

Optimierende Compiler

Partielle Redundanzeliminierung (PRE)

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

Sommersemester 2010

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Einleitung

- Finde wiederholte Berechnungen auf einem Ausführungspfad
- ... und beseitige alle außer der ersten
- Bisher kennengelernte Verfahren
 - LVN, SVN, DVN
 - GCSE

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Totale Redundanz

Ein Ausdruck ist **total** redundant, wenn er auf **allen** Pfaden zu einer redundanten Verwendung berechnet wird.

Partielle Redundanz

Ein Ausdruck ist **partiell** redundant, wenn er auf **einigen, aber nicht auf allen** Pfaden zu einer redundanten Verwendung berechnet wird.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

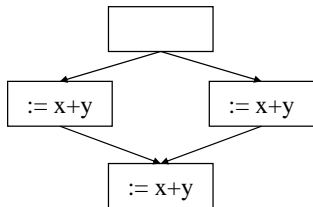
Datenfluss

Totale Redundanz

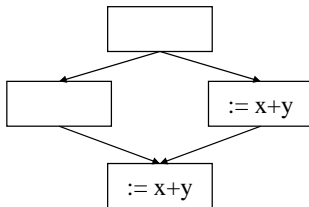
Ein Ausdruck ist **total** redundant, wenn er auf **allen** Pfaden zu einer redundanten Verwendung berechnet wird.

Partielle Redundanz

Ein Ausdruck ist **partiell** redundant, wenn er auf **einigen, aber nicht auf allen** Pfaden zu einer redundanten Verwendung berechnet wird.



fully redundant



partially redundant

CSE erkennt nur **totale** Redundanz.

Quelle: CMU CS 15-745 2009

OC

A. Koch

Einleitung

PRE

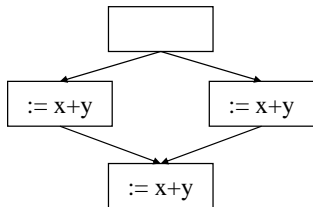
Konzepte

E-Pfad PRE

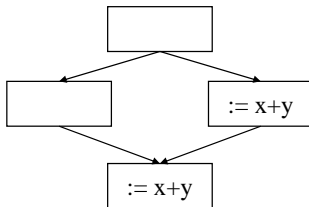
Notation

E-Pfade

Datenfluss



fully redundant



partially redundant

CSE erkennt nur **totale** Redundanz.

- Loop Invariant Code Motion
- Bewege Anweisungen, die jede Iteration denselben Wert liefern
 - Dafür müssen die Operanden **schleifeninvariant** sein
- ... aus der Schleife **heraus**

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Ein Operand ist **schleifeninvariant**, wenn

- er **konstant** ist, oder
- alle seine Definitionen **außerhalb** der Schleife liegen, oder
 - Erinnerung: Datenflußproblem REACHES
“Erreichende Definitionen”
- er eine einzelne Definition **innerhalb** der Schleife hat, die aber selbst invariant ist.

Naiver Ansatz: Bewege invariante Anweisungen **S** vor Schleifenkopf (pre-header).

Geht aber nur, wenn jedes **S**

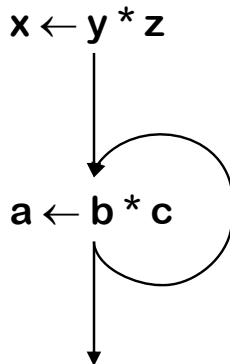
- alle **Verwendungen** seiner LHS dominiert
- alle **Schleifenausgänge** dominiert
(`break`, `continue`, ...)

Naiver Ansatz: Bewege invariante Anweisungen **S** vor Schleifenkopf (pre-header).

Geht aber nur, wenn jedes **S**

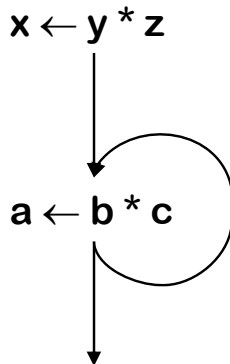
- alle **Verwendungen** seiner LHS dominiert
- alle **Schleifenausgänge** dominiert
(**break**, **continue**, ...)

Schleifeninvariante
Ausdrücke sind eine Art der
partiellen Redundanz



$b * c$ nur redundant auf Rückwärtskante.

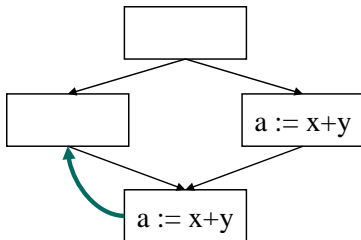
Schleifeninvariante
Ausdrücke sind eine Art der
partiellen Redundanz

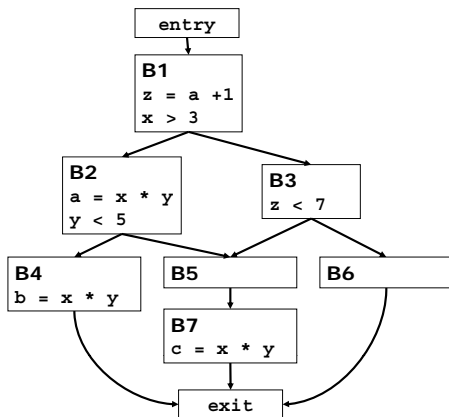


b*c nur redundant auf Rückwärtskante.

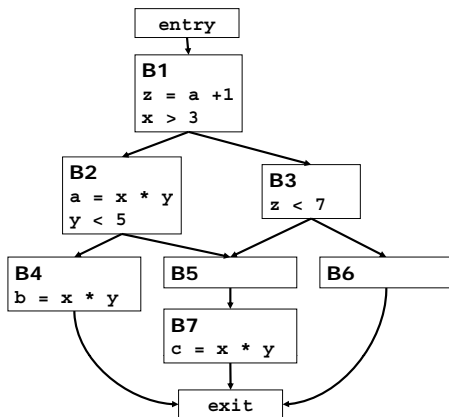
Eliminierung partieller Redundanz

- Bewege partiell redundante Berechnungen an ihre **optimalen** Stellen
 - Vermeide so Doppelberechnungen
- Beinhaltet CSE und LICM

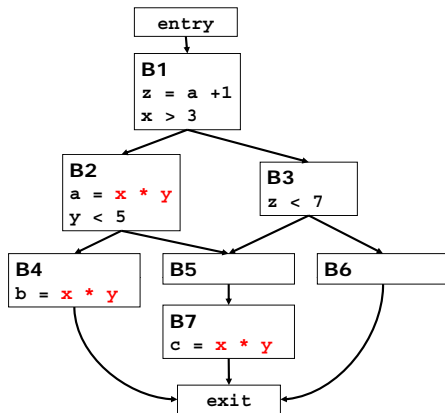




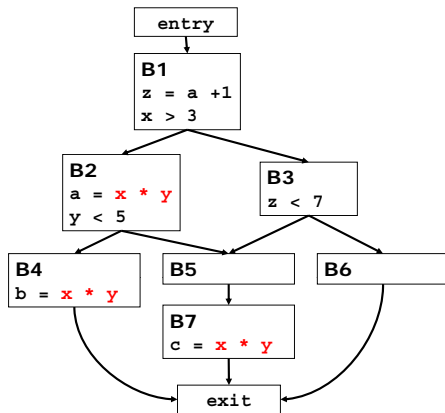
Welche Ausdrücke sind partiell redundant?



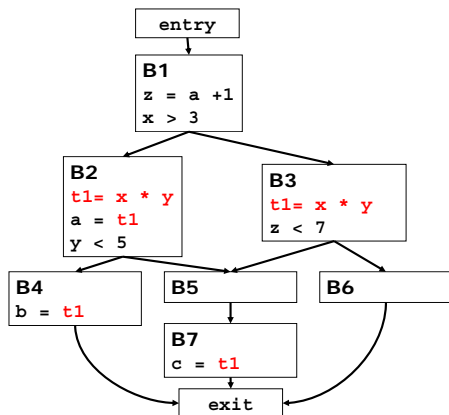
Welche Ausdrücke sind partiell redundant?



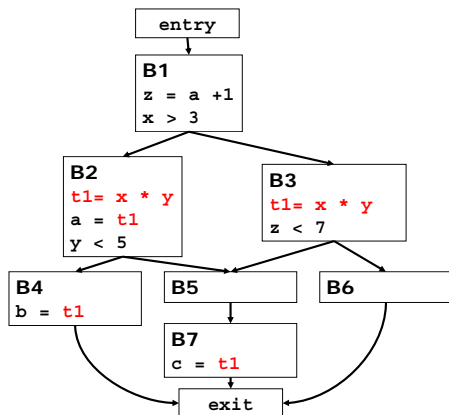
Wo die Berechnung tatsächlich durchführen, wo die Kopien verwenden?



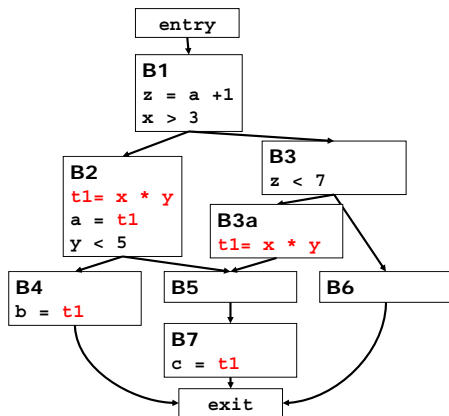
Wo die Berechnung tatsächlich durchführen, wo die Kopien verwenden?



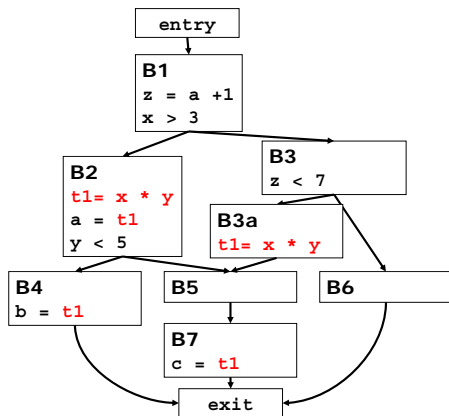
Ist das die optimale Lösung?



Ist das die optimale Lösung?

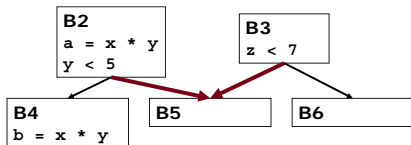


Was ist besonders an Kante (B3, B5)?

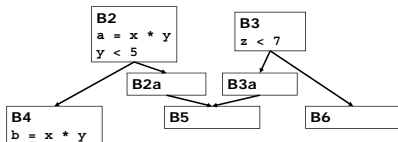


Was ist besonders an Kante (**B3**, **B5**)?

Vor Aufteilen

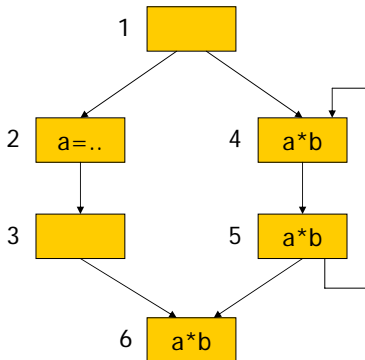


Alle kritischen Kanten aufgeteilt



Nachteil: Potentiell langsamer (Compile- und Laufzeit!)

- CSE: $a*b$ in **B5**
- LICM: $a*b$ in **B4**
- Bewegung: $a*b$ von **B6** nach **B3**



Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminatibility paths (E-paths)

Optimale Lebenszeit

Die Lebenszeit von einer Neuberechnung zu einer Verwendung sollte so **kurz** wie möglich sein.

➔ Benötigt weniger Register

Berechnungsoptimalität

Zur Programmlaufzeit sollen so **wenige** Berechnungen wie möglich ausgeführt werden.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Optimale Lebenszeit

Die Lebenszeit von einer Neuberechnung zu einer Verwendung sollte so **kurz** wie möglich sein.

➔ Benötigt weniger Register

Berechnungsoptimalität

Zur Programmlaufzeit sollen so **wenige** Berechnungen wie möglich ausgeführt werden.

