## In der Folge

...zum Einfluss der Repräsentation:

- Basisblöcke vs. Einzelanweisungen: Vor- und Nachteile
- Weitere Beispiele konkreter Datenflussanalysen/Datenflussanalyseprobleme



## Basisblöcke: Vermeintliche Vorteile

...und die ihnen gemeinhin aus ihrer Verwendung zugeschriebenen Vorteile ("Folk Knowledge"):

- *Performanz*: "...weil weniger Knoten in die teure iterative Fixpunktberechnung involviert sind".
- *Kompaktheit*: "...weil größere Programme in den Hauptspeicher passen".



## Basisblöcke: Sichere Nachteile

...und die in der Folge aus ihrer Verwendung behaupteten Nachteile:

- *Höhere konzeptuelle Komplexität*: ... Basisblöcke führen zu einer unerwünschten *Hierarchisierung*, die sowohl theoretische Überlegungen wie Implementierungen erschwert.
- *Notwendigkeit von Prä- und Postprozessen*:: ... i.a. erforderlich, um die hierarchieinduzierten Zusatzprobleme zu behandeln (z.B. bei *dead code elimination*, *constant propagation*, ...); oder "trickbehaftete" Formulierungen nötig macht, um sie zu umgehen (z.B. bei *partial redundancy elimination*).
- *Eingeschränkte Allgemeinheit*: ... bestimmte praktisch relevante Analysen und Optimierungen sind nur schwer oder gar nicht auf Basisblockebene auszudrücken (z.B. *faint variable elimination*).



## $MOP$ -Ansatz (Einzelanweisungen)

... für kantenbenannte Einzelanweisungsgraphen:

**Die $MOP$-Lösung:**

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\iota(c_s) \mid p \in \mathbf{P}_G[s,n]\,\}$$



## $MaxFP$ -Ansatz (Einzelanweisungen)

... für kantenbenannte Einzelanweisungsgraphen:

**Die $MaxFP$-Lösung:**

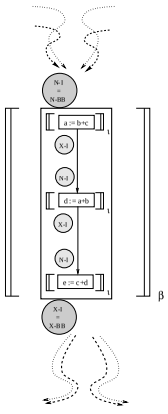
$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathsf{inf}^*_{c_s}(n)$$

wobei $\mathsf{inf}^*_{c_s}$ die größte Lösung des $MaxFP$ -Gleichungssystems bezeichnet:

$$\mathbf{inf}(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{\, [\![ (m,n) ]\!]_\iota(\mathbf{inf}(m)) \mid m \in pred_G(n)\,\} & \text{sonst} \end{cases}$$



## Hierarchisierung durch Basisblöcke






## Vereinbarung

In der Folge werden

- Basisblockknoten mit fett gesetzten Buchstaben (**m**, **n**,...)
- Einzelanweisungsknoten mit normal gesetzten Buchstaben (m, n,...)

bezeichnet.

Weiters bezeichnen

- $\llbracket\ \rrbracket_\beta$ und
- $\llbracket\ \rrbracket_\iota$

(lokale) abstrakte Datenflussanalysefunktionale auf Basisblock- bzw. Einzelanweisungs- (Instruktions-) Ebene.



## $MOP$ -Ansatz (Basisblöcke) (1)

... für knotenbenannte Basisblockgraphen:

**Die $MOP$-Lösung: (Basisblockebene)**

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, \mathbf{n} \in \mathbf{N}.\ MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\, N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ X\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\,)$$

mit

$$N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s},\mathbf{n}[\,\}$$
$$X\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s},\mathbf{n}]\,\}$$

## *MOP* -Ansatz (Basisblöcke) (2)

**Die *MOP*-Lösung: (Anweisungsebene)**

$\forall\, c_{\mathrm{s}} \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df}$
$\quad (\ N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n),\ X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n)\ )$

mit...



## *MOP* -Ansatz (Basisblöcke) (3)

...mit

$$N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df} \begin{cases} N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(\mathtt{block}(n)) \\ \quad \text{falls } n = start(\mathtt{block}(n)) \\ \\ \llbracket p \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(\mathtt{block}(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad} \\ \quad \text{von } start(\mathtt{block}(n)) \\ \quad \text{bis (ausschließlich) } n) \end{cases}$$

$$X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df} \llbracket p \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(\mathtt{block}(n)))$$
$(p \text{ Präfix von } start(\mathtt{block}(n)))$
bis (einschließlich) $n$)



## *MaxFP* -Ansatz (Basisblöcke) (1)

...für knotenbenannte Basisblockgraphen:

**Die *MaxFP*-Lösung: (Basisblockebene)**

$\forall\, c_{\mathrm{s}} \in \mathcal{C}\, \forall\, \mathbf{n} \in \mathbf{N}.\ MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathrm{s}})}(\mathbf{n}) =_{df}$

$\quad (\ N\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathrm{s}})}(\mathbf{n}),\ X\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathrm{s}})}(\mathbf{n})\ )$

mit

$N\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathrm{s}})}(\mathbf{n}) =_{df} \mathsf{pre}^{\beta}_{c_{\mathrm{s}}}(\mathbf{n})$ und

$X\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathrm{s}})}(\mathbf{n}) =_{df} \mathsf{post}^{\beta}_{c_{\mathrm{s}}}(\mathbf{n})$

wobei...



