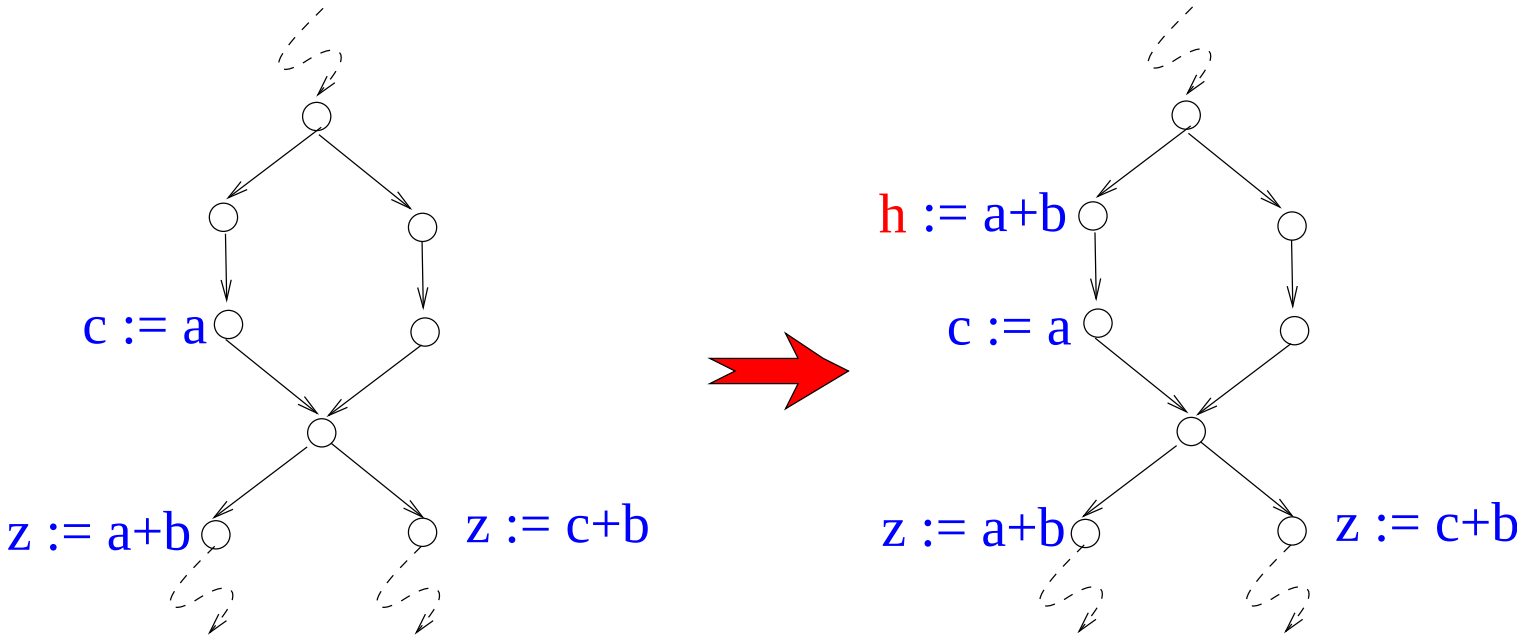
Optimalität ist verloren!



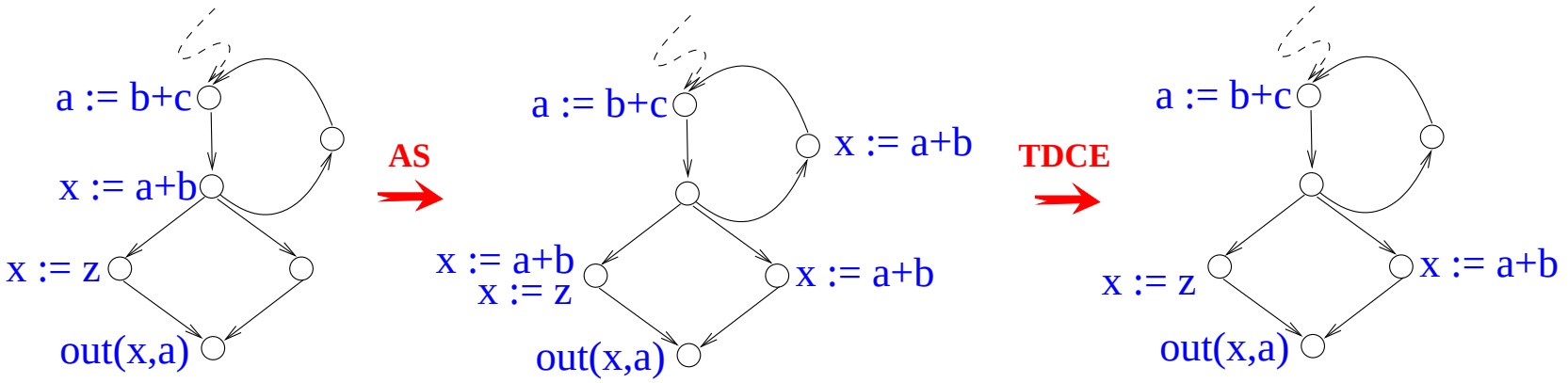
Incomparable!