OC

A. Koch

Einleitung

PRE

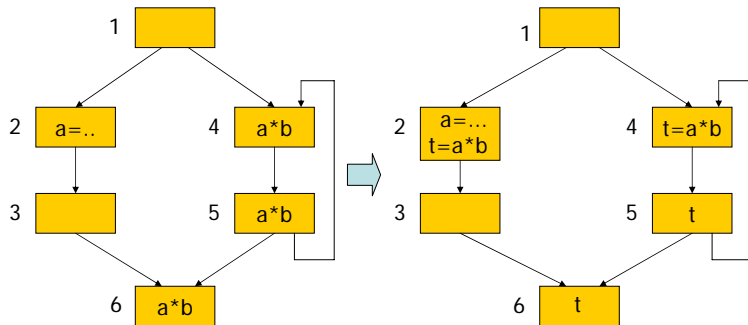
Konzepte

E-Pfad PRE

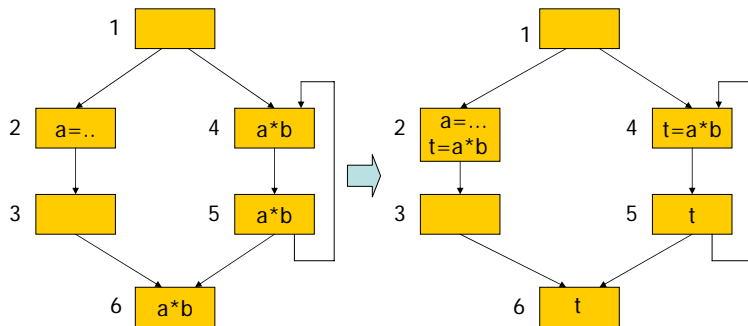
Notation

E-Pfade

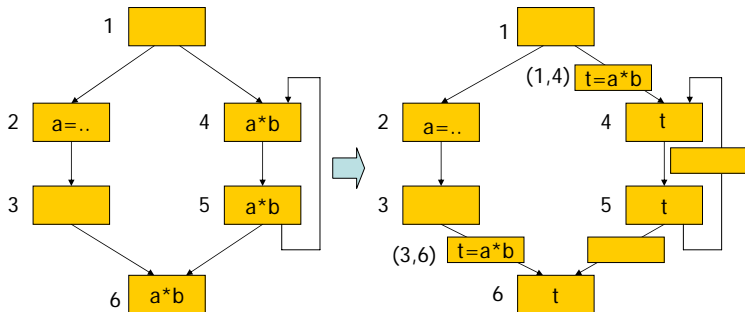
Datenfluss



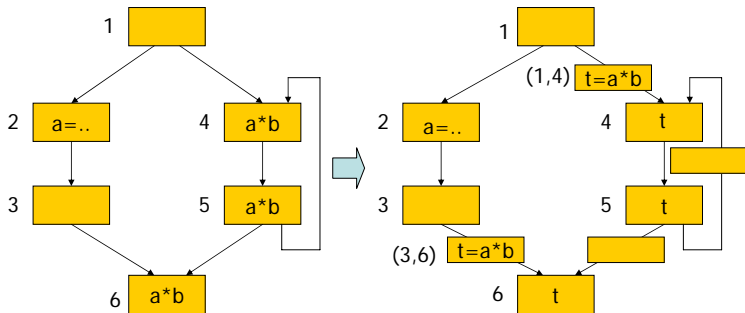
- Neue Berechnung von $a*b$ eingefügt in **B2**
- **B3** wäre besser (kürzere Lebenszeit!)
- $a*b$ in **B4** nicht berechnungsoptimal
 - Wegen (**B1,B2**) nicht am Ende von **B1** eingefügt
- $a*b$ gesichert in **B2+B4**, wiederverwendet in **B5+B6**



- Neue Berechnung von $a*b$ eingefügt in **B2**
- **B3** wäre besser (kürzere Lebenszeit!)
- $a*b$ in **B4** nicht berechnungsoptimal
 - Wegen (**B1,B2**) nicht am Ende von **B1** eingefügt
- $a*b$ gesichert in **B2+B4**, wiederverwendet in **B5+B6**



- Alle Kanten zu Join-Knoten aufteilen
- $a*b$ einfügen in (B3,B6) (hat optimale Lebenszeit!)
- $a*b$ einfügen in (B1,B4) (berechnungsoptimal)
- Erzeugt zusätzliche Blöcke (leere entfernbare)



- Alle Kanten zu Join-Knoten aufteilen
- $a*b$ einfügen in **(B3,B6)** (hat optimale Lebenszeit!)
- $a*b$ einfügen in **(B1,B4)** (berechnungsoptimal)
- Erzeugt zusätzliche Blöcke (leere entfernbar)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Konzepte

Verfügbarkeit (Availability)

Ein Ausdruck e ist an einer Programmstelle p verfügbar, wenn sein Wert auf **allen** Pfaden von Programmanfang zu p berechnet wird.

↳ Totale Redundanz von e an p

Partielle Verfügbarkeit (Partial Availability)

Ein Ausdruck e ist an einer Programmstelle p partiell verfügbar, wenn sein Wert auf **einigen** Pfaden von Programmanfang zu p berechnet wird.

↳ Partielle Redundanz von e an p

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Verfügbarkeit (Availability)

Ein Ausdruck e ist an einer Programmstelle p verfügbar, wenn sein Wert auf **allen** Pfaden von Programmanfang zu p berechnet wird.

↳ Totale Redundanz von e an p

Partielle Verfügbarkeit (Partial Availability)

Ein Ausdruck e ist an einer Programmstelle p partiell verfügbar, wenn sein Wert auf **einigen** Pfaden von Programmanfang zu p berechnet wird.

↳ Partielle Redundanz von e an p

OC

A. Koch

Einleitung

PRE

Konzepte

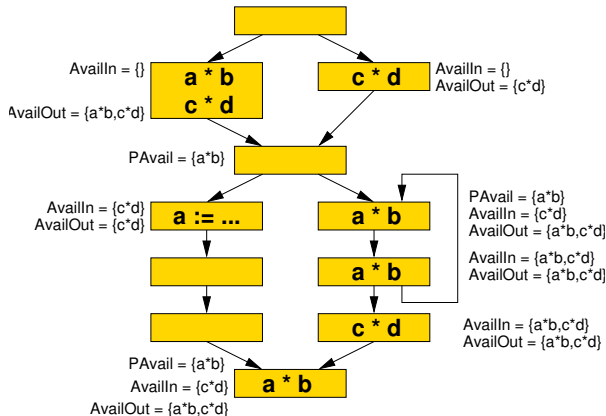
E-Pfad PRE

Notation

E-Pfade

Datenfluss

Beispiel: Verfügbare Ausdrücke



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Nach oben exponierte Ausdrücke (Upwards Exposed)

Ausdrücke, deren Operanden vom Blockanfang bis zu ihrer Stelle nicht überschrieben werden sind nach **oben exponiert**.

➔ Ihre Berechnung könnte an den Blockanfang vorgezogen werden

- → UEEExpr
- Auch genannt **lokal vorziehbar** (locally anticipatable, ANTloc)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Nach oben exponierte Ausdrücke (Upwards Exposed)

Ausdrücke, deren Operanden vom Blockanfang bis zu ihrer Stelle nicht überschrieben werden sind nach **oben exponiert**.

➔ Ihre Berechnung könnte an den Blockanfang vorgezogen werden

- → UEEExpr
- Auch genannt **lokal vorziehbar** (locally anticipatable, ANTloc)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Nach unten exponierte Ausdrücke (Downward Exposed)

Ausdrücke, deren Operanden von ihrer Stelle bis zum Blockende nicht überschrieben werden sind nach **unten exponiert**.

→ Ihre Berechnung könnte an das Blockende verzögert werden

- → DEExpr

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Nach unten exponierte Ausdrücke (Downward Exposed)

Ausdrücke, deren Operanden von ihrer Stelle bis zum Blockende nicht überschrieben werden sind nach **unten exponiert**.

➔ Ihre Berechnung könnte an das Blockende verzögert werden

- → DEExpr

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

a := 42

w := a + b

x := c + d

y := e + f

z := a + e

e := 23

nicht UExpr/ANTloc, DEExpr

UExpr/ANTloc, DEExpr

UExpr/ANTloc, nicht DEExpr

nicht UExpr/ANTloc, nicht DEExpr

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

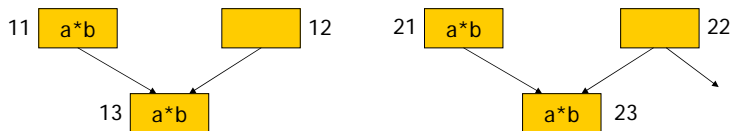
E-Pfade

Datenfluss

Vorziehbarkeit (Anticipatability)

Die Berechnung eines Ausdrucks e ist **vorziehbar** an eine Programmstelle p , wenn er auf allen Pfaden von p zum Programmende berechnet wird.

➔ Auch genannt: Very Busy Expression



- **a*b** vorziehbar nach Block 12
- Aber **nicht** nach Block 22

Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminability paths (E-paths)

OC

A. Koch

Einleitung

PRE

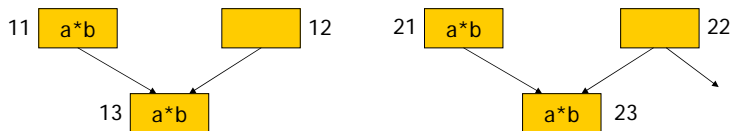
Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss



- $a*b$ vorziehbar nach Block 12
- Aber **nicht** nach Block 22

Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminability paths (E-paths)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Sichere Berechnung

Ein Ausdruck e kann an der Stelle p **sicher** berechnet werden, wenn er dort bereits **verfügbar** ist oder dorthin **vorgezogen** werden kann.

➔ Ziel: Gleicher Wert ohne weitere eventuelle Berechnungsfehler (exceptions, z.B. Division-by-Zero)

- Im 1. Fall wurde der Ausdruck so bereits berechnet und könnte gefahrlos noch ein weiteres Mal berechnet werden
- Im 2. Fall würden eventuelle Fehler ohnehin auftreten, da dieser Ausdruck später in jedem Fall berechnet würde

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Sichere Berechnung

Ein Ausdruck e kann an der Stelle p **sicher** berechnet werden, wenn er dort bereits **verfügbar** ist oder dorthin **vorgezogen** werden kann.

➔ Ziel: Gleicher Wert ohne weitere eventuelle Berechnungsfehler (exceptions, z.B. Division-by-Zero)

- Im 1. Fall wurde der Ausdruck so bereits berechnet und könnte gefahrlos noch ein weiteres Mal berechnet werden
- Im 2. Fall würden eventuelle Fehler ohnehin auftreten, da dieser Ausdruck später in jedem Fall berechnet würde

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Sichere Berechnung

Ein Ausdruck e kann an der Stelle p **sicher** berechnet werden, wenn er dort bereits **verfügbar** ist oder dorthin **vorgezogen** werden kann.

➔ Ziel: Gleicher Wert ohne weitere eventuelle Berechnungsfehler (exceptions, z.B. Division-by-Zero)

- Im 1. Fall wurde der Ausdruck so bereits berechnet und könnte gefahrlos noch ein weiteres Mal berechnet werden
- Im 2. Fall würden eventuelle Fehler ohnehin auftreten, da dieser Ausdruck später in jedem Fall berechnet würde

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Sichere Berechnung

Ein Ausdruck e kann an der Stelle p **sicher** berechnet werden, wenn er dort bereits **verfügbar** ist oder dorthin **vorgezogen** werden kann.