## *MaxFP* -Ansatz (Basisblöcke) (2)

...wobei $\mathsf{pre}^{\beta}_{c_{\mathrm{s}}}$ und $\mathsf{post}^{\beta}_{c_{\mathrm{s}}}$ die größten Lösungen des folgenden Gleichungssystems bezeichnen:

$$\mathbf{pre(n)} = \begin{cases} c_{\mathrm{s}} \text{ falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{\, \mathbf{post(m)} \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \,\} \text{ sonst} \end{cases}$$

$$\mathbf{post(n)} = \llbracket\, \mathbf{n} \,\rrbracket_\beta(\mathbf{pre(n)})$$



## *MaxFP* -Ansatz (Basisblöcke) (3)

**Die *MaxFP*-Lösung: (Anweisungsebene)**

$\forall\, c_{\mathrm{s}} \in \mathcal{C} \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df}$

$\quad (\ N\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n),\ X\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n)\ )$

mit

$N\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df} \mathsf{pre}^{\iota}_{c_{\mathrm{s}}}(n)$ und

$X\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_{\mathrm{s}})}(n) =_{df} \mathsf{post}^{\iota}_{c_{\mathrm{s}}}(n)$



## *MaxFP* -Ansatz (Basisblöcke) (4)

...wobei $\mathsf{pre}^{\iota}_{c_{\mathrm{s}}}$ und $\mathsf{post}^{\iota}_{c_{\mathrm{s}}}$ die größten Lösungen des folgenden Gleichungssystems bezeichnen:

$$\mathbf{pre}(n) = \begin{cases} \mathsf{pre}^{\beta}_{c_{\mathrm{s}}}(\mathtt{block}(n)) \\ \quad \text{falls } n = start(\mathtt{block}(n)) \\ \\ \mathbf{post}(m) \\ \quad \text{sonst } (m \text{ ist hier der eindeutig} \\ \quad \text{bestimmte Vorgänger von } n \\ \quad \text{in } \mathtt{block}(n)) \end{cases}$$

$$\mathbf{post}(n) = \llbracket\, n \,\rrbracket_\iota(\mathbf{pre}(n))$$



## Verfügbarkeit von Ausdrücken (1)

...für knotenbenannte BB-Graphen:

**Phase I: Die Basisblockebene**

Lokale Prädikate: (assoziiert mit BB-Knoten)

- BB-XCOMP$_\beta(t)$: $\beta$ enthält eine Anweisung $\iota$, die $t$ berechnet, und weder $\iota$ noch eine andere Anweisung von $\beta$ nach $\iota$ modifiziert einen Operanden von $t$.
- BB-TRANSP$_\beta(t)$: $\beta$ enthält keine Anweisung, die einen Operanden von $t$ modifiziert.



## Verfügbarkeit von Ausdrücken (2)

Das Gleichungssystem von Phase I:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} false & \text{falls } \beta = \mathbf{s} \\ \prod\limits_{\tilde{\beta} \in pred(\beta)} \text{BB-X-AVAIL}_{\tilde{\beta}} & \text{sonst} \end{cases}$$

$$\text{BB-X-AVAIL}_\beta = \text{BB-N-AVAIL}_\beta \cdot \text{BB-TRANSP}_\beta + \text{BB-XCOMP}_\beta$$

## Verfügbarkeit von Ausdrücken (3)

### Phase II: Die Anweisungsebene

Lokale Prädikate: (assoziiert mit EA-Knoten)

- $\mathbf{COMP}_\iota(t)$: $\iota$ berechnet $t$.
- $\mathbf{TRANSP}_\iota(t)$: $\iota$ modifiziert keinen Operanden von $t$.
- BB-N-AVAIL*, BB-X-AVAIL*: größte Lösung des Gleichungssystem von Phase I.

Das Gleichungssystem von Phase II:

$$\text{N-AVAIL}_\iota = \begin{cases} \text{BB-N-AVAIL}^*_{\texttt{block}(\iota)} & \text{falls } \iota = start(\texttt{block}(\iota)) \\ \text{X-AVAIL}_{pred(\iota)} & \text{sonst} \\ & \text{(beachte: } |pred(\iota)| = 1) \end{cases}$$

$$\text{X-AVAIL}_\iota = \begin{cases} \text{BB-X-AVAIL}^*_{\texttt{block}(\iota)} \\ \quad \text{falls } \iota = end(\texttt{block}(\iota)) \\ (\text{N-AVAIL}_\iota + \mathbf{COMP}_\iota) \cdot \mathbf{TRANSP}_\iota & \text{sonst} \end{cases}$$

## Verfügbarkeit von Ausdrücken (4)

...für knotenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit Knoten)

- $\mathbf{COMP}_\iota(t)$: $\iota$ berechnet $t$.
- $\mathbf{TRANSP}_\iota(t)$: $\iota$ modifiziert keinen Operanden von $t$.

Das Gleichungssystem:

$$\text{N-AVAIL}_\iota = \begin{cases} false & \text{falls } \iota = s \\ \prod_{\hat{\iota} \in pred(\iota)} \text{X-AVAIL}_{\hat{\iota}} & \text{sonst} \end{cases}$$

$$\text{X-AVAIL}_\iota = (\text{N-AVAIL}_\iota + \mathbf{COMP}_\iota) \cdot \mathbf{TRANSP}_\iota$$



## Verfügbarkeit von Ausdrücken (5)

...für kantenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit EA-Kanten)

- $\mathbf{COMP}_\varepsilon(t)$: Anweisung $\iota$ von Kante $\varepsilon$ berechnet $t$.
- $\mathbf{TRANSP}_\varepsilon(t)$: Anweisung $\iota$ von Kante $\varepsilon$ ändert keinen Operanden von $t$.