Und sogar noch schlechter...

Performanz kann verloren gehen, wenn naiv angewandt!

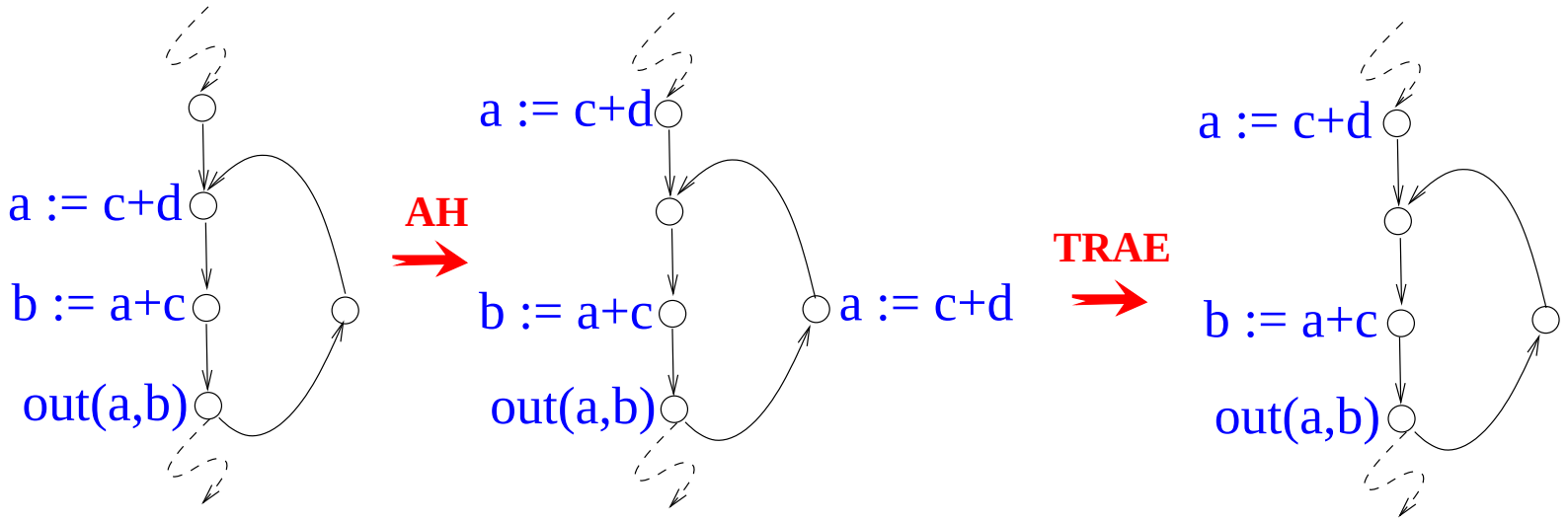


(II) Abhängigkeiten von Transformationen



$\rightsquigarrow$ ...Partial Dead-Code Elimination (PDCE)

Abhängigkeiten von Transformationen



...2nd Order Effects!

$\rightsquigarrow$ ...Partially Redundant Assignment Elimination (PRAE)

Konzeptuell

...können wir PREE, PRAE und PDCE wie folgt verstehen:

- PREE = EH ; TREE
- PRAE = (AH + TRAE)*
- PDCE = (AS + TDCE)*

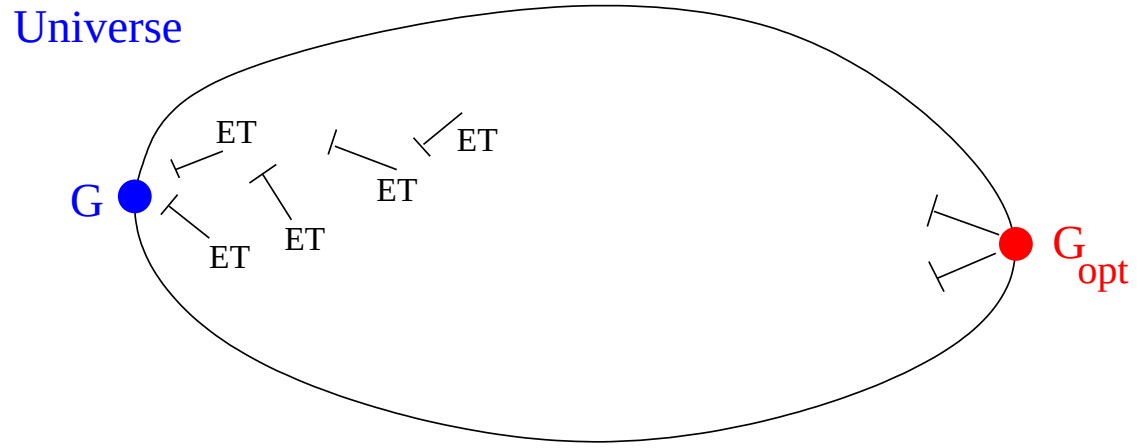
PRAE/PDCE – Optimalitätsergebnisse

Ableitungsrelation $\vdash$...

- PRAE... $G \vdash_{AH,TRAE} G'$ (ET={AH,TRAE})
- PDCE... $G \vdash_{AS,TDCE} G'$ (ET={AS,TDCE})

Wir können beweisen...

Optimalitätstheorem
Für PRAE und PDCE ist $\vdash_{ET}$ konfluent und terminierend



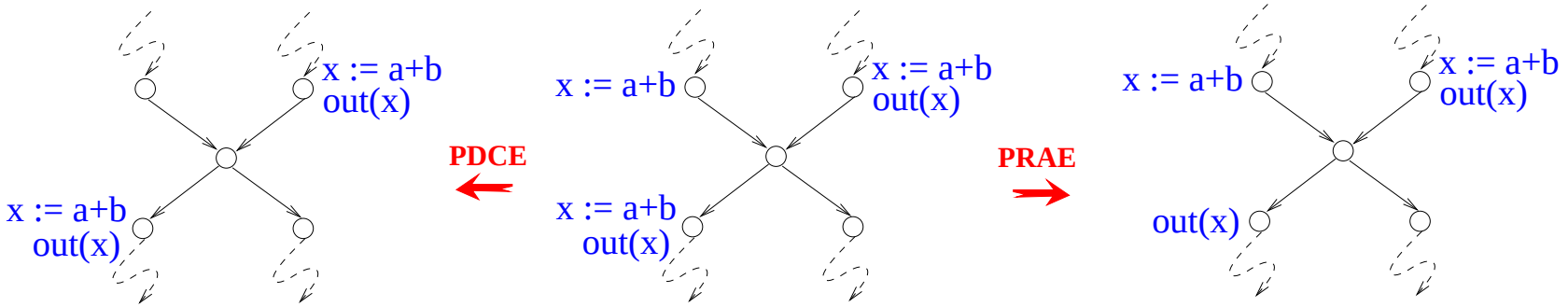
Betrachte jetzt...

- Assignment Placement AP

  $AP = (AH + TRAE + AS + TDCE)^*$

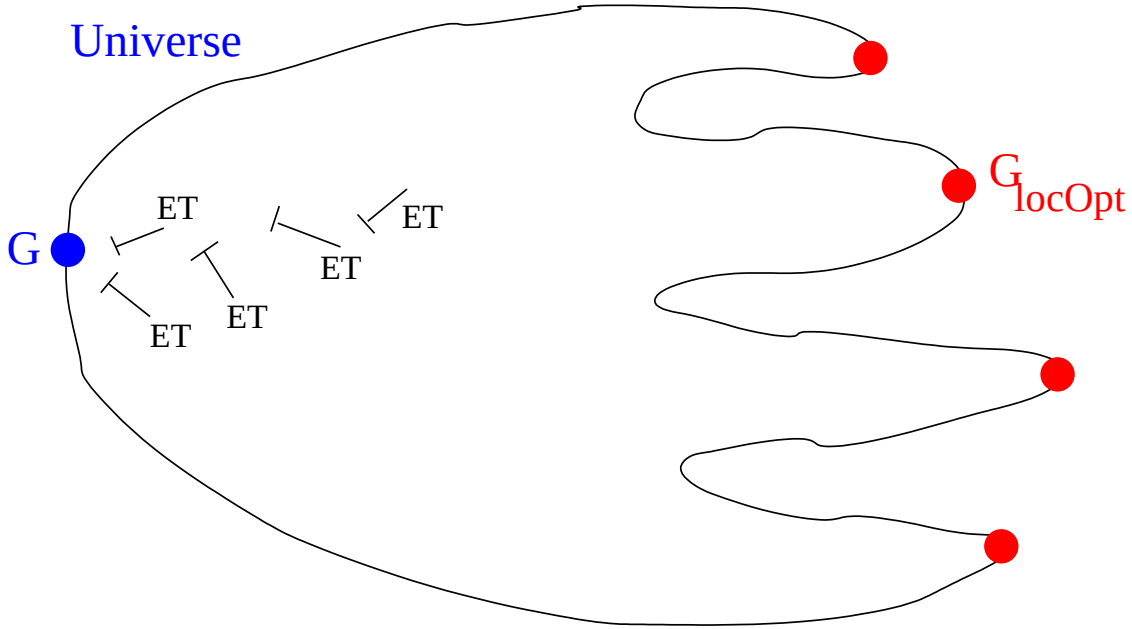
...sollte noch mächtiger sein!