➔ Ziel: Gleicher Wert ohne weitere eventuelle Berechnungsfehler (exceptions, z.B. Division-by-Zero)

- Im 1. Fall wurde der Ausdruck so bereits berechnet und könnte gefahrlos noch ein weiteres Mal berechnet werden
- Im 2. Fall würden eventuelle Fehler ohnehin auftreten, da dieser Ausdruck später in jedem Fall berechnet würde

OC

A. Koch

Einleitung

PRE

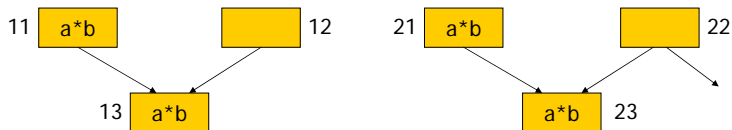
Konzepte

E-Pfad PRE

Notation

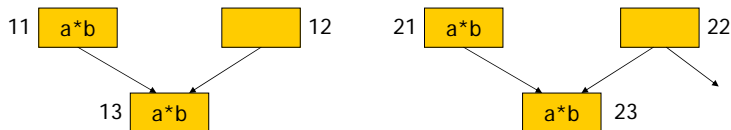
E-Pfade

Datenfluss



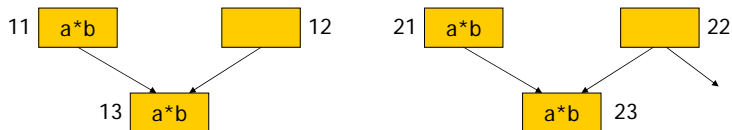
- Neuberechnung von $a*b$ in Block 12 ist sicher
- $a*b$ in Block 22 ist unsicher
- $a*b$ in Kante (Block 22, Block 23) wäre sicher

Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminability paths (E-paths)



- Neuberechnung von $a*b$ in Block 12 ist sicher
- $a*b$ in Block 22 ist unsicher
- $a*b$ in Kante (Block 22, Block 23) wäre sicher

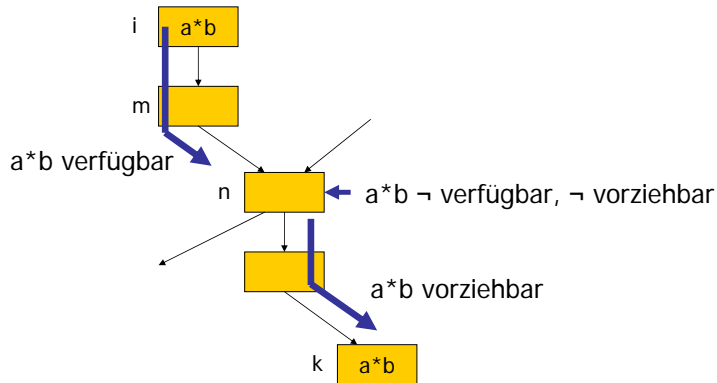
Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminability paths (E-paths)



- Neuberechnung von $a*b$ in Block 12 ist sicher
- $a*b$ in Block 22 ist unsicher
- $a*b$ in **Kante** (Block 22, Block 23) wäre sicher

Quelle: Dhamdhere, Code optimization by partial redundancy elimination using Eliminability paths (E-paths)

Beispiel: Verfügbar, vorziehbar, sicher



OC

A. Koch

Einleitung

PRE

Konzepte

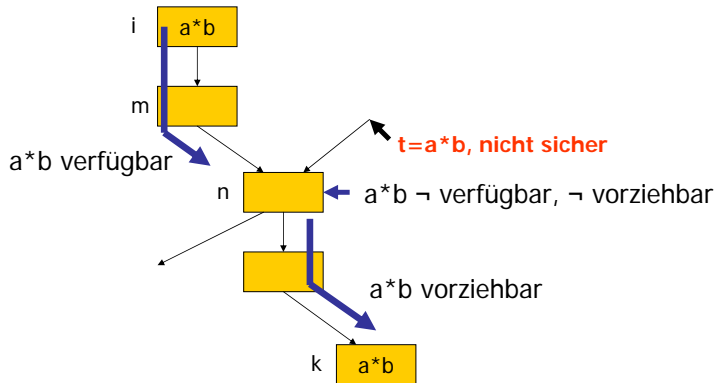
E-Pfad PRE

Notation

E-Pfade

Datenfluss

Beispiel: Verfügbar, vorziehbar, sicher



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Transparenz

Ein Block **B** ist **transparent** in Hinblick auf einen Ausdruck e , wenn er selbst keine Zuweisungen an Operanden von e enthält.

Leere

Ein Block **B** ist **leer** in Hinblick auf einen Ausdruck e , wenn er selbst weder eine Auswertung von e enthält noch Zuweisungen an Operanden von e .

↪ Schreibweise: $\text{empty}(\mathbf{B}) = \text{TRUE}$ bezüglich e

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Transparenz

Ein Block **B** ist **transparent** in Hinblick auf einen Ausdruck e , wenn er selbst keine Zuweisungen an Operanden von e enthält.

Leere

Ein Block **B** ist **leer** in Hinblick auf einen Ausdruck e , wenn er selbst weder eine Auswertung von e enthält noch Zuweisungen an Operanden von e .

→ Schreibweise: $\text{empty}(\mathbf{B}) = \text{TRUE}$ bezüglich e

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

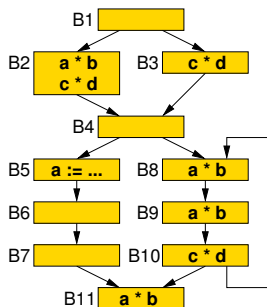
Transparenz

Ein Block **B** ist **transparent** in Hinblick auf einen Ausdruck e , wenn er selbst keine Zuweisungen an Operanden von e enthält.

Leere

Ein Block **B** ist **leer** in Hinblick auf einen Ausdruck e , wenn er selbst weder eine Auswertung von e enthält noch Zuweisungen an Operanden von e .

➔ Schreibweise: $\text{empty}(\mathbf{B}) = \text{TRUE}$ bezüglich e

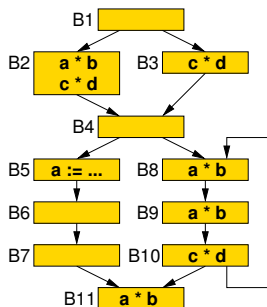


• Für $a*b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
 $\text{empty}(b) = \text{TRUE}$

• Für $c*d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
 $\text{empty}(b) = \text{TRUE}$

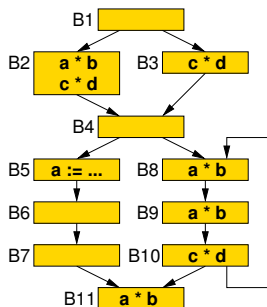


• Für $a*b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
empty(b) = TRUE

• Für $c*d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
empty(b) = TRUE

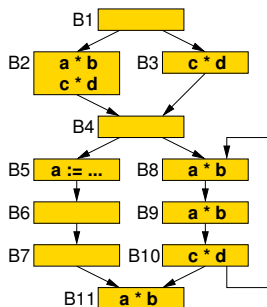


• Für $a * b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
empty(b) = TRUE

• Für $c * d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
empty(b) = TRUE

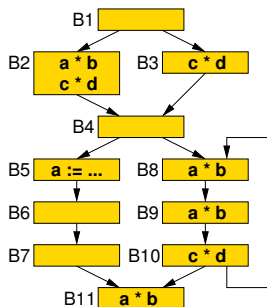


• Für $a*b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
empty(b) = TRUE

• Für $c*d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
empty(b) = TRUE

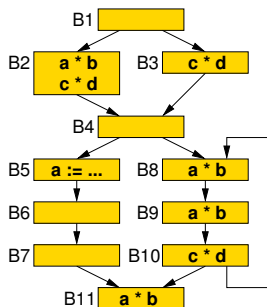


• Für $a * b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
empty(b) = TRUE

• Für $c * d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
empty(b) = TRUE

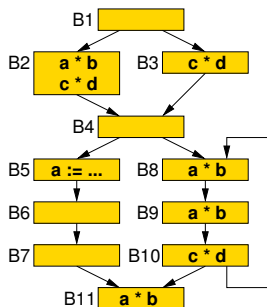


• Für $a * b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
 $\text{empty}(b) = \text{TRUE}$

• Für $c * d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
 $\text{empty}(b) = \text{TRUE}$



• Für $a * b$

- Alle Blöcke ausser **B5** sind transparent
- $b \in \{ \mathbf{B1, B3, B4, B6, B7, B10} \}$:
 $\text{empty}(b) = \text{TRUE}$

• Für $c * d$

- Alle Blöcke sind transparent
- $b \in \{ \mathbf{B1, B4, B5, B6, B7, B8, B9, B11} \}$:
 $\text{empty}(b) = \text{TRUE}$

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge **schnelleren** Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge **schnelleren** Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge schnelleren Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge **schnelleren** Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge schnelleren Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge schnelleren Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge schnelleren Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Bearbeite **alle** partiell redundanten Verwendungen von Ausdrücken e
 - Berechnungsoptimalität
- Füge Neuberechnungen von e an **sicheren** Stellen ein
- Lösche nun total redundant gewordene Berechnungen von e
- Achte auf **kurze Lebenszeiten** von Neuberechneten Werten
- Vermeide unnötiges Aufteilen von Kanten
- Bevorzuge schnelleren Algorithmus
- Am besten auch noch möglichst einfach zu verstehen

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

PRE mit Eliminierungspfaden

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

E-path_PRE – Partial Redundancy Elimination Made Easy
ACM SIGPLAN Notices, 2002, vol. 37, no 8, pp. 53-65

- Dhanajay M. Dhamdhere
- Korrigierte Fassung auf OC Web-Site
- Verfeinert zusammen mit Dheeraj Kumar 2006

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

E-path_PRE – Partial Redundancy Elimination Made Easy
ACM SIGPLAN Notices, 2002, vol. 37, no 8, pp. 53-65

- Dhanajay M. Dhamdhere
- Korrigierte Fassung auf OC Web-Site
- Verfeinert zusammen mit Dheeraj Kumar 2006

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

E-path_PRE – Partial Redundancy Elimination Made Easy
ACM SIGPLAN Notices, 2002, vol. 37, no 8, pp. 53-65

- Dhanajay M. Dhamdhere
- Korrigierte Fassung auf OC Web-Site
- Verfeinert zusammen mit Dheeraj Kumar 2006

Eliminierbarkeitspfad (E-Pfad, eliminatability path)

Ein E-Pfad für einen Ausdruck e ist ein Pfad b_i, \dots, b_k im CFG so dass

- 1 e lokal **verfügbar** in b_i und lokal **vorziehbar** in b_k ist
- 2 Für $b \in (b_i, \dots, b_k)$ gilt: ***empty***(b) = TRUE
- 3 e ist **sicher** auf jeder Ausgangskante eines Blocks $b \in [b_i, \dots, b_k)$

Notation: Ein Pfad $[b_i, \dots, b_k]$ enthält seine Anfangs- und Endblöcke, ein Pfad (b_i, \dots, b_k) nicht.

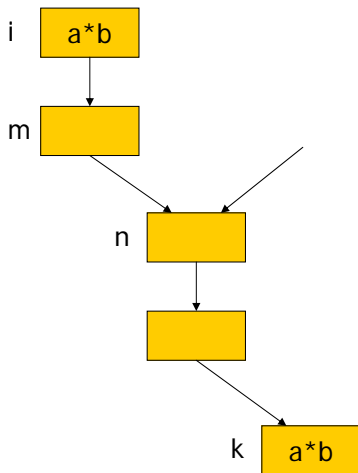
Eliminierbarkeitspfad (E-Pfad, eliminatability path)

Ein E-Pfad für einen Ausdruck e ist ein Pfad b_i, \dots, b_k im CFG so dass

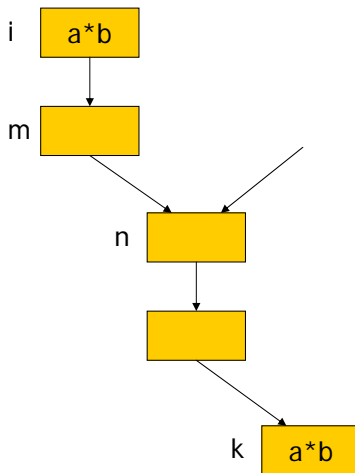
- 1 e lokal **verfügbar** in b_i und lokal **vorziehbar** in b_k ist
- 2 Für $b \in (b_i, \dots, b_k)$ gilt: ***empty***(b) = TRUE
- 3 e ist **sicher** auf jeder Ausgangskante eines Blocks $b \in [b_i, \dots, b_k)$

Notation: Ein Pfad $[b_i, \dots, b_k]$ enthält seine Anfangs- und Endblöcke, ein Pfad (b_i, \dots, b_k) nicht.