Das Gleichungssystem:

$$\texttt{Avail}_n = \begin{cases} false & \text{falls } n = s \\ \prod_{m \in pred(n)} (\texttt{Avail}_m + \mathbf{COMP}_{(m,n)}) \cdot \mathbf{TRANSP}_{(m,n)} & \\ & \text{sonst} \end{cases}$$



## Weitere Beispiele

- Constant folding (Konstantenfaltung)
- Faint Variable Elimination (Geistervariablenelimination)

...für die Varianten *knotenbenannte Basisblockgraphen* und *kantenbenannte Einzelanweisungsgraphen*.



## Konstantenfaltung: Einfache Konstanten

Zunächst zwei Hilfsfunktionen:

- Die Rückwärtssubstitution
- Die Zustandstransformation(sfunktion)



## Rückwärtssubstitution & Zustandstransformation (1)

Sei $\iota \equiv (x := t)$ eine Anweisung. Dann definieren wir:

- Rückwärtssubstitution
  $\delta_\iota : \mathbf{T} \to \mathbf{T}$ durch $\delta_\iota(s) =_{df} s[t/x]$ für alle $s \in \mathbf{T}$, wobei $s[t/x]$ die simultane Ersetzung aller Vorkommen von $x$ in $s$ durch $t$ bezeichnet.
- Zustandstransformation (Erinnerung)

$$\theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$



## Zusammenhang von $\delta$ und $\theta$

Bezeichne $\mathcal{I}$ die Menge aller Anweisungen.

**Substitutionslemma**

$$\forall\, t \in \mathbf{T}\ \forall \sigma \in \Sigma\ \forall\, \iota \in \mathcal{I}.\ \mathcal{E}(\delta_\iota(t))(\sigma) = \mathcal{E}(t)(\theta_\iota(\sigma))$$

*Beweis*: ...induktiv über den Aufbau von $t$.



## Einfache Konstanten (Einzelanweisungen) (1)

...für kantenbenannte Einzelanweisungsgraphen:

Bemerkung:

- $\text{CP}_n \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung

Das Gleichungssystem:

$\forall\, v \in \mathbf{V}.\ \text{CP}_n =$

$$\begin{cases} \sigma_0(v) & \text{falls } n = s \\ \bigsqcap \{\, \mathcal{E}(\delta_{(m,n)}(v))(\text{CP}_m) \mid m \in pred(n) \,\} & \text{sonst} \end{cases}$$

## Rückwärtssubstitution & Zustandstransformation (2)

Ausdehnung von $\delta$ und $\theta$ auf Pfade (und somit insbesondere auch auf Basisblöcke):

- $\Delta_p : \mathbf{T} \to \mathbf{T}$ definiert durch $\Delta_p =_{df} \delta_{n_q}$ für $q = 1$ und durch $\Delta_{(n_1,\ldots,n_{q-1})} \circ \delta_{n_q}$ für $q > 1$
- $\Theta_p : \Sigma \to \Sigma$ definiert durch $\Theta_p =_{df} \theta_{n_1}$ für $q = 1$ und durch $\Theta_{(n_2,\ldots,n_q)} \circ \theta_{n_1}$ für $q > 1$.



## Zusammenhang von $\Delta$ und $\Theta$

Bezeichne $\mathcal{B}$ die Menge aller Basisblöcke.

**Verallgemeinertes Substitutionslemma**

$$\forall t \in \mathbf{T}\ \forall \sigma \in \Sigma\ \forall \beta \in \mathcal{B}.\ \mathcal{E}(\Delta_\beta(t))(\sigma) = \mathcal{E}(t)(\Theta_\beta(\sigma))$$

*Beweis*: ...induktiv über die Länge von $p$.



## Einfache Konstanten (Basisblöcke) (1)

...für knotenbenannte Basisblockgraphen:

**Phase I: Basisblockebene**

Bemerkung:

- $\Delta_\beta(v) =_{df} \delta_{\iota_1} \circ \ldots \circ \delta_{\iota_q}(v)$, wobei $\beta \equiv \iota_1; \ldots; \iota_q$.
- BB-N-CP$_\beta$, BB-X-CP$_\beta$, N-CP$_\iota$, X-CP$_\iota \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung



## Einfache Konstanten (Basisblöcke) (2)

Das Gleichungssystem von Phase I:

$$\text{BB-N-CP}_\beta \;=\; \begin{cases} \sigma_0 \ \text{ falls } \beta = \mathrm{s} \\ \bigsqcap\{\text{BB-X-CP}_{\tilde\beta} \mid \tilde\beta \in pred(\beta)\} \\ \text{ sonst} \end{cases}$$

$$\forall v \in \mathbf{V}.\ \text{BB-X-CP}_\beta(v) \;=\; \mathcal{E}(\Delta_\beta(v))(\text{BB-N-CP}_\beta)$$



## Einfache Konstanten (Basisblöcke) (3)

**Phase II: Anweisungsebene**

Vorberechnete Resultate (aus Phase I):

- BB-N-CP$^\star$, BB-X-CP$^\star$: die größte Lösung des Gleichungssystems von Phase I.