In der Tat, aber...



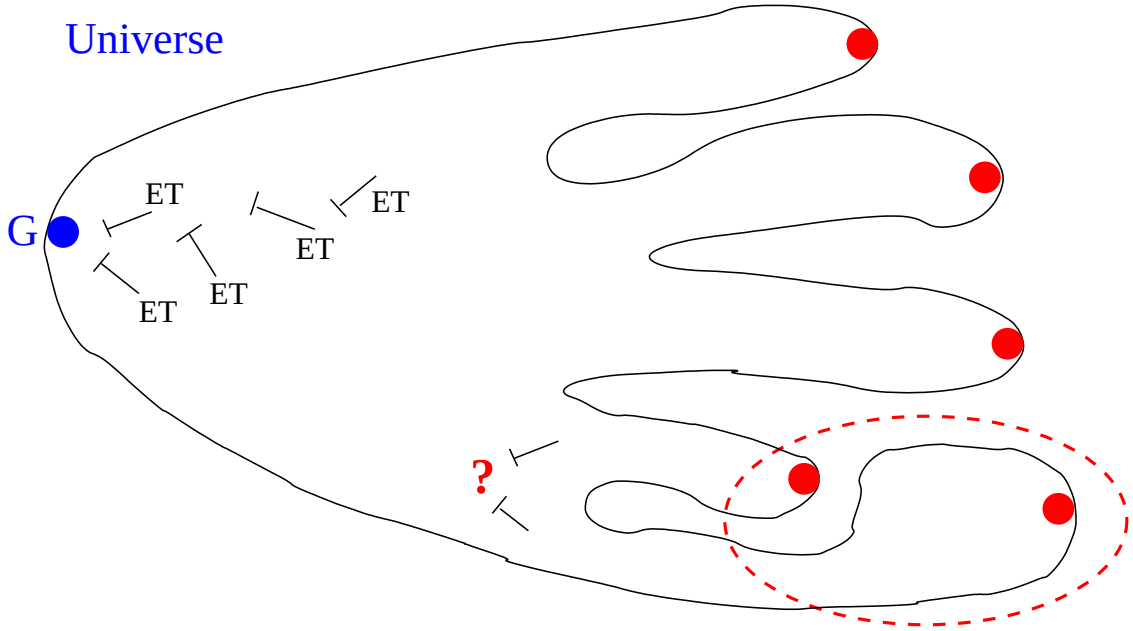
Konfluenz...

...und folglich (globale) Optimalität ist verloren!