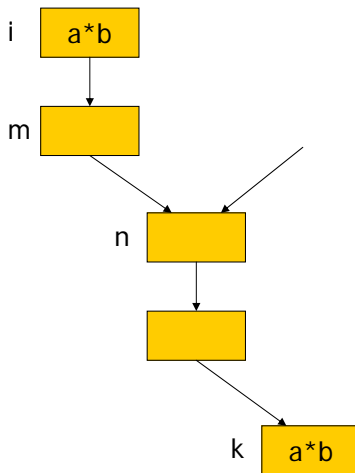
- $a*b$ ist **verfügbar** an Ende von $[i \dots m]$
- $a*b$ ist **vorziehbar** am Ende von $[n \dots k]$
- Berechnung von $a*b$ in Block k ist **eliminierbar**

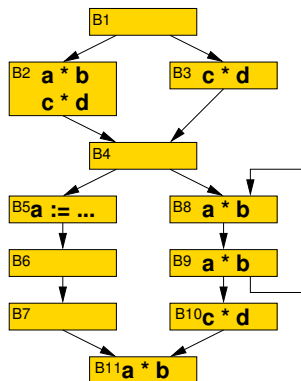


- $a*b$ ist **verfügbar** an Ende von $[i \dots m]$
- $a*b$ ist **vorziehbar** am Ende von $[n \dots k]$
- Berechnung von $a*b$ in Block k ist **eliminierbar**



- $a*b$ ist **verfügbar** an Ende von $[i \dots m]$
- $a*b$ ist **vorziehbar** am Ende von $[n \dots k]$
- Berechnung von $a*b$ in Block k ist **eliminierbar**



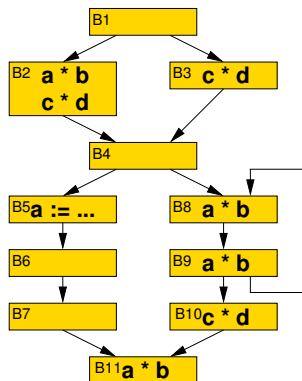


• Für $a * b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- **Nicht:** $[b_2, b_4, b_8]$

• Für $c * d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

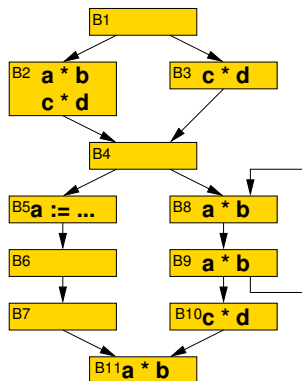


• Für $a*b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- Nicht: $[b_2, b_4, b_8]$

• Für $c*d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

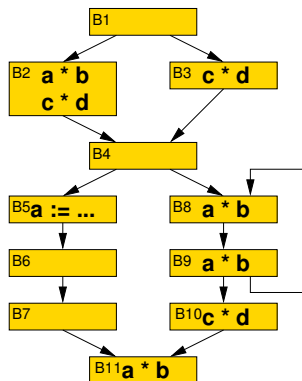


• Für $a*b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- Nicht: $[b_2, b_4, b_8]$

• Für $c*d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

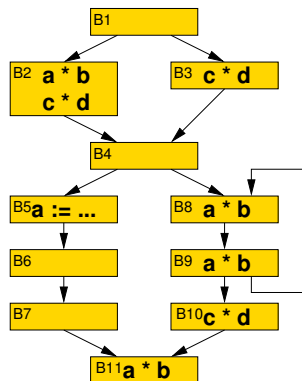


• Für $a*b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- Nicht: $[b_2, b_4, b_8]$

• Für $c*d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

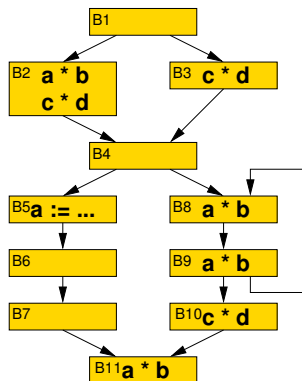


• Für $a*b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- **Nicht:** $[b_2, b_4, b_8]$

• Für $c*d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

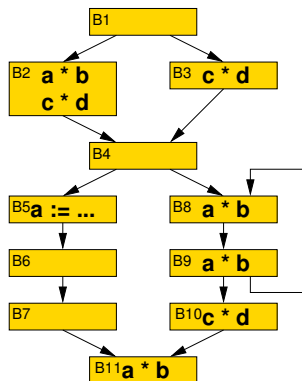


• Für $a * b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- **Nicht:** $[b_2, b_4, b_8]$

• Für $c * d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

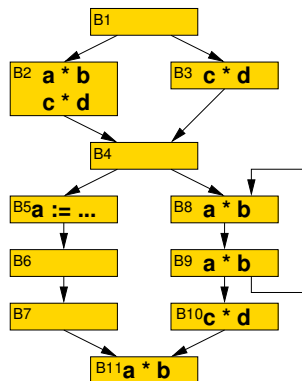


• Für $a * b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- **Nicht:** $[b_2, b_4, b_8]$

• Für $c * d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$



• Für $a*b$

- $[b_8, b_9]$
- $[b_9, b_8]$
- $[b_9, b_{10}, b_{11}]$
- **Nicht:** $[b_2, b_4, b_8]$

• Für $c*d$

- $[b_2, b_4, b_8, b_9, b_{10}]$
- $[b_3, b_4, b_8, b_9, b_{10}]$

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des Blocks b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten Kante (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

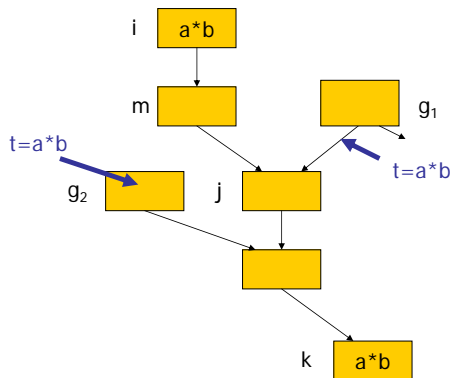
Notation

E-Pfade

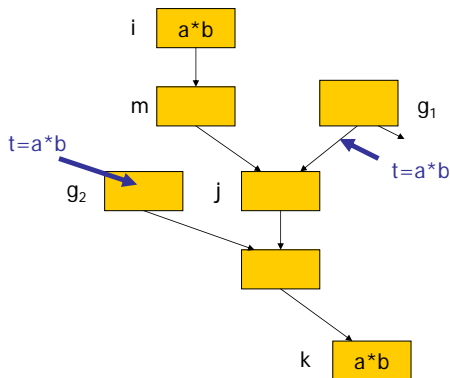
Datenfluss

- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.

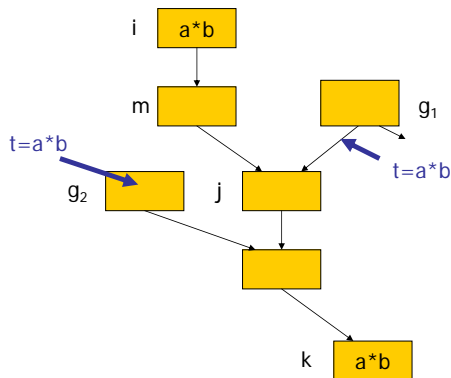
- Das erste Vorkommen von e im Endblock b_k des E-Pfades $[b_i, \dots, b_k]$ ist **eliminierbar**
- Das vorherige Evaluationsergebnis des Ausdrucks e wird dazu **gesichert** in der Variable t_e
- Wird der Pfad (b_i, \dots, b_k) von einem Block **ausserhalb** b_h über eine Kante (b_h, b_j) betreten
- ... muss e (falls nötig) mit $t_e := e$ **berechnet** und **gesichert** werden ...
 - Am Ende des **Blocks** b_h , falls $|\text{succ}(b_h)| = 1$
 - Auf der aufgeteilten **Kante** (b_h, b_j) sonst
 - Lebenszeitoptimale Platzierung (so spät wie möglich)
- Eine Neuberechnung und Sicherung ist nicht nötig, falls b_h **selber** auf einem E-Pfad für e liegt.



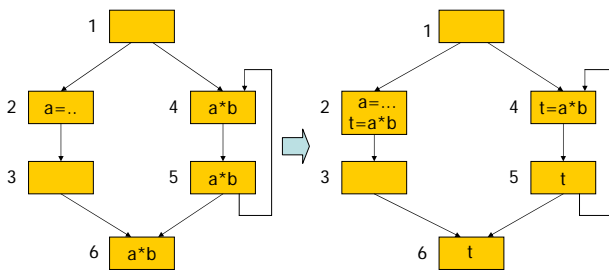
- E-Pfad: $[i, \dots, k]$
- Einfügen von Berechnungen von e und Kopieren
 - In Kante (g_1, j) und Block g_2



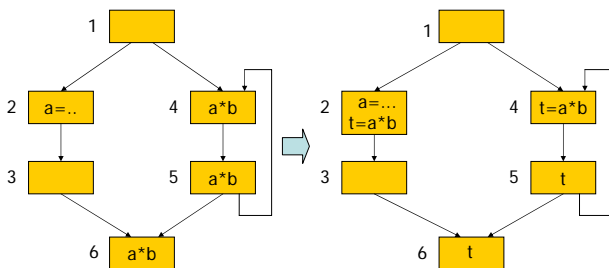
- E-Pfad: $[i, \dots, k]$
- Einfügen von Berechnungen von e und Kopieren
 - In Kante (g_1, j) und Block g_2



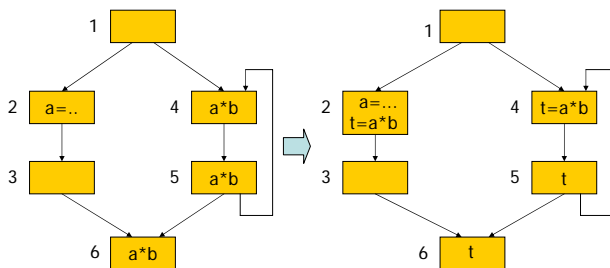
- E-Pfad: $[i, \dots, k]$
- Einfügen von Berechnungen von e und Kopieren
 - In Kante (g_1, j) und Block g_2



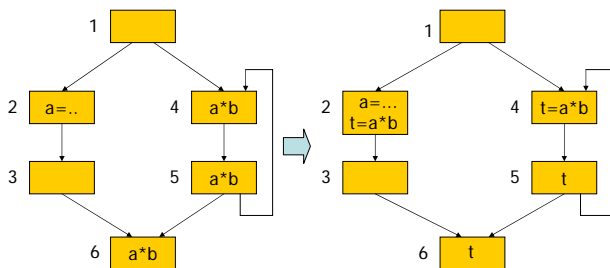
- E-Pfade: $[4,5]$, $[5,4]$, $[5,6]$
- Herstellen totaler Redundanz: Einfügen in Ast 2,3
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5,4]$ ist E-Pfad, Berechnung in 4 muss entfernbar sein
→ Kante $(1,4)$ aufteilen, dort Berechnen und Kopieren



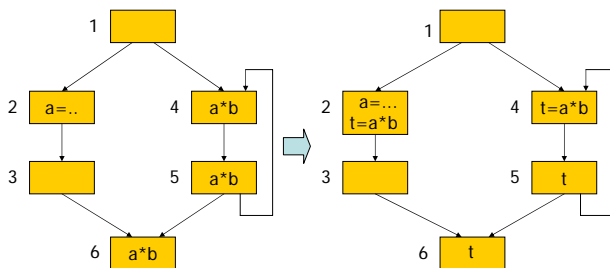
- E-Pfade: $[4,5]$, $[5,4]$, $[5,6]$
- **Herstellen totaler Redundanz: Einfügen in Ast 2,3**
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5,4]$ ist E-Pfad, Berechnung in 4 muss entfernbar sein
→ Kante $(1,4)$ aufteilen, dort Berechnen und Kopieren



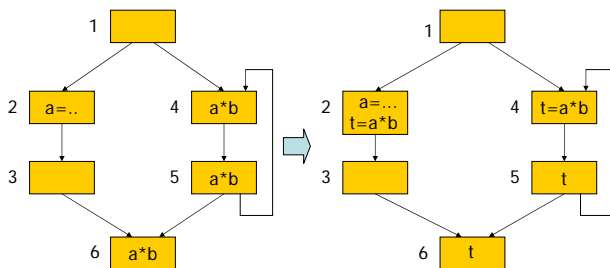
- E-Pfade: $[4,5]$, $[5,4]$, $[5,6]$
- Herstellen totaler Redundanz: Einfügen in Ast 2,3
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5,4]$ ist E-Pfad, Berechnung in 4 muss entfernbar sein
→ Kante $(1,4)$ aufteilen, dort Berechnen und Kopieren



- E-Pfade: $[4, 5]$, $[5, 4]$, $[5, 6]$
- Herstellen totaler Redundanz: Einfügen in Ast 2,3
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5, 4]$ ist E-Pfad, Berechnung in 4 muss entfernbare sein
→ Kante $(1,4)$ aufteilen, dort Berechnen und Kopieren



- E-Pfade: $[4, 5]$, $[5, 4]$, $[5, 6]$
- Herstellen totaler Redundanz: Einfügen in Ast 2,3
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5, 4]$ ist E-Pfad, Berechnung in 4 muss entfernbare sein
→ Kante $(1,4)$ aufteilen, dort Berechnen und Kopieren



- E-Pfade: $[4, 5]$, $[5, 4]$, $[5, 6]$
- Herstellen totaler Redundanz: Einfügen in Ast 2,3
 - Nach MRA in 2, mit E-Pfaden in 3: Lebenszeitoptimal
- Entfernen von Redundanz in 4
 - Nach MRA nicht möglich
 - $[5, 4]$ ist E-Pfad, Berechnung in 4 muss entfernbare sein
→ Kante (1,4) aufteilen, dort Berechnen und Kopieren

Wert von e sichern

Füge Anweisung $t_e := e$ vor einer Benutzung von e ein und ersetze Benutzung durch t_e

Neue Berechnung von e einfügen

Füge Anweisung $t_e := e$ ein

Eliminiere redundante Berechnung von e

Ersetze e durch t_e

An welchen Stellen diese Aktionen ausführen?