## Einfache Konstanten (Basisblöcke) (4)

Das Gleichungssystem von Phase II:

$$\text{N-CP}_\iota \;=\; \begin{cases} \text{BB-N-CP}^\star_{\mathtt{block}(\iota)} \\ \ \text{falls } \iota = start(\mathtt{block}(\iota)) \\ \text{X-CP}_{pred(\iota)} \\ \ \text{sonst (beachte: } |pred(\iota)| = 1) \end{cases}$$

$$\forall v \in \mathbf{V}.\ \text{X-CP}_\iota(v) \;=\; \begin{cases} \text{BB-X-CP}^\star_{\mathtt{block}(\iota)}(v) \\ \ \text{falls } \iota = end(\mathtt{block}(\iota)) \\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) \ \text{ sonst} \end{cases}$$



## Geistervariablenelimination (1)

...für kantenbenannte Einzelanweisungsgraphen:

Lokale Prädikate: (assoziiert mit Einzelanweisungskanten)

- USED$_\varepsilon(v)$: Anweisung $\iota$ von Kante $\varepsilon$ benutzt $v$.
- MOD$_\varepsilon(v)$: Anweisung $\iota$ von Kante $\varepsilon$ modifiziert $v$.
- RELV-USED$_\varepsilon(v)$: $v$ ist eine Variable, die in der Anweisung $\iota$ von Kante $\varepsilon$ vorkommt und von dieser Anweisung "zu leben gezwungen" wird (z.B. für $\iota$ eine Ausgabeanweisung).
- ASS-USED$_\varepsilon(v)$: $v$ ist eine in der Zuweisung $\iota$ von Kante $\varepsilon$ rechtsseitig vorkommende Variable.



## Geistervariablenelimination (2)

Das Gleichungssystem:

$$\text{FAINT}_n(v) =$$
$$\prod_{m \in succ(n)} \overline{\text{RELV-USED}_{(n,m)}(v)} \cdot$$
$$(\text{FAINT}_m(v) \ + \ \text{MOD}_{(n,m)}(v)) \cdot$$
$$(\text{FAINT}_m(LhsVar_{(n,m)}) + \overline{\text{ASS-USED}_{(n,m)}(v)})$$

## Geistervariablenelimination (3)

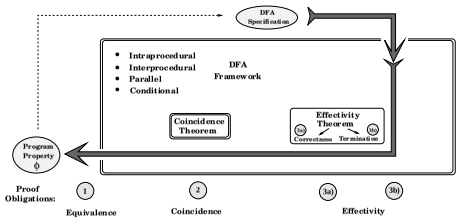
...ein typisches Beispiel für ein DFA-Problem, für das eine Formulierung

- auf (knoten- und kantenbenannten) Einzelanweisungsgraphen offensichtlich ist,
- auf (knoten- und kantenbenannten) Basisblockgraphen alles andere als ersichtlich ist.



## Für uns reicht...

...die allgemeine Werkzeugkistensicht und das Wissen, dass je nach Repräsentationsvariante unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen entstehen.






## Optimale Programmoptimierung

Zum Aufwärmen...

- Einige einfache Beispiele (siehe Tafel)

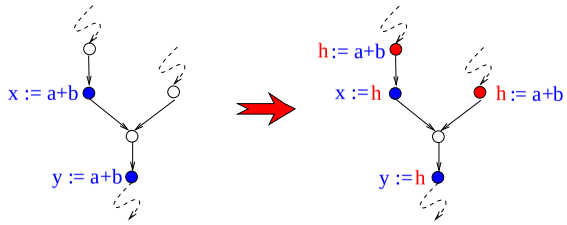
Anschließend zum echten Einstieg...

- *Partielle Redundanzelimination* (*PRE*) alias *Code Motion* (*CM*)



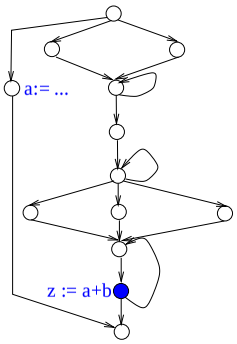
## PRE – Worum geht es?