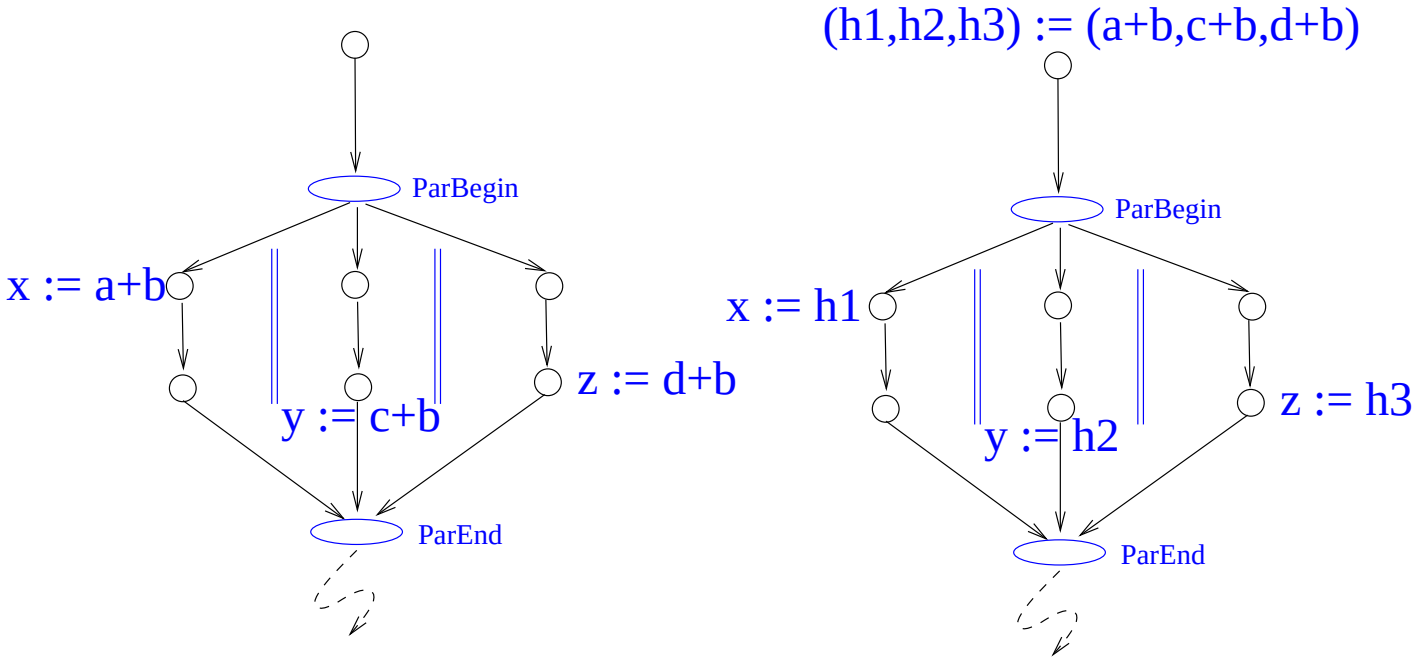


Noch schlechter...

...es gibt Szenarien, in denen wir mit Universen wie dem folgenden enden können:



(III) Paradigmenabhängigkeiten



Original Program **After Earliestness Transformation**

...ein naiver Transfer der Transformationsstrategie führt hier zu einem im wesentlichen sequentiellen Programm!

Interprozedurale Datenflussanalyse

In zwei Teilen...

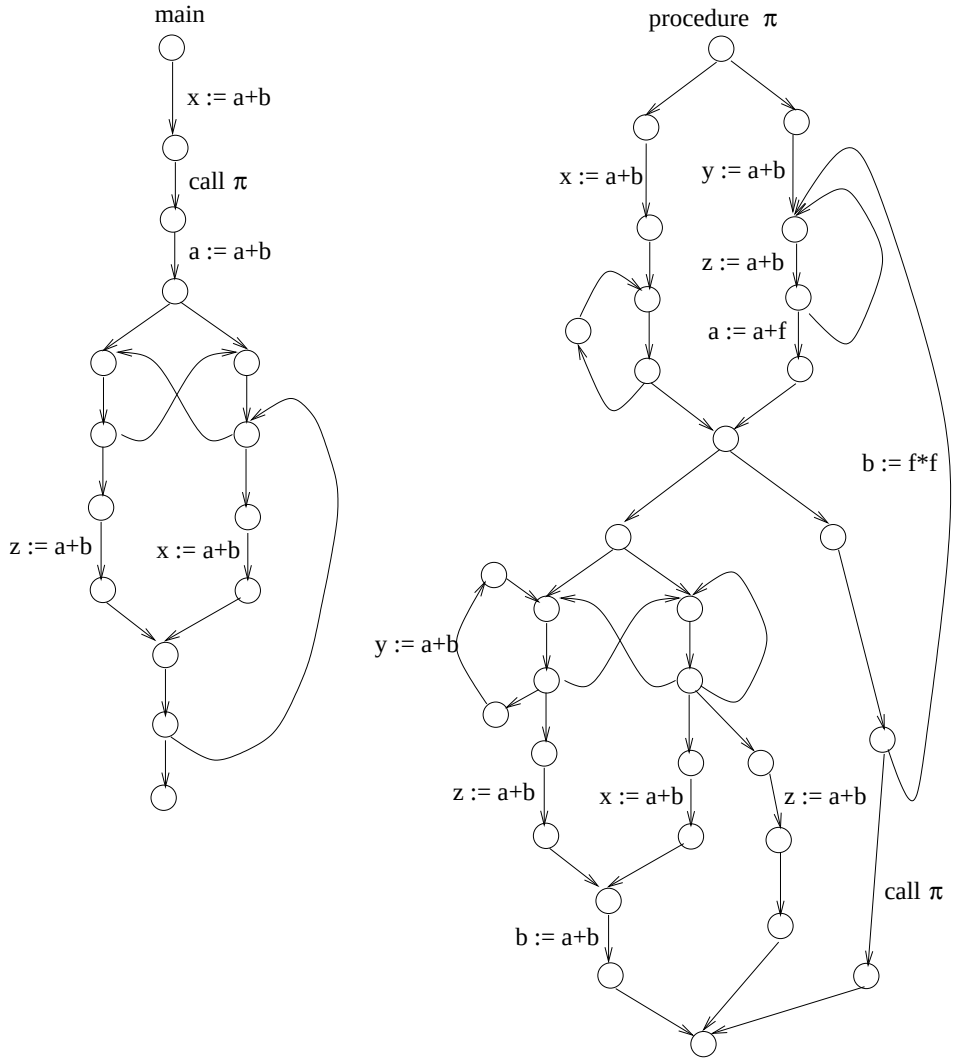
- Der Basisfall: Parameterlose Prozeduren, keine lokalen Variablen
- Erweiterungen
  - Wertparameter und lokale Variablen
  - Referenzparameter
  - Prozedurparameter