Wert von e sichern

Füge Anweisung $t_e := e$ vor einer Benutzung von e ein und ersetze Benutzung durch t_e

Neue Berechnung von e einfügen

Füge Anweisung $t_e := e$ ein

Eliminiere redundante Berechnung von e

Ersetze e durch t_e

An welchen Stellen diese Aktionen ausführen?

Wert von e sichern

Füge Anweisung $t_e := e$ vor einer Benutzung von e ein und ersetze Benutzung durch t_e

Neue Berechnung von e einfügen

Füge Anweisung $t_e := e$ ein

Eliminiere redundante Berechnung von e

Ersetze e durch t_e

An welchen Stellen diese Aktionen ausführen?

Wert von e sichern

Füge Anweisung $t_e := e$ vor einer Benutzung von e ein und ersetze Benutzung durch t_e

Neue Berechnung von e einfügen

Füge Anweisung $t_e := e$ ein

Eliminiere redundante Berechnung von e

Ersetze e durch t_e

An welchen Stellen diese Aktionen ausführen?

Wert von e sichern

Füge Anweisung $t_e := e$ vor einer Benutzung von e ein und ersetze Benutzung durch t_e

Neue Berechnung von e einfügen

Füge Anweisung $t_e := e$ ein

Eliminiere redundante Berechnung von e

Ersetze e durch t_e

An welchen Stellen diese Aktionen ausführen?

Neue Notation

Prädikate bestimmen für jeden Ausdruck e , ob eine Aussage wahr oder falsch ist

Wahr für alle am Ende eines Blocks b verfügbaren Ausdr.

$$\text{avail}(b) = \text{TRUE für } e \Leftrightarrow e \in \text{AVAILOUT}(b)$$

Wahr für alle in den Block b vorziehbaren Ausdrücke

$$\text{ant}(b) = \text{TRUE für } e \Leftrightarrow e \in \text{ANT}(b)$$

Wahr für alle Ausdrücke, für die beides gilt

$$\text{avail}(b) \cdot \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cap \text{ANT}(b))$$

Wahr für alle Ausdrücke, für die mindestens eines gilt

$$\text{avail}(b) + \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cup \text{ANT}(b))$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Prädikate bestimmen für jeden Ausdruck e , ob eine Aussage wahr oder falsch ist

Wahr für alle am Ende eines Blocks b verfügbaren Ausdr.

$\text{avail}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{AVAILOUT}(b)$

Wahr für alle in den Block b vorziehbaren Ausdrücke

$\text{ant}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{ANT}(b)$

Wahr für alle Ausdrücke, für die beides gilt

$\text{avail}(b) \cdot \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cap \text{ANT}(b))$

Wahr für alle Ausdrücke, für die mindestens eines gilt

$\text{avail}(b) + \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cup \text{ANT}(b))$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Prädikate bestimmen für jeden Ausdruck e , ob eine Aussage wahr oder falsch ist

Wahr für alle am Ende eines Blocks b verfügbaren Ausdr.

$\text{avail}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{AVAILOUT}(b)$

Wahr für alle in den Block b vorziehbaren Ausdrücke

$\text{ant}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{ANT}(b)$

Wahr für alle Ausdrücke, für die beides gilt

$\text{avail}(b) \cdot \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cap \text{ANT}(b))$

Wahr für alle Ausdrücke, für die mindestens eines gilt

$\text{avail}(b) + \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cup \text{ANT}(b))$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Prädikate bestimmen für jeden Ausdruck e , ob eine Aussage wahr oder falsch ist

Wahr für alle am Ende eines Blocks b verfügbaren Ausdr.

$\text{avail}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{AVAILOUT}(b)$

Wahr für alle in den Block b vorziehbaren Ausdrücke

$\text{ant}(b) = \text{TRUE}$ für $e \Leftrightarrow e \in \text{ANT}(b)$

Wahr für alle Ausdrücke, für die beides gilt

$\text{avail}(b) \cdot \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cap \text{ANT}(b))$

Wahr für alle Ausdrücke, für die mindestens eines gilt

$\text{avail}(b) + \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cup \text{ANT}(b))$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Prädikate bestimmen für jeden Ausdruck e , ob eine Aussage wahr oder falsch ist

Wahr für alle am Ende eines Blocks b verfügbaren Ausdr.

$$\text{avail}(b) = \text{TRUE für } e \Leftrightarrow e \in \text{AVAILOUT}(b)$$

Wahr für alle in den Block b vorziehbaren Ausdrücke

$$\text{ant}(b) = \text{TRUE für } e \Leftrightarrow e \in \text{ANT}(b)$$

Wahr für alle Ausdrücke, für die beides gilt

$$\text{avail}(b) \cdot \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cap \text{ANT}(b))$$

Wahr für alle Ausdrücke, für die mindestens eines gilt

$$\text{avail}(b) + \text{ant}(b) = \text{TRUE} \Leftrightarrow e \in (\text{AVAILOUT}(b) \cup \text{ANT}(b))$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Konjunktion

$cpin(b) = \prod_p cpout(p) = \text{TRUE}$ für Kopie c
 $\Leftrightarrow c \in \bigcap_{p \in pred(b)} CPOUT(p)$

Disjunktion

$liveout(b) = \sum_s livein(s) = \text{TRUE}$ für Variable v
 $\Leftrightarrow v \in \bigcup_{s \in succ(b)} LIVEIN(s)$

Konjunktion

$cpin(b) = \prod_p cpout(p) = \text{TRUE}$ für Kopie c
 $\Leftrightarrow c \in \bigcap_{p \in pred(b)} CPOUT(p)$

Disjunktion

$liveout(b) = \sum_s livein(s) = \text{TRUE}$ für Variable v
 $\Leftrightarrow v \in \bigcup_{s \in succ(b)} LIVEIN(s)$

Für alle Ausdrücke eindeutige Zuordnung festlegen

$a*b \rightarrow$ Bit 1

$c*d \rightarrow$ Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Für alle Ausdrücke eindeutige Zuordnung festlegen

a*b → Bit 1

c*d → Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Für alle Ausdrücke eindeutige Zuordnung festlegen

a*b → Bit 1

c*d → Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Für alle Ausdrücke eindeutige Zuordnung festlegen

a*b → Bit 1

c*d → Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

Für alle Ausdrücke eindeutige Zuordnung festlegen

a*b → Bit 1

c*d → Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

Für alle Ausdrücke eindeutige Zuordnung festlegen

a*b → Bit 1

c*d → Bit 2

...

Wert	entspricht Prädikat x	entspricht Menge
00	$\neg x_{a*b}, \neg x_{c*d}$	\emptyset
01	$\neg x_{a*b}, x_{c*d}$	$\{c * d\}$
10	$x_{a*b}, \neg x_{c*d}$	$\{a * b\}$
11	x_{a*b}, x_{c*d}	$\{a * b, c * d\}$

- Bitvektoren können sehr lang werden
- Prädikate liefern Ergebnisse als Bitvektor
- Damit logische Verknüpfungen effizient implementierbar

Zuordnung von Ausdrücken e an Bits

$a \cdot b$ \rightarrow Bit 1

$c \cdot d$ \rightarrow Bit 2

$a \cdot d$ \rightarrow Bit 3

...

Gegeben: Zwei Prädikate x und y bezüglich Ausdruck e

Annahme: $x = 101, y = 011$

$$x \cdot y = x \& y = 001$$

$$x + y = x | y = 111$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Zuordnung von Ausdrücken e an Bits

$a \cdot b$ \rightarrow Bit 1

$c \cdot d$ \rightarrow Bit 2

$a \cdot d$ \rightarrow Bit 3

...

Gegeben: Zwei Prädikate x und y bezüglich Ausdruck e

Annahme: $x = 101, y = 011$

$$x \cdot y = x \& y = 001$$

$$x + y = x | y = 111$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Zuordnung von Ausdrücken e an Bits

$a \cdot b$ \rightarrow Bit 1

$c \cdot d$ \rightarrow Bit 2

$a \cdot d$ \rightarrow Bit 3

...

Gegeben: Zwei Prädikate x und y bezüglich Ausdruck e

Annahme: $x = 101, y = 011$

$$x \cdot y = x \& y = 001$$

$$x + y = x | y = 111$$

OC

A. Koch

Einleitung

PRE

Konzepte

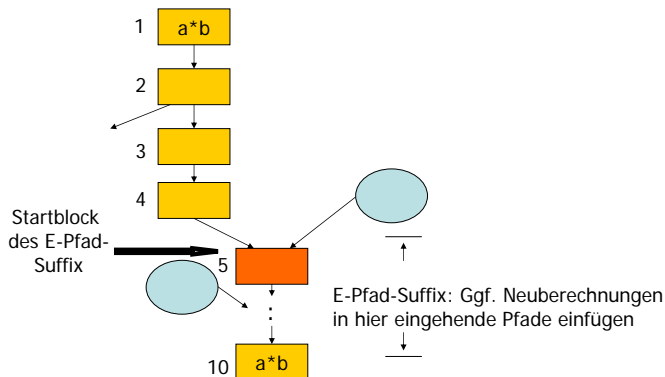
E-Pfad PRE

Notation

E-Pfade

Datenfluss

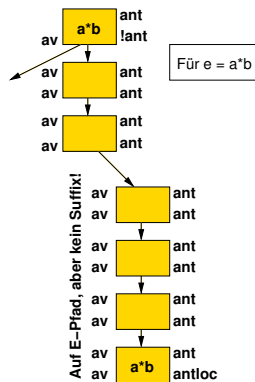
Bestimmen von E-Pfaden



$$\begin{aligned} b \in (1,2) & \quad \text{avail}(b) \cdot \neg \text{ant}(b) \\ b \in (3,4) & \quad \text{avail}(b) \cdot \text{ant}(b) \\ b \in (5, \dots, 10) & \quad \neg \text{avail}(b) \cdot \text{ant}(b) \end{aligned}$$

E-Pfad-Suffix

Anforderung an E-Pfad-Suffix: $\neg \text{avail}(b) \cdot \text{ant}(b)$



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg \text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$bis **Start des E-Pfades** b_i erreicht
- Suche vorwärts von b_m durch Segment
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$bis **Ende des E-Pfades** erreicht

- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg\text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg\text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg\text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente $\text{avail}(b) \cdot \text{ant}(b)$
 $\text{avail}(b) \cdot \neg\text{ant}(b)$
bis **Start des E-Pfades** b_i erreicht
- Suche vorwärts von b_m durch Segment $\neg\text{avail}(b) \cdot \text{ant}(b)$
bis **Ende des E-Pfades** erreicht