...im Kern um die Vermeidung von Mehrfachberechnungen von Werten



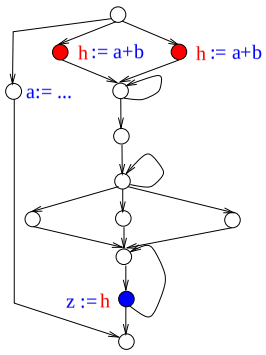



## PRE – besonders wirksam im Zshg. mit Schleifen



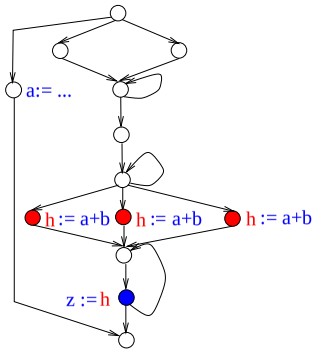



## Ein redundanzfreies Programm



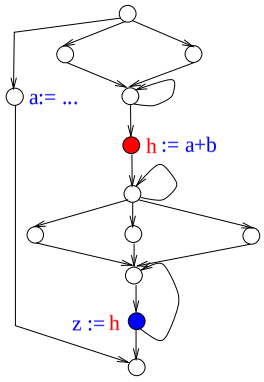



## Aber welches soll es sein? Dieses? Das vorige?






## Oder vielleicht dieses?

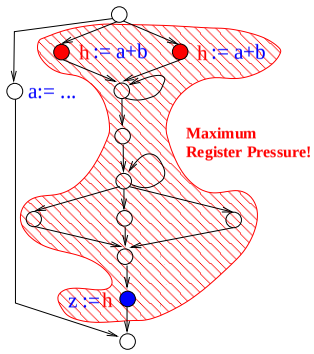


## Auf die (Optimierungs-) Ziele kommt es an!



## Die erste Transformation

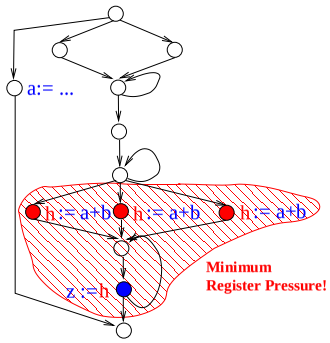
...redundanzfrei, aber maximaler Registerdruck






## Die zweite Transformation

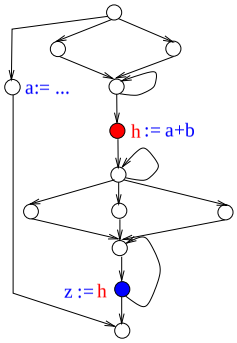
...ebenfalls redundanzfrei, aber mit minimalem Registerdruck!






## Die dritte Transformation

...redundanzfrei, moderater Registerdruck, aber keine Codereplikation!






## Auf die (Optimierungs-) Ziele kommt es an!

In unseren Beispielen...

- *Performanz*: Vermeidung von Mehrfachberechnungen
  $\rightsquigarrow$ Berechnungsqualität/-optimalität
- *Registerdruck*: Vermeidung unnötigen Schiebens
  $\rightsquigarrow$ Lebenszeitqualität/-optimalität
- *Platz*: Vermeidung von Codereplikation
  $\rightsquigarrow$ Platzqualität/-optimalität



## Zum Nachdenken

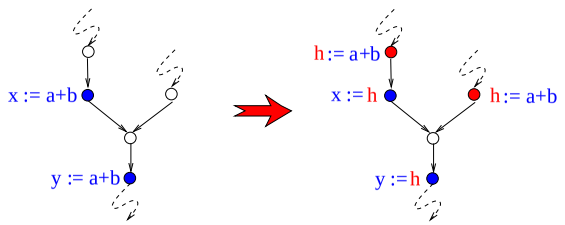
Sei $P$ ein Programm mit Vorkommen partiell redundanter Berechnungen.

Ist es immer möglich, $P$ so in ein Programm $P'$ zu transformieren, dass $P$ und $P'$ bedeutungsgleich sind, $P'$ aber frei von partiell redundanten Berechnungen ist?



## In Medias Res – PRE

*Ziel*: ...die Vermeidung von Mehrfachberechnungen von Werten






## Bezeichnungen (1)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph. Dann bezeichnen:

- $pred(n) =_{df} \{m \mid (m, n) \in E\}$: Menge aller *Vorgänger*
- $succ(n) =_{df} \{m \mid (n, m) \in E\}$: Menge aller *Nachfolger*
- $source(e)$, $dest(e)$: *Anfangs-* und *Endknoten* einer Kante
- *Endlicher Pfad*: Kantenfolge $(e_1, \ldots, e_k)$ mit $dest(e_i) = source(e_{i+1})$ für alle $1 \leq i < k$
- Statt Kantenfolgen betrachten wir entsprechend auch Knotenfolgen als Pfade, so zweckmäßig.

## Bezeichnungen (2)

- $p = \langle e_1, \ldots, e_k \rangle$ *Pfad von* $m$ *nach* $n$, falls $source(e_1) = m$ und $dest(e_k) = n$
- $\mathbf{P}[m, n]$: Menge aller Pfade von $m$ nach $n$
- $\lambda_p$: *Länge* von $p$, d.h. die Anzahl der Kanten von $p$
- $\varepsilon$: Pfad der Länge 0
- $N_J \subseteq N$: Menge der *Join*-Knoten, d.h. Menge der Knoten mit mehr als einem Vorgänger
- $N_B \subseteq N$: Menge der *Branch*-Knoten, d.h. Menge der Knoten mit mehr als einem Nachfolger



## Vereinbarung

Ohne Beschränkung der Allgemeinheit...

- Jeder Knoten in einem Flussgraphen liegt auf einem Pfad von $\mathbf{s}$ nach $\mathbf{e}$

  *Intuition*: Es gibt keine unerreichbaren Teile in einem Flussgraphen.

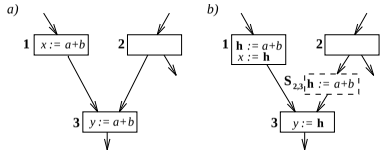
...eine generell übliche Vereinbarung für Analyse und Optimierung!



## Kritische Kanten

Eine Kante heißt *kritisch*, wenn sie von einem branch- zu einem join-Knoten führt.

*Zur Illustration*: ...mit Knoten $\mathbf{S}_{2,3}$ als *künstlichem* (*synthetic*) *Knoten*, der die kritische Kante von Knoten **2** nach **3** spaltet.






## Verfahrensspezifische Vereinbarung

Ohne Beschränkung der Allgemeinheit...

In der Folge betrachten wir Flussgraphen...