Interprozedurale Datenflussanalyse

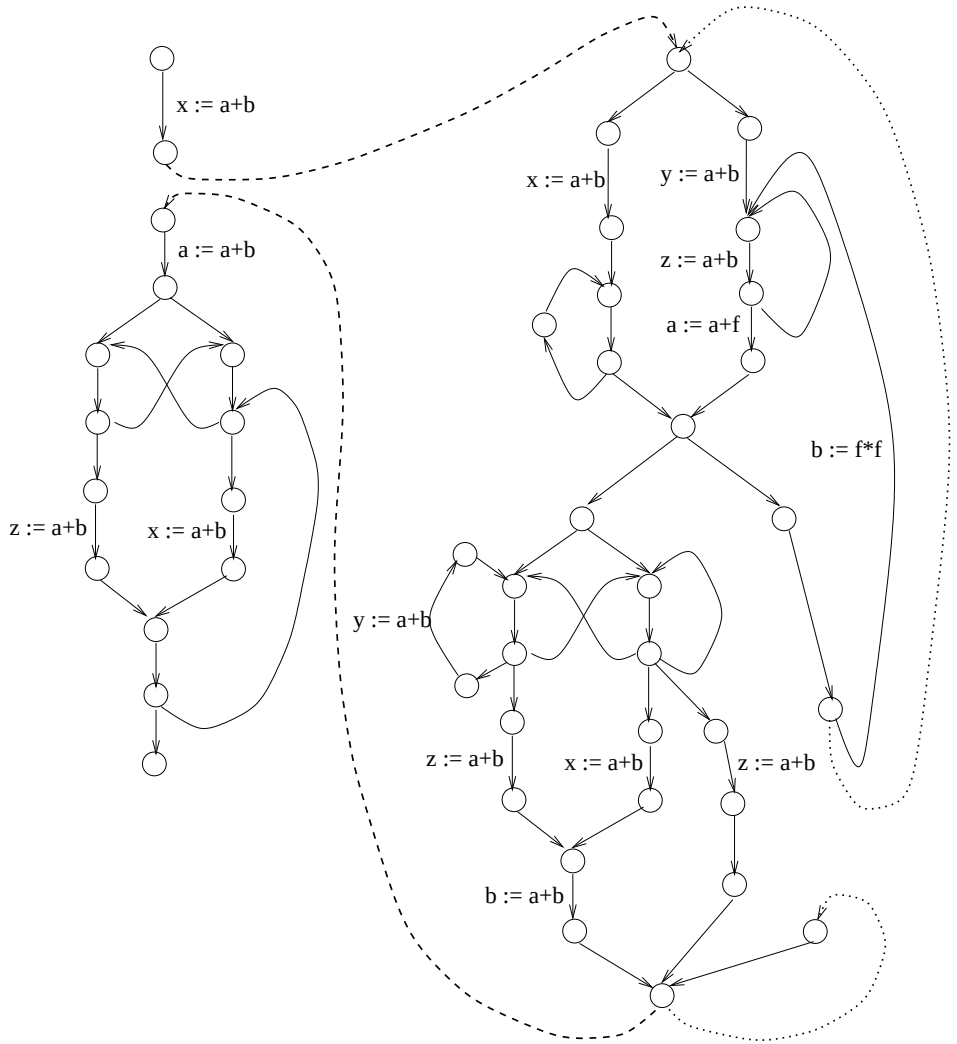
Darstellung von Programmen durch

- Flussgraphsysteme
- Interprozedurale Flussgraphen

Interprozedurale Flussgraphsysteme



Interprozedurale Flussgraphen



(Lokale) abstrakte Semantik

Zwei Komponenten:

- *Datenflussanalyseverband* $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$
- *Datenflussanalysefunktional* $[\![\ ]\!]' : E^* \to (\mathcal{C} \to \mathcal{C})$

Der *IMOP*-Ansatz

Die *IMOP*-Lösung:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ \mathit{IMOP}_{c_{\mathbf{s}}}(n) =_{df} \bigsqcap \{\, [\![\, p \,]\!]'(c_{\mathbf{s}}) \mid p \in \mathbf{IP}[\mathbf{s}, n] \,\}$$

wobei IP$[\mathbf{s}, n]$ die Menge der *interprozedural gültigen Pfade* von $\mathbf{s}$ nach n bezeichnet.