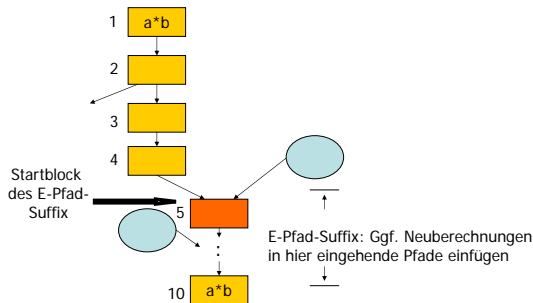
- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg\text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg\text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg\text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente $\text{avail}(b) \cdot \text{ant}(b)$
 $\text{avail}(b) \cdot \neg\text{ant}(b)$
bis **Start des E-Pfades** b_i erreicht
- Suche vorwärts von b_m durch Segment $\neg\text{avail}(b) \cdot \text{ant}(b)$
bis **Ende des E-Pfades** erreicht

- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg\text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg\text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg\text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente $\text{avail}(b) \cdot \text{ant}(b)$
 $\text{avail}(b) \cdot \neg\text{ant}(b)$
bis **Start des E-Pfades** b_i erreicht
- Suche vorwärts von b_m durch Segment $\neg\text{avail}(b) \cdot \text{ant}(b)$
bis **Ende des E-Pfades** erreicht

- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg \text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$bis **Start des E-Pfads** b_i erreicht
- Suche vorwärts von b_m durch Segment
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$bis **Ende des E-Pfades** erreicht

- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg \text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$bis **Start des E-Pfades** b_i erreicht
- Suche vorwärts von b_m durch Segment
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$bis **Ende des E-Pfades** erreicht

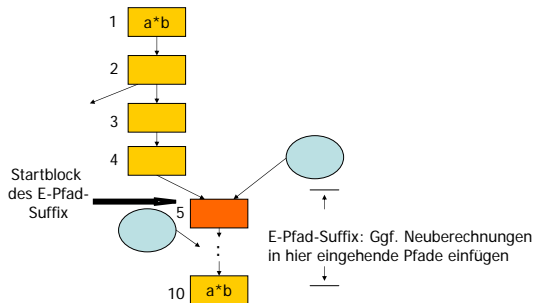
- E-Pfad $[b_i, \dots, b_k]$ hat drei (ggf. leere) Segmente
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$
- Finde **Start des E-Pfad-Suffixes**: Suche Block b_m mit $\neg \text{avail}(b_m) \cdot \text{ant}(b_m) \cdot \sum_p \text{avail}(p)$
- Suche rückwärts von b_m durch Segmente
 - $\text{avail}(b) \cdot \text{ant}(b)$
 - $\text{avail}(b) \cdot \neg \text{ant}(b)$bis **Start des E-Pfads** b_i erreicht
- Suche vorwärts von b_m durch Segment
 - $\neg \text{avail}(b) \cdot \text{ant}(b)$bis **Ende des E-Pfades** erreicht



Start des E-Pfades Sichere Wert von e in t_e

E-Pfad-Suffix Füge Neuberechnungen $t_e := e$ in eingehende Pfade ein

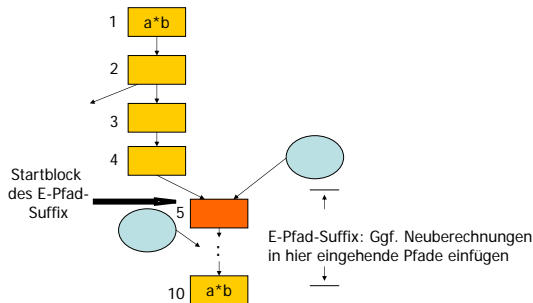
Ende des E-Pfades Ersetze redundante Berechnung von e durch t_e



Start des E-Pfades Sichere Wert von e in t_e

E-Pfad-Suffix Füge Neuberechnungen $t_e := e$ in eingehende Pfade ein

Ende des E-Pfades Ersetze redundante Berechnung von e durch t_e



Start des E-Pfades Sichere Wert von e in t_e

E-Pfad-Suffix Füge Neuberechnungen $t_e := e$ in eingehende Pfade ein

Ende des E-Pfades Ersetze redundante Berechnung von e durch t_e

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Löse E-Pfad-Problem für alle Ausdrücke **gleichzeitig**
- Erinnerung: Ergebnisse der Prädikate sind Bitvektoren
- Bitweise Verknüpfung mit AND und OR
- Berechnet nicht nur E-Pfade
- Direkte Bestimmung von
 - Blöcken mit zu eliminierenden Ausdrücken
 - Einfügestellen (Blöcke, Kanten) für Neuberechnungen
 - Stellen für Sichern von Werten

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Datenflussproblem

- Datenflussproblem über Prädikate von **Ausdrücken**
- Zerlegung in
 - Verschiedene Teilprobleme
 - Vorberechenbare Eigenschaften
 - Herleitbare Eigenschaften (durch DF-Löser)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Datenflussproblem über Prädikate von **Ausdrücken**
- Zerlegung in
 - Verschiedene Teilprobleme
 - Vorberechenbare Eigenschaften
 - Herleitbare Eigenschaften (durch DF-Löser)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Datenflussproblem über Prädikate von **Ausdrücken**
- Zerlegung in
 - Verschiedene Teilprobleme
 - Vorberechenbare Eigenschaften
 - Herleitbare Eigenschaften (durch DF-Löser)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Datenflussproblem über Prädikate von **Ausdrücken**
- Zerlegung in
 - Verschiedene Teilprobleme
 - Vorberechenbare Eigenschaften
 - Herleitbare Eigenschaften (durch DF-Löser)

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- Datenflussproblem über Prädikate von **Ausdrücken**
- Zerlegung in
 - Verschiedene Teilprobleme
 - Vorberechenbare Eigenschaften
 - Herleitbare Eigenschaften (durch DF-Löser)

Ausdruck ist lokal verfügbar (locally available)

$\text{comp}(b) = \text{TRUE}$: e wird in b berechnet (computed) und seine Operanden hinterher nicht überschrieben

$\Rightarrow e \in \text{DEEXPR}(b)$

Ausdruck ist lokal vorziehbar (locally anticipatable)

$\text{antloc}(b) = \text{TRUE}$: Operanden von e werden vor Berechnung nicht zugewiesen

$\Rightarrow e \in \text{UEEXPR}(b)$

Block ist transparent für Ausdruck

$\text{transp}(b) = \text{TRUE}$: b hat keine Zuweisungen an Operanden von e

$\Rightarrow e \notin \text{EXPRKILL}(b)$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Ausdruck ist lokal verfügbar (locally available)

$\text{comp}(b) = \text{TRUE}$: e wird in b berechnet (computed) und seine Operanden hinterher nicht überschrieben

↳ $e \in \text{DEEXPR}(b)$

Ausdruck ist lokal vorziehbar (locally anticipatable)

$\text{antloc}(b) = \text{TRUE}$: Operanden von e werden vor Berechnung nicht zugewiesen

↳ $e \in \text{UEEXPR}(b)$

Block ist transparent für Ausdruck

$\text{transp}(b) = \text{TRUE}$: b hat keine Zuweisungen an Operanden von e

↳ $e \notin \text{EXPRKILL}(b)$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Ausdruck ist lokal verfügbar (locally available)

$\text{comp}(b) = \text{TRUE}$: e wird in b berechnet (computed) und seine Operanden hinterher nicht überschrieben

↳ $e \in \text{DEEXPR}(b)$

Ausdruck ist lokal vorziehbar (locally anticipatable)

$\text{antloc}(b) = \text{TRUE}$: Operanden von e werden vor Berechnung nicht zugewiesen

↳ $e \in \text{UEEXPR}(b)$

Block ist transparent für Ausdruck

$\text{transp}(b) = \text{TRUE}$: b hat keine Zuweisungen an Operanden von e

↳ $e \notin \text{EXPRKILL}(b)$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Bit 1: a+b
Bit 2: c+d
Bit 3: e+f
Bit 4: a+e
Bit 5: q+r
Bit 6: x+y

comp(B1) = 111100
antloc(B1) = 011000
transp(B1) = 010010

B1

a := 42

w := a + b

x := c + d

y := e + f

z := a + e

e := 23

nicht UEEExpr/ANTloc, DEEExpr

UEEExpr/ANTloc, DEEExpr

UEEExpr/ANTloc, nicht DEEExpr

nicht UEEExpr/ANTloc, nicht DEEExpr

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

$availin(b)$ e ist verfügbar bei Eintritt in b
 $availout(b)$ e ist verfügbar bei Austritt aus b

Berechnung

$$\begin{aligned}availin(b) &= \prod_p availout(p) \\availout(b) &= availin(b) \cdot transp(b) + comp(b)\end{aligned}$$

$\text{availin}(b)$ e ist verfügbar bei Eintritt in b
 $\text{availout}(b)$ e ist verfügbar bei Austritt aus b

Berechnung

$$\begin{aligned}\text{availin}(b) &= \prod_p \text{availout}(p) \\ \text{availout}(b) &= \text{availin}(b) \cdot \text{transp}(b) + \text{comp}(b)\end{aligned}$$

$\text{availin}(b)$ e ist verfügbar bei Eintritt in b
 $\text{availout}(b)$ e ist verfügbar bei Austritt aus b

Berechnung

$$\begin{aligned}\text{availin}(b) &= \prod_p \text{availout}(p) \\ \text{availout}(b) &= \text{availin}(b) \cdot \text{transp}(b) + \text{comp}(b)\end{aligned}$$

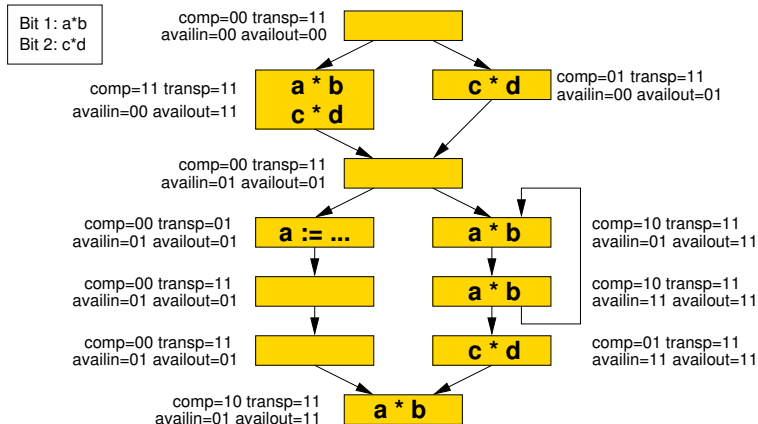
$availin(b)$ e ist verfügbar bei Eintritt in b
 $availout(b)$ e ist verfügbar bei Austritt aus b

Berechnung

$$\begin{aligned} availin(b) &= \prod_p availout(p) \\ availout(b) &= availin(b) \cdot transp(b) + comp(b) \end{aligned}$$

Beispiel verfügbare Ausdrücke

In Prädikatenotation



$$availin(b) = \prod_p availout(p)$$
$$availout(b) = availin(b) \cdot transp(b) + comp(b)$$

$\text{antin}(b)$ e ist vorziehbar bei Eintritt in b

$\text{antout}(b)$ e ist vorziehbar bei Austritt aus b

Berechnung

$$\text{antin}(b) = \text{antout}(b) \cdot \text{transp}(b) + \text{antloc}(b)$$

$$\text{antout}(b) = \prod_s \text{antin}(s)$$

$\text{antin}(b)$ e ist vorziehbar bei Eintritt in b
 $\text{antout}(b)$ e ist vorziehbar bei Austritt aus b

Berechnung

$$\begin{aligned}\text{antin}(b) &= \text{antout}(b) \cdot \text{transp}(b) + \text{antloc}(b) \\ \text{antout}(b) &= \prod_s \text{antin}(s)\end{aligned}$$

$\text{antin}(b)$ e ist vorziehbar bei Eintritt in b
 $\text{antout}(b)$ e ist vorziehbar bei Austritt aus b

Berechnung

$$\begin{aligned}\text{antin}(b) &= \text{antout}(b) \cdot \text{transp}(b) + \text{antloc}(b) \\ \text{antout}(b) &= \prod_s \text{antin}(s)\end{aligned}$$

$\text{antin}(b)$ e ist vorziehbar bei Eintritt in b
 $\text{antout}(b)$ e ist vorziehbar bei Austritt aus b