- in Form knotenbenannter EA-Graphen,
- bei denen alle Kanten, die in einem join-Knoten enden, durch Einfügen eines sog. *künstlichen* Knotens aufgespalten sind,

...eine PRE-spezifische Vereinbarung.



## Hintergrund

...dieser Vereinbarung:

- Der PRE-Prozess vereinfacht sich dadurch.

  $\leadsto$ *Berechnungsoptimale* Ergebnisse können bereits erzielt werden, wenn erforderliche Hilfsvariableninitialisierungen einheitlich an Knotenanfängen erfolgen.



## Bemerkung

Berechnungsoptimale Ergebnisse sind auch möglich, wenn ausschließlich kritische Kanten gespalten werden.

Dann aber muss ein PRE-Algorithmus in der Lage sein, sowohl N- als auch X-Initialisierungen (an Knoten) durchführen zu können.

Prinzipiell ist das kein Problem; mit vorstehender Vereinbarung ist die Präsentation des PRE-Algorithmus aber noch einfacher.



## Arbeitsplan

In der Folge werden wir definieren...

- Die Menge der PRE-Transformationen
- Die Menge der *zulässigen* PRE-Transformationen
- Die Menge der *berechnungsoptimalen* PRE-Transformationen
- Die BCM-Transformation als spezielle berechnungsoptimale PRE-Transformation



## Die Menge der PRE-Transformationen

Generelles (Transformations-) Muster für einen Term $t$...

- Deklariere eine neue Hilfvariable $h$ für $t$ in $G$
- Füge an einigen Knoten von $G$ die Anweisung $h := t$ ein
- Ersetze einige der originalen Vorkommen von $t$ in $G$ durch $h$

*Bem.*: $t$ wird oft auch als *Kandidatenausdruck* bezeichnet.

## Beobachtung

Zwei (auf Knoten definierte) Prädikate

- $Insert_{CM}$
- $Repl_{CM}$

sind ausreichend, eine PRE- (bzw. CM-) Transformation vollständig zu beschreiben (beachte: die Deklaration der Hilfsvariablen $h$ ist für jede CM-Transformation identisch und braucht deshalb nicht gesondert betrachtet zu werden).



## CM-Transformationen

...bezeichne $\mathcal{CM}$ die Menge aller CM-Transformationen (für den Kandidatenausdruck $t$).



## Beobachtung

Offenbar ist nicht jede Transformation in $\mathcal{CM}$ bedeutungserhaltend und damit akzeptabel.

Das führt uns auf den Begriff der *zulässigen* CM-Transformationen...



## Zulässige CM-Transformationen

Sei $CM \in \mathcal{CM}$.

$CM$ heißt *zulässig*, wenn $CM$ *sicher* und *korrekt* ist.

Intuition:

- *Sicher*: ...es gibt keinen Pfad, auf dem durch Einfügen einer Initialisierung ein neuer Wert berechnet wird.
- *Korrekt*: ...die Hilfsvariable ist an jeder Benutzungsstelle "richtig" initialisiert, d.h. sie enthält denselben Wert, den eine Neuberechnung von $t$ an dieser Stelle liefert.



## Zur Formalisierung

...sind folgende (lokale) Prädikate erforderlich.

- $Comp(n)$: $n$ enthält ein Vorkommen des Kandidatenausdrucks $t$.
- $Transp(n)$: $n$ ist transparent für $t$, d.h., $n$ weist keinem Operanden von $t$ einen (neuen) Wert zu.

Ebenfalls nützlich:

- $Comp_{CM}(n) =_{df} Insert_{CM}(n) \vee Comp(n) \wedge \neg Repl_{CM}(n)$: Programmpunkte, an denen nach Anwendung von $CM$ der Ausdruck $t$ berechnet wird.



## Globalisierung von Prädikaten auf Pfade

Sei $p$ ein Pfad und bezeichne $p_i$ den i-ten Knoten von $p$.

Mit diesen Bezeichnungen treffen wir die folgende Vereinbarung:

- $Predicate^{\forall}(p) \iff \forall\, 1 \leq i \leq \lambda_p.\ Predicate(p_i)$
- $Predicate^{\exists}(p) \iff \exists\, 1 \leq i \leq \lambda_p.\ Predicate(p_i)$



## Sicherheit und Korrektheit

**Definition** [Sicherheit und Korrektheit]
Sei $n \in N$. Wir definieren:

1. $Safe(n) \iff_{df}$
   $\forall \langle n_1, \ldots, n_k \rangle \in \mathbf{P}[s, e]\ \forall i.\ (n_i = n) \Rightarrow$
   $i)\ \exists j < i.\ Comp(n_j) \wedge Transp^{\forall}(\langle n_j, \ldots, n_{i-1} \rangle)\ \vee$
   $ii)\ \exists j \geq i.\ Comp(n_j) \wedge Transp^{\forall}(\langle n_i, \ldots, n_{j-1} \rangle)$
2. Sei $CM \in \mathcal{CM}$. Dann:
   $Correct_{CM}(n) \iff_{df} \forall \langle n_1, \ldots, n_k \rangle \in \mathbf{P}[s, n]$
   $\exists i.\ Insert_{CM}(n_i) \wedge Transp^{\forall}(\langle n_i, \ldots, n_{k-1} \rangle)$



## Aufwärts- und Abwärtssicherheit

Die Einschränkung der Definition für *Sicherheit* auf (i) bzw. (ii) führt auf die Begriffe

- *Aufwärtssicherheit*
- *Abwärtssicherheit*

## Intuition

Eine Berechnung $t$ ist an einer Programmstelle $n$

- *aufwärtssicher*, wenn $t$ auf allen Pfaden von s nach $n$ berechnet wird und auf die jeweils letzte Berechnung von $t$ keine Modifikation eines Operanden von $t$ mehr erfolgt.
- *abwärtssicher*, wenn $t$ auf allen Pfaden von $n$ nach e berechnet wird und der jeweils ersten Berechnung von $t$ keine Modifikation eines Operanden von $t$ vorausgeht.