Der *IMaxFP*-Ansatz 1(3)

Zweistufiger Ansatz

Stufe 1: Semantik von Prozeduren (2nd-Order Effects)

$$[\![n]\!] =$$

$$\begin{cases} Id_{\mathcal{C}} & \text{falls } n \in \{ \mathbf{s}_0, \ldots, \mathbf{s}_k \} \\ \bigsqcap \{ [\![(m,n)]\!] \circ [\![m]\!] \mid m \in pred_{flowGraph(n)}(n) \} & \text{sonst} \end{cases}$$

und

$$[\![e]\!] = \begin{cases} [\![e]\!]' & \text{falls } e \in E \backslash E_{call} \\ [\![end(caller(e))]\!] & \text{sonst} \end{cases}$$

Der *IMaxFP*-Ansatz 2(3)

Stufe 2: Die "eigentliche" interprozedurale DFA

Das *IMaxFP*-Gleichungssystem

$$\textbf{inf}\ (n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s}_0 \\ \bigsqcap \{\, \textbf{inf}\ (src(e)) \mid e \in caller(\mathit{flowGraph}(n)) \,\} & \text{falls } n \in \{\mathbf{s}_1, \ldots, \mathbf{s}_k\} \\ \bigsqcap \{\, [\![(m,n)]\!](\textbf{inf}\ (m)) \mid m \in pred_{\mathit{flowGraph}(n)}(n) \,\} & \text{sonst} \end{cases}$$

Der $IMaxFP$-Ansatz 3(3)

Die $IMaxFP$-Lösung

$\forall\, c_{\mathbf{S}} \in \mathcal{C}\ \forall\, n \in N.\ IMaxFP_{c_{\mathbf{S}}}(n) =_{df} \mathbf{inf}\ ^{*}_{c_{\mathbf{S}}}(n)$

Hauptergebnisse 1(2)

Safety and Coincidence of the First Stage:

Theorem [2nd-Order]
For all $e \in E_{call}$, we have:

1. $[\![e]\!] \sqsubseteq \bigsqcap\{[\![p]\!]' \mid p \in \mathbf{CIP}[src(e), dst(e)]\}$, if the data-flow functional $[\![\ ]\!]'$ is monotonic,

2. $[\![e]\!] = \bigsqcap\{[\![p]\!]' \mid p \in \mathbf{CIP}[src(e), dst(e)]\}$, if the data-flow functional $[\![\ ]\!]'$ is distributive.

Hauptergebnisse 2(2): Sicherheit und Koinzidenz

Theorem [Interprocedural Safety]
The $IMaxFP$-solution is a lower approximation of the $IMOP$-solution, i.e.,

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ IMaxFP_{c_{\mathbf{s}}}(n) \sqsubseteq IMOP_{c_{\mathbf{s}}}(n)$$

if the data-flow functional $[\![\ ]\!]'$ is monotonic.

Theorem [Interprocedural Coincidence]
The $IMaxFP$-solution coincides with the $IMOP$-solution, i.e.,

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ IMaxFP_{c_{\mathbf{s}}}(n) = IMOP_{c_{\mathbf{s}}}(n)$$

if the data-flow functional $[\![\ ]\!]'$ is distributive.