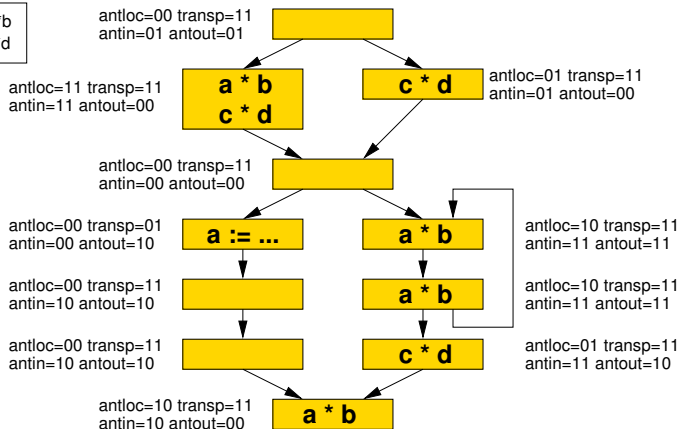
Berechnung

$$\begin{aligned}\text{antin}(b) &= \text{antout}(b) \cdot \text{transp}(b) + \text{antloc}(b) \\ \text{antout}(b) &= \prod_s \text{antin}(s)\end{aligned}$$

Beispiel vorziehbare Ausdrücke

In Prädikatennotation

Bit 1: $a * b$
Bit 2: $c * d$

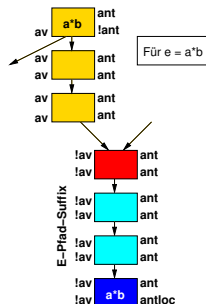


$$antin(b) = antout(b) \cdot transp(b) + antloc(b)$$

$$antout(b) = \prod_s antin(s)$$

$\text{epsin}(b)$ Blockanfang von b
liegt auf einem
E-Pfad-Suffix für e

$\text{epsout}(b)$ Blockende von b liegt
auf einem
E-Pfad-Suffix für e

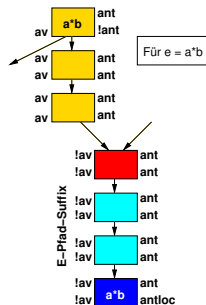


Berechnung

$$\begin{aligned} \text{epsin}(b) &= (\sum_p (\text{availout}(p) + \text{epsout}(p))) \cdot \text{ant}_{in}(b) \cdot \neg \text{avail}_{in}(b) \\ \text{epsout}(b) &= \text{epsin}(b) \cdot \neg \text{antloc}(b) \end{aligned}$$

$\text{epsin}(b)$ Blockanfang von b
liegt auf einem
E-Pfad-Suffix für e

$\text{epsout}(b)$ Blockende von b liegt
auf einem
E-Pfad-Suffix für e

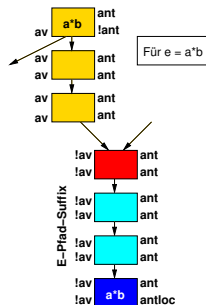


Berechnung

$$\begin{aligned} \text{epsin}(b) &= (\sum_p (\text{availout}(p) + \text{epsout}(p))) \cdot \text{ant}_{in}(b) \cdot \neg \text{avail}_{in}(b) \\ \text{epsout}(b) &= \text{epsin}(b) \cdot \neg \text{antloc}(b) \end{aligned}$$

$\text{epsin}(b)$ Blockanfang von b
liegt auf einem
E-Pfad-Suffix für e

$\text{epsout}(b)$ Blockende von b liegt
auf einem
E-Pfad-Suffix für e

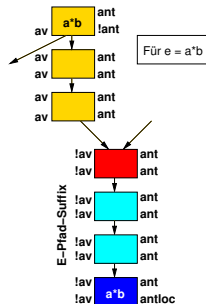


Berechnung

$$\begin{aligned} \text{epsin}(b) &= (\sum_p (\text{availout}(p) + \text{epsout}(p))) \cdot \text{ant}_{in}(b) \cdot \neg \text{avail}_{in}(b) \\ \text{epsout}(b) &= \text{epsin}(b) \cdot \neg \text{antloc}(b) \end{aligned}$$

$\text{epsin}(b)$ Blockanfang von b
liegt auf einem
E-Pfad-Suffix für e

$\text{epsout}(b)$ Blockende von b liegt
auf einem
E-Pfad-Suffix für e

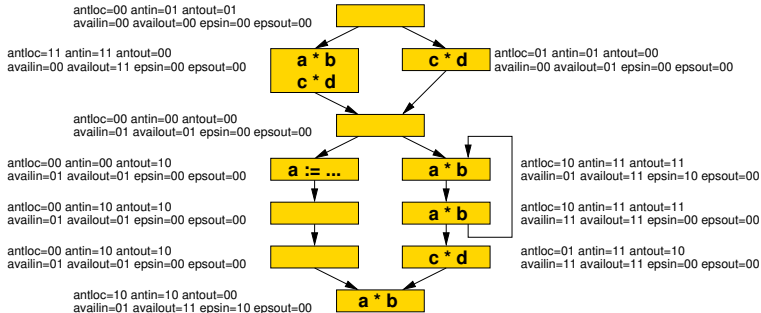


Berechnung

$$\begin{aligned} \text{epsin}(b) &= (\sum_p (\text{availout}(p) + \text{epsout}(p))) \cdot \text{ant}_{in}(b) \cdot \neg \text{avail}_{in}(b) \\ \text{epsout}(b) &= \text{epsin}(b) \cdot \neg \text{ant}_{loc}(b) \end{aligned}$$

Beispiel E-Pfad-Suffix

In Prädikatennotation



$$\text{epsin}(b) = (\sum_p (\text{availout}(p) + \text{epsout}(p))) \cdot \text{antin}(b) \cdot \neg \text{availin}(b)$$

$$\text{epsout}(b) = \text{epsin}(b) \cdot \neg \text{antloc}(b)$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Idee: Bestimmen der Blöcke der redundanten Berechnungen

Redundante $e \in \text{UEExpr}(b_k)$ liegen im **Endblock** b_k eines E-Pfades

E-Pfad-Suffix $\neq \emptyset$

Endblock des

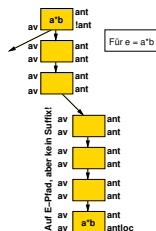
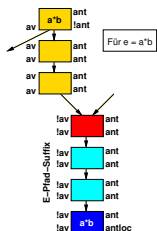
E-Pfad-Suffixes

$\Leftrightarrow \text{epsin}(b) \cdot \text{antloc}(b)$

E-Pfad-Suffix $= \emptyset$

Endblock des E-Pfades

$\Leftrightarrow \text{availin}(b) \cdot \text{antloc}(b)$



Idee: Bestimmen der Blöcke der redundanten Berechnungen

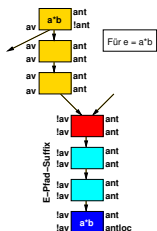
Redundante $e \in \text{UEExpr}(b_k)$ liegen im **Endblock** b_k eines E-Pfades

E-Pfad-Suffix $\neq \emptyset$

Endblock des

E-Pfad-Suffixes

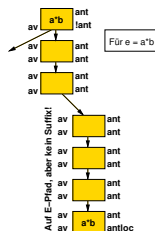
$\Leftrightarrow \text{epsin}(b) \cdot \text{antloc}(b)$



E-Pfad-Suffix $= \emptyset$

Endblock des **E-Pfades**

$\Leftrightarrow \text{availin}(b) \cdot \text{antloc}(b)$



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

$\text{redund}(b)$ Berechnung von e in b ist redundant und kann durch τ_e ersetzt werden

Berechnung

$$\text{redund}(b) = (\text{epsin}(b) + \text{availin}(b)) \cdot \text{antloc}(b)$$

Genauer: Alle lokal antizipierbaren e , also $e \in \text{UEExpr}(b_k)$ sind redundant.

→ Können **eliminiert** werden

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

$\text{redund}(b)$ Berechnung von e in b ist redundant und kann durch τ_e ersetzt werden

Berechnung

$$\text{redund}(b) = (\text{epsin}(b) + \text{availin}(b)) \cdot \text{antloc}(b)$$

Genauer: Alle lokal antizipierbaren e , also $e \in \text{UEExpr}(b_k)$ sind redundant.

→ Können **eliminiert** werden

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

$\text{redund}(b)$ Berechnung von e in b ist redundant und kann durch τ_e ersetzt werden

Berechnung

$$\text{redund}(b) = (\text{epsin}(b) + \text{availin}(b)) \cdot \text{antloc}(b)$$

Genauer: Alle lokal antizipierbaren e , also $e \in \text{UEExpr}(b_k)$ sind redundant.

→ Können **eliminiert** werden

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

$\text{redund}(b)$ Berechnung von e in b ist redundant und kann durch τ_e ersetzt werden

Berechnung

$$\text{redund}(b) = (\text{epsin}(b) + \text{availin}(b)) \cdot \text{antloc}(b)$$

Genauer: Alle lokal antizipierbaren e , also $e \in \text{UEExpr}(b_k)$ sind redundant.

→ Können **eliminiert** werden

OC

A. Koch

Einleitung

PRE

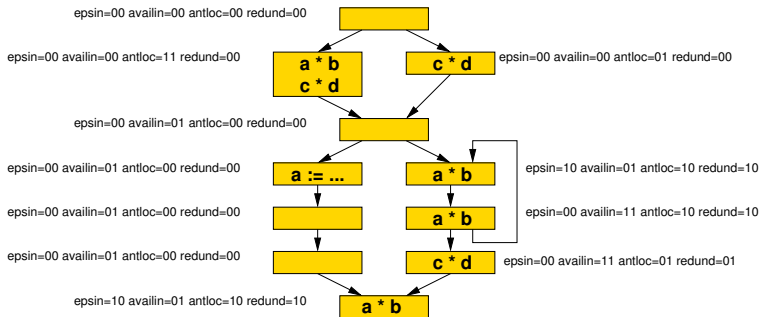
Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss



$$\text{redund}(b) = (\text{epsin}(b) + \text{availin}(b)) \cdot \text{antloc}(b)$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

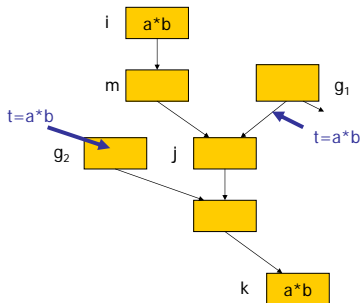
E-Pfade

Datenfluss

Einfügestellen für neue Berechnungen

Wo muß $t_e := e$ eingefügt werden?

An Eintrittspunkten in den Pfad!

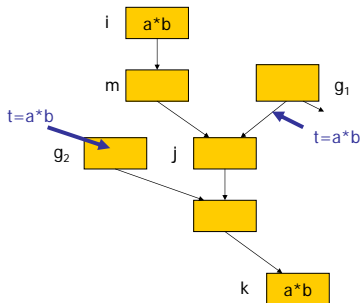


- Wenn alle Nachfolger von externem Block in E-Pfad:
In Block
- Sonst in aufgeteilte Kante

Einfügestellen für neue Berechnungen

Wo muß $t_e := e$ eingefügt werden?

An Eintrittspunkten in den Pfad!

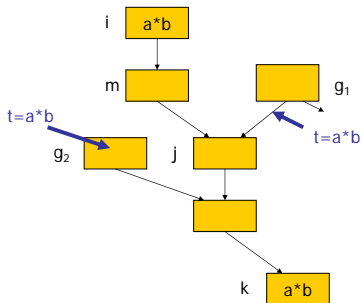


- Wenn alle Nachfolger von externem Block in E-Pfad:
In **Block**
- Sonst in aufgeteilte **Kante**

Einfügestellen für neue Berechnungen

Wo muß $t_e := e$ eingefügt werden?

An Eintrittspunkten in den Pfad!