## Aufwärts- und Abwärtssicherheit

**Definition** [Aufwärts- und Abwärtssicherheit]

1. $\forall\, n \in N.$ *U-Safe*$(n) \iff_{df}$
   $\forall\, p \in \mathbf{P}[s, n]\ \exists\, i < \lambda_p.$ *Comp* $(p_i) \wedge$ *Transp* $^{\forall}(\, p[i, \lambda_p[\,)$
2. $\forall\, n \in N.$ *D-Safe*$(n) \iff_{df}$
   $\forall\, p \in \mathbf{P}[n, e]\ \exists\, i \leq \lambda_p.$ *Comp* $(p_i) \wedge$ *Transp* $^{\forall}(\, p[1, i[\,)$



## Zulässige CM-Transformationen

Damit können wir jetzt genauer definieren...

**Definition** [Zulässige CM-Transformation]
Eine CM-Transformation $CM \in \mathcal{CM}$ heißt *zulässig* gdw für jeden Knoten $n \in N$ gelten folgende beide Eigenschaften:

1. $Insert_{CM}(n) \Rightarrow$ *Safe*$(n)$
2. $Repl_{CM}(n) \Rightarrow$ *Correct*$_{CM}(n)$

Die Menge aller zulässigen CM-Transformationen bezeichnen wir mit $\mathcal{CM}_{Adm}$.



## Erste Aussagen... (1)

**Korrektheitslemma**

$$\forall\, CM \in \mathcal{CM}_{Adm}\ \forall\, n \in N.\ \textit{Correct}_{CM}(n) \Rightarrow \textit{Safe}(n)$$



## Erste Aussagen... (2)

**Sicherheitslemma**

$$\forall\, n \in N.\ \textit{Safe}(n) \iff \textit{D-Safe}(n) \vee \textit{U-Safe}(n)$$



## Berechnungsbesser, berechnungsoptimal

Eine CM-Transformation $CM \in \mathcal{CM}_{Adm}$ heißt *berechnungsbesser* als eine CM-Transformation $CM' \in \mathcal{CM}_{Adm}$ gdw

$$\forall\ p \in \mathbf{P}[s, e].\ |\ \{i \mid \textit{Comp}_{CM}(p_i)\}\ | \leq |\ \{i \mid \textit{Comp}_{CM'}(p_i)\}\ |$$

*Bemerkung*: Die Relation "berechnungsbesser" ist eine Quasiordnung, d.h. eine reflexive und transitive Relation.



## Berechnungsoptimalität

**Definition** [Berechnungsoptimale CM-Transformation]
Eine zulässige CM-Transformation $CM \in \mathcal{CM}_{Adm}$ heißt *berechnungsoptimal* gdw $CM$ ist berechnungsbesser als jede andere zulässige CM-Transformation.

Wir bezeichnen die Menge der berechnungsoptimalen CM-Transformationen mit $\mathcal{CM}_{CmpOpt}$.



## Konzeptuell

...PRE kann als zweistufiger Prozess gesehen werden

1. Vorziehen von Ausdrücken (Expression hoisting)
   ...vorziehen von Ausdrücken an "frühere" sichere Berechnungspunkte
2. Beseitigung total redundanter Ausdrücke (Total redundancy elimination)
   ...beseitigen von Berechnungen, die durch das Vorziehen total redundant geworden sind

## Extreme Strategie – Frühestheitprinzip

Platziere Berechnungen so früh wie möglich...

- Theorem [Berechnungsoptimalität]
  ...vorziehen von Ausdrücken zu ihren frühesten sicheren Berechnungspunkten liefert berechnungsoptimale Programme

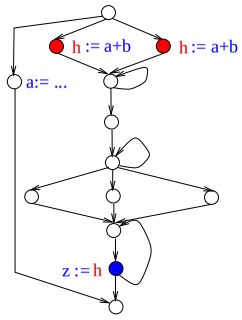
  $\rightsquigarrow$ ...bekannt als Busy Code Motion



## Frühestheitprinzip

Platzieren von Berechnungen so früh wie möglich...

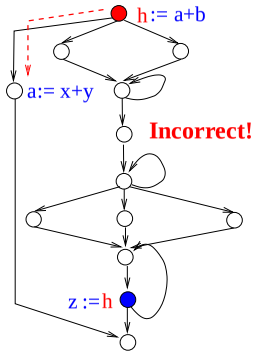
...liefert berechnungsoptimale Programme.






## Beachte: Frühest heißt in der Tat...

...so früh wie möglich, aber nicht früher!






## Busy Code Motion

*Intuition*:

Platziere Berechnungen *so früh wie möglich* im Programm, ohne Sicherheit und Korrektheit zu verletzen!

*Beachte*: Berechnungen werden dadurch so weit wie möglich entgegen des Kontrollflusses verschoben

$\rightsquigarrow$ ...liefert die Motivation für die Wahl der Bezeichnung *busy*.