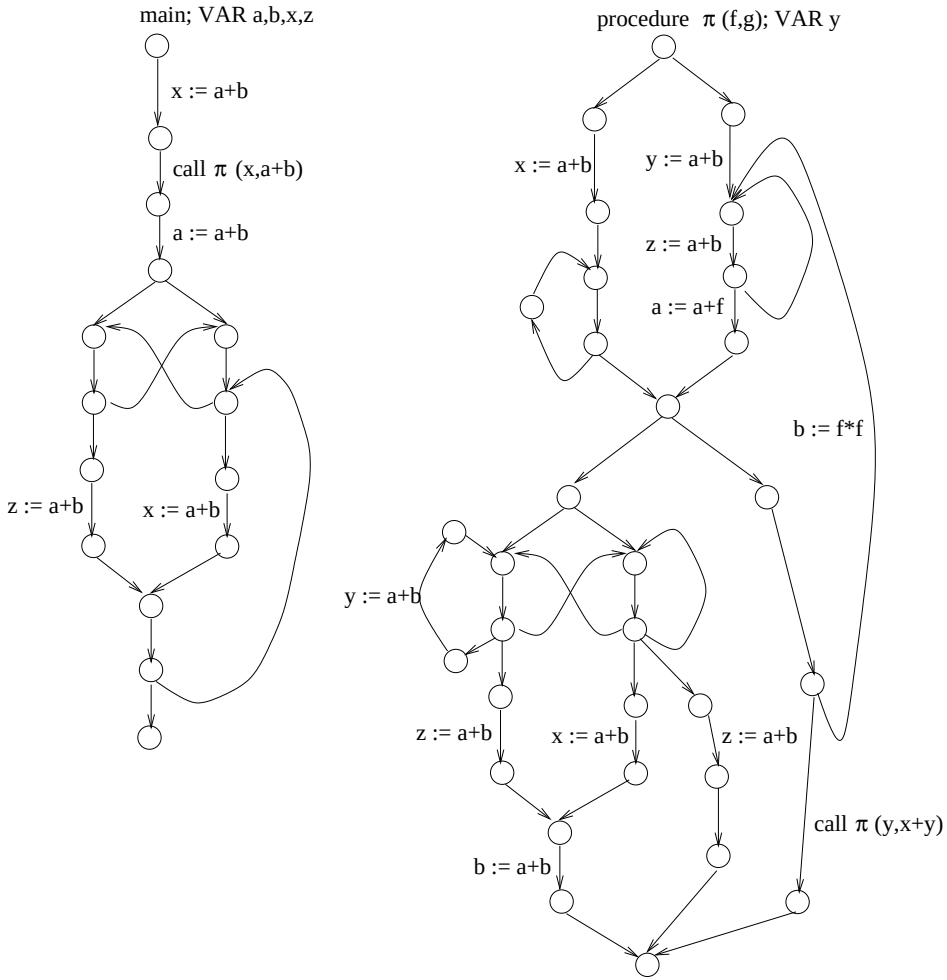
Interprozedurale Erweiterungen

Zunächst...

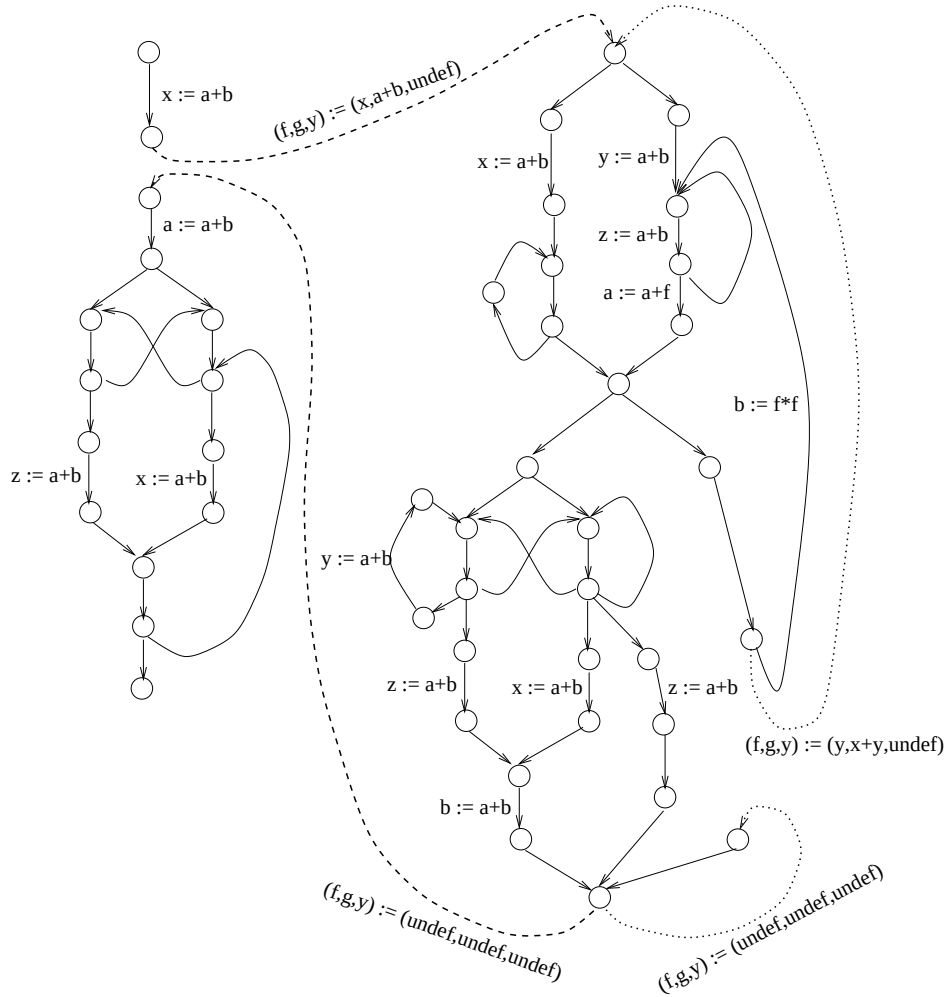
Hinzunahme von

- Wertparametern und lokalen Variablen

Interprozedurale Flussgraphsysteme



Interprozedurale Flussgraphen



Für das Weitere

Siehe ausgeteilten Umdruck!