- Wenn alle Nachfolger von externem Block in E-Pfad:
In **Block**
- Sonst in aufgeteilte **Kante**

Efügestellen für neue Berechnungen bestimmen

$\text{insert}(b_h)$ Füge $t_e := e$ am Ende von Block b_h ein

$\text{insert}(b_h, b_j)$ Füge $t_e := e$ in aufgeteilte Kante (b_h, b_j) ein
 $b_h \notin \text{E-Pfad}, b_j \in \text{E-Pfad } (b_i, \dots, b_k]$

Berechnung

$$\text{insert}(b_h) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \prod_s \text{epsin}(s)$$

$$\text{insert}(b_h, b_j) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \neg \text{insert}(b_h) \cdot \text{epsin}(b_j)$$

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

Einfügestellen für neue Berechnungen bestimmen

$\text{insert}(b_h)$ Füge $\tau_e := e$ am Ende von Block b_h ein

$\text{insert}(b_h, b_j)$ Füge $\tau_e := e$ in aufgeteilte Kante (b_h, b_j) ein

$b_h \notin \text{E-Pfad}, b_j \in \text{E-Pfad } (b_i, \dots, b_k]$

Berechnung

$$\text{insert}(b_h) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \prod_s \text{epsin}(s)$$

$$\text{insert}(b_h, b_j) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \neg \text{insert}(b_h) \cdot \text{epsin}(b_j)$$

OC

A. Koch

Einleitung

PRE

Konzepte

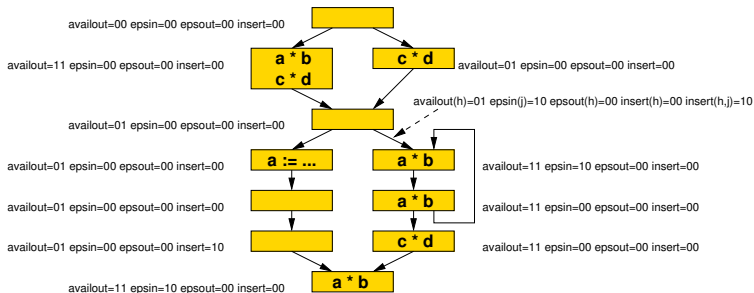
E-Pfad PRE

Notation

E-Pfade

Datenfluss

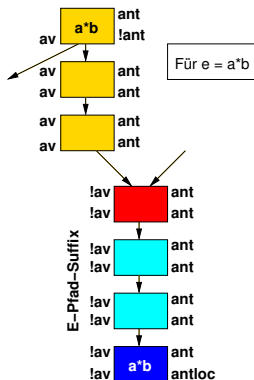
Beispiel Einfügen von Berechnungen



$$\text{insert}(b_h) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \prod_s \text{epsin}(s)$$

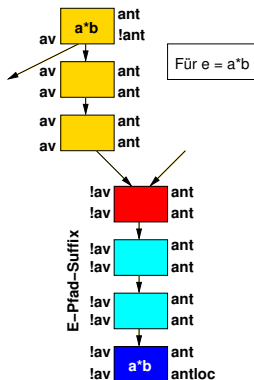
$$\text{insert}(b_h, b_j) = \neg \text{availout}(b_h) \cdot \neg \text{epsout}(b_h) \cdot \neg \text{insert}(b_h) \cdot \text{epsin}(b_j)$$

Am Anfang des E-Pfades!



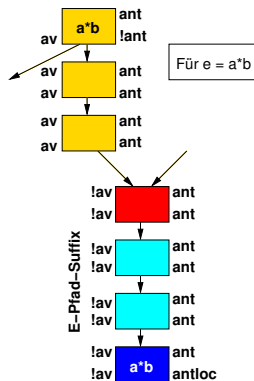
➔ Startblock des E-Pfades bestimmen!

Am Anfang des E-Pfades!

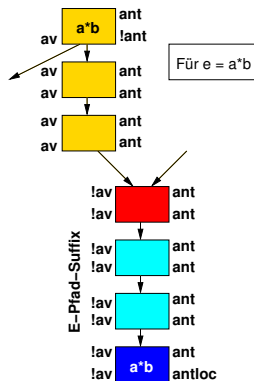


➔ **Startblock** des E-Pfades bestimmen!

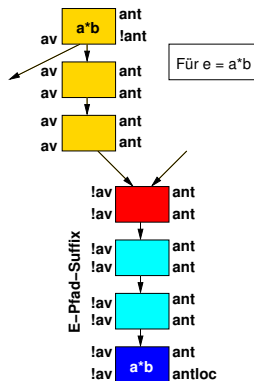
- Beginne bei bekanntem Block und suche **rückwärts**
- Beginne bei **Startblock des E-Pfad-Suffix**
 - ... falls E-Pfad einen Suffix hat
- Sonst: Suche von **Endblock des E-Pfades** aus rückwärts
- Bis nicht-redundante Berechnung von e gefunden



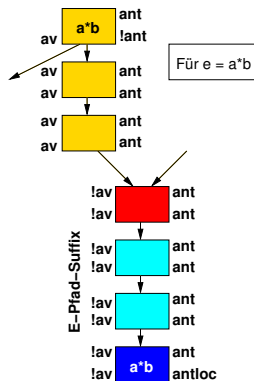
- Beginne bei bekanntem Block und suche **rückwärts**
- Beginne bei **Startblock des E-Pfad-Suffix**
 - ... falls E-Pfad einen Suffix hat
- Sonst: Suche von **Endblock des E-Pfades** aus rückwärts
- Bis nicht-redundante Berechnung von e gefunden



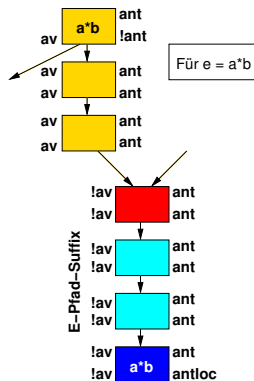
- Beginne bei bekanntem Block und suche **rückwärts**
- Beginne bei **Startblock des E-Pfad-Suffix**
 - ... falls E-Pfad einen Suffix hat
- Sonst: Suche von **Endblock des E-Pfades** aus rückwärts
- Bis nicht-redundante Berechnung von e gefunden



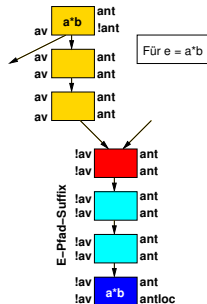
- Beginne bei bekanntem Block und suche **rückwärts**
- Beginne bei **Startblock des E-Pfad-Suffix**
 - ... falls E-Pfad einen Suffix hat
- Sonst: Suche von **Endblock des E-Pfades** aus rückwärts
- Bis nicht-redundante Berechnung von e gefunden



- Beginne bei bekanntem Block und suche **rückwärts**
- Beginne bei **Startblock des E-Pfad-Suffix**
 - ... falls E-Pfad einen Suffix hat
- Sonst: Suche von **Endblock des E-Pfades** aus rückwärts
- Bis nicht-redundante Berechnung von e gefunden



- svupin(*b*)** Sichere
Berechnungsergebnis für *e*
über Blockeingang von *b*
- svupout(*b*)** Sichere
Berechnungsergebnis für *e*
über Blockausgang von *b*
- save(*b*)** Sichere
Berechnungsergebnis in τ_e
in Block *b*



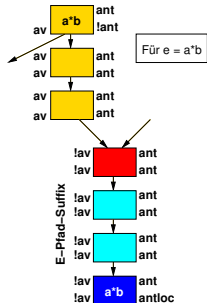
Berechnung

$$\begin{aligned}
 \text{svupout}(b) &= (\sum_s (\text{epsin}(s) + \text{redund}(s) + \text{svupin}(s))) \cdot \text{availout}(b) \\
 \text{svupin}(b) &= \text{svupout}(b) \cdot \neg \text{comp}(b) \\
 \text{save}(b) &= \text{svupout}(b) \cdot \text{comp}(b) \cdot \neg (\text{redund}(b) \cdot \text{transp}(b))
 \end{aligned}$$

svupin(b) Sichere
Berechnungsergebnis für e
über Blockeingang von b

svupout(b) Sichere
Berechnungsergebnis für e
über Blockausgang von b

save(b) Sichere
Berechnungsergebnis in τ_e
in Block b



Berechnung

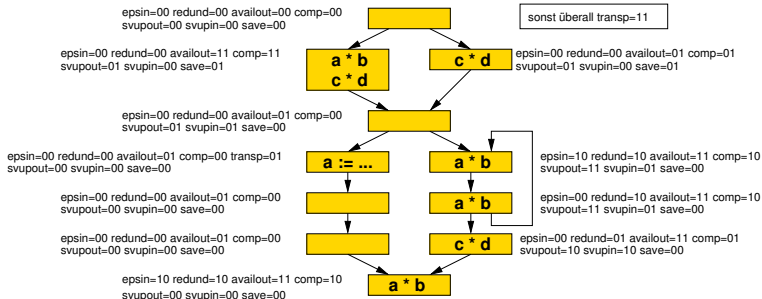
$$\text{svupout}(b) = (\sum_s (\text{epsin}(s) + \text{reund}(s) + \text{svupin}(s))) \cdot \text{availout}(b)$$

$$\text{svupin}(b) = \text{svupout}(b) \cdot \neg \text{comp}(b)$$

$$\text{save}(b) = \text{svupout}(b) \cdot \text{comp}(b) \cdot \neg (\text{reund}(b) \cdot \text{transp}(b))$$

Beispiel Bestimmung des E-Pfad-Startblöcke

Stellen zum Sichern der Berechnungsergebnisse



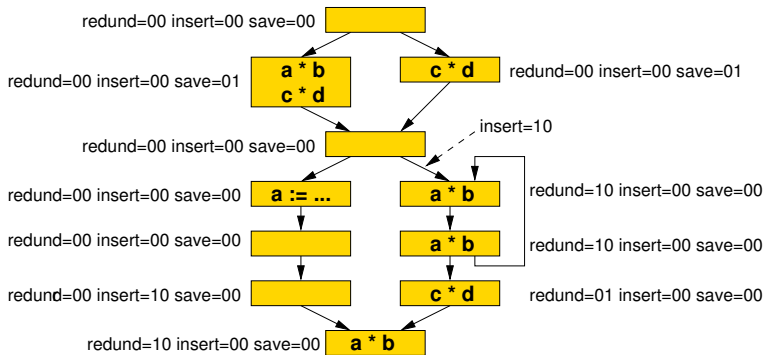
$$svpout(b) = (\sum_s (epsin(s) + redund(s) + svpin(s))) \cdot availout(b)$$

$$svpin(b) = svpout(b) \cdot \neg comp(b)$$

$$save(b) = svpout(b) \cdot comp(b) \cdot \neg (redund(b) \cdot transp(b))$$

Beispiel Zusammenfassung Datenflussanalyse

Bisher berechnete Ergebnisse



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

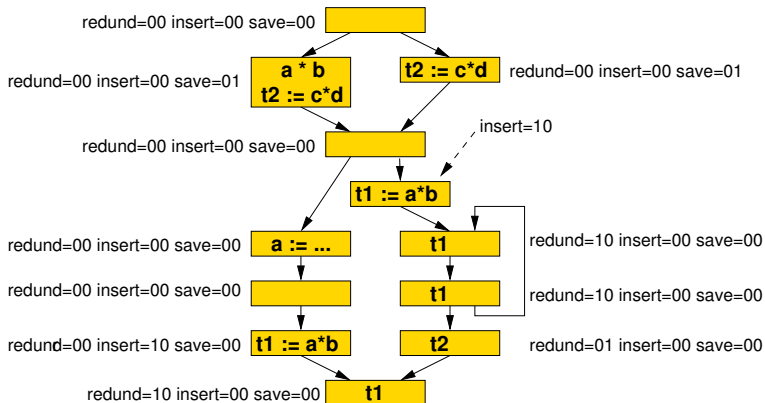
Notation

E-Pfade

Datenfluss

Beispiel PRE-Optimierung

Auf Basis der Datenfluss-Ergebnisse



OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflußproblems → D. Kumar 2006, geringerer Rechenaufwand

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflußproblems → D. Kumar 2006, geringerer Rechenaufwand

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflußproblems → D. Kumar 2006, geringerer Rechenaufwand

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflußproblems
→ D. Kumar 2006, geringerer Rechenaufwand

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflußproblems
→ D. Kumar 2006, geringerer Rechenaufwand

OC

A. Koch

Einleitung

PRE

Konzepte

E-Pfad PRE

Notation

E-Pfade

Datenfluss

- PRE ist eine sehr mächtige Optimierung
- Lösbar durch komplexes Datenflußproblem
 - Hier aber schon deutlich einfacher als klassische Verfahren!
- Noch weitere Verfeinerung möglich
 - Optimierung auf SSA-Form → SSA-PRE
 - Verfeinerte Stellung des Datenflussproblems
→ D. Kumar 2006, geringerer Rechenaufwand