## Frühestheit

**Definition** [Frühestheit]

$\forall\, n \in N.\ \texttt{Earliest}(n) =_{df} \mathit{Safe}(n) \wedge$

$$\begin{cases} \mathit{true} & \text{falls } n = s \\ \bigvee\limits_{m \in \mathit{pred}(n)} \neg \mathit{Transp}(m) \vee \neg \mathit{Safe}(m) & \text{sonst} \end{cases}$$



## Die BCM-Transformation

- $\mathit{Insert}_{BCM}(n) =_{df} \texttt{Earliest}(n)$
- $\mathit{Repl}_{BCM}(n) =_{df} \mathit{Comp}(n)$



## Das BCM-Theorem

**BCM-Theorem**

Die $BCM$-Transformation ist berechnungsoptimal, d.h., $BCM \in \mathcal{CM}_{CmpOpt}$.

Der Beweis für das BCM-Theorem stützt sich ab auf das *Frühestheit*- und das *BCM-Lemma*...



## Das Frühestheitlemma

**Frühestheitlemma**

Sei $n \in N$. Dann gilt:

1. $\mathit{Safe}(n) \Rightarrow \forall\, p \in \mathbf{P}[s,n]\ \exists\, i \leq \lambda_p.$
   $\texttt{Earliest}(p_i) \wedge\ \mathit{Transp}^{\forall}(p[i, \lambda_p[\,)$
2. $\texttt{Earliest}(n) \iff$
   $\mathit{D\text{-}Safe}(n) \wedge \bigwedge\limits_{m \in \mathit{pred}(n)} (\neg \mathit{Transp}(m) \vee \neg \mathit{Safe}(m))$
3. $\texttt{Earliest}(n) \iff$
   $\mathit{Safe}(n) \wedge\ \forall\, CM \in \mathcal{CM}_{Adm}.\ \mathit{Correct}_{CM}(n) \Rightarrow \mathit{Insert}_{CM}(n)$

## Das BCM-Lemma

**BCM-Lemma**

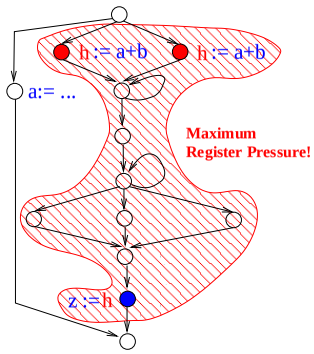
Sei $p \in \mathbf{P}[s,e]$. Dann gilt:

1. $\forall\, i \leq \lambda_p.\ \mathit{Insert}_{BCM}(p_i) \iff$
   $\exists\, j \geq i.\ p[i,j] \in \mathit{FU\text{-}LtRg}(BCM)$
2. $\forall\, CM \in \mathcal{CM}_{Adm}\ \forall\, i,\ j \leq \lambda_p.\ p[i,j] \in \mathit{LtRg}(BCM) \Rightarrow$
   $\mathit{Comp}^{\exists}_{CM}(p[i,j])$
3. $\forall\, CM \in \mathcal{CM}_{CmpOpt}\ \forall\, i \leq \lambda_p.\, \mathit{Comp}_{CM}(p_i) \Rightarrow$
   $\exists\, j \leq i \leq l.\ p[j,l] \in \mathit{FU\text{-}LtRg}(BCM)$



## Die BCM-Transformation

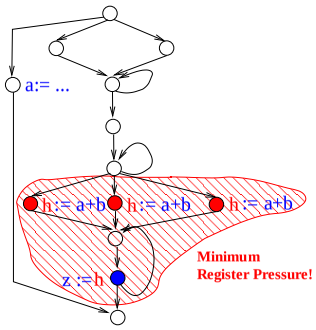
...berechnungsoptimal, aber maximaler Registerdruck






## Die LCM-Transformation

...ebenfalls berechnungsoptimal, aber mit minimalem Registerdruck!






## Duale extreme Strategie – Spätestheitprinzip

Platziere Berechnungen so spät wie möglich...

- Theorem [Optimalität]
  ...vorziehen von Ausdrücken so wenig wie möglich, aber so weit wie nötig (um berechnungsoptimal zu werden), liefert berechnungsoptimale Programme
  mit minimalem Registerdruck

  $\leadsto$ ...bekannt als Lazy Code Motion



## Lazy Code Motion

*Intuition*:

Platziere Berechnungen *so spät wie möglich* im Programm, ohne Sicherheit, Korrektheit und Berechnungsoptimalität zu verletzen!

*Beachte*: Berechnungen werden dadurch so wenig wie möglich entgegen des Kontrollflusses verschoben

$\leadsto$ ...liefert die Motivation für die Wahl der Bezeichnung *lazy*.



## Arbeitsplan

In der Folge werden wir definieren...

- Die Menge der *lebenszeitoptimalen* PRE-Transformationen
- Die LCM-Transformation als eindeutig bestimmte einzige lebenszeitoptimale PRE-Transformation



## Vorschau auf die weiteren Vorlesungstermine...

- Mo, 03.12.2007: Vorlesung von 16:15 Uhr bis 17:45 Uhr im Hörsaal 14, TU-Hauptgebäude
- Mo, 10.12.2007: Vorlesung von 16:15 Uhr bis 17:45 Uhr im Hörsaal 14, TU-Hauptgebäude
- *Mo, 17.12./24.12./31.12.2007: Keine Vorlesung(en)!* (*Ferialzeit*)