Compiler 2

3. Block: SSA-Form und CFG→SSA Wandlung

TECHNISCHE UNIVERSITÄT DARMSTADT

Organisatorisches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Unterbrechung des normalen Compile-Flusses
- Einführung einer neuen Zwischendarstellung

Relevante Papers 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Ab jetzt Auszüge aus:

Single-Pass Generation of Static Single Assignment Form for Structured Languages

MARC M. BRANDIS and HANSPETER MÖSSENBÖCK

ACM Transactions on Programming Languages and Systems 16(6): 1684-1698, Nov.1994

- Erzeugung von SSA-Form aus strukturierten Programmierprachen
- Sehr gut zu lesen

Relevante Papers 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Practical Improvements to the Construction and Destruction of Static Single Assignment Form

BRIGGS, COOPER, HARVEY, SIMPSON

SOFTWARE: PRACTICE AND EXPERIENCE, VOL. 28(8), 128 (July 1998)

- Umwandeln aus der SSA-Form ($\rightarrow$ nächste Woche)
- Recht gut zu lesen

Relevante Papers 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Efficiently Computing Static Single Assignment Form and the Control Dependence Graph

CYTRON, FERRANTE, ROSEN, WEGMAN, ZADECK

ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 13 , Issue 4 (October 1991)

- *Das* SSA-Paper schlechthin
- Keine ganz einfache Lektüre mehr
- Aber viele Details nur hier behandelt
  - Z.B. Behandlung von Arrays etc.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorschau Redundanzeliminierung

Optimieren redundanter Berechnungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Eingabe-Code

```
a := b + c;
b := a - d;
c := b + c;
d := a - d;
```

Value Numbering

```
a3 := b1 + c2;
b5 := a3 - d4;
c6 := b5 + c2;
d5 := a3 - d4;
```

Umschreiben

```
a := b + c;
b := a - d;
c := b + c;
d := b;
```

➥Redundante Berechnung von `a - d` vermieden

Probleme

TECHNISCHE UNIVERSITÄT DARMSTADT

Bisher: Zugriff auf Werte über Namen (von Variablen)

Eingabe-Code

$a \leftarrow x + y$
$* \quad b \leftarrow x + y$
$a \leftarrow 17$
$* \quad c \leftarrow x + y$

Value Numbering

$a^3 \leftarrow x^1 + y^2$
$* \quad b^3 \leftarrow x^1 + y^2$
$a^4 \leftarrow 17$
$* \quad c^3 \leftarrow x^1 + y^2$

Umgeschrieben

$a^3 \leftarrow x^1 + y^2$
$* \quad b^3 \leftarrow a^3$
$a^4 \leftarrow 17$
$* \quad c^3 \leftarrow a^3$ (oops!)

- Zugriff auf Wert 3 über Variablennamen `a`
- Nicht mehr möglich!
- Möglichkeiten
  - Führe Buch über den Wert haltende Variablen (hier `b`)
  - Mache Sicherheitskopien von Variablen (a^3 nach t^3)
  - Vergebe eindeutige Namen für Zuweisungen
    - Kein Überschreiben mehr möglich

Eindeutig benannte Zuweisungsziele

TECHNISCHE UNIVERSITÄT DARMSTADT

Durchnumerieren der LHS-Variablen
($\rightarrow$ Variablenversionen)

Eingabe-Code	Value Numbering	Umgeschrieben
$a_0 \leftarrow x_0 + y_0$	$a_0^3 \leftarrow x_0^1 + y_0^2$	$a_0^3 \leftarrow x_0^1 + y_0^2$
$*\ b_0 \leftarrow x_0 + y_0$	$*\ b_0^3 \leftarrow x_0^1 + y_0^2$	$*\ b_0^3 \leftarrow a_0^3$
$a_1 \leftarrow 17$	$a_1^4 \leftarrow 17$	$a_1^4 \leftarrow 17$
$*\ c_0 \leftarrow x_0 + y_0$	$*\ c_0^3 \leftarrow x_0^1 + y_0^2$	$*\ c_0^3 \leftarrow a_0^3$

- Wert 3 verfügbar als a_0^3
- Hier nur etwas mehr Verwaltungsaufwand
- Aber echte Probleme kommen noch!
  - Beim Überschreiten von Basisblockgrenzen
  - Eine Lösung: Static Single Assignment-Form von CFGs

Static Single Assignment-Form

Idee SSA

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Normal                    // SSA-Form
v := 0;                      v1 := 0;
x := v + 1;                  x1 := v1 + 1;
v := 2;                      v2 := 2;
y := v + 3                   y1 := v2 + 3
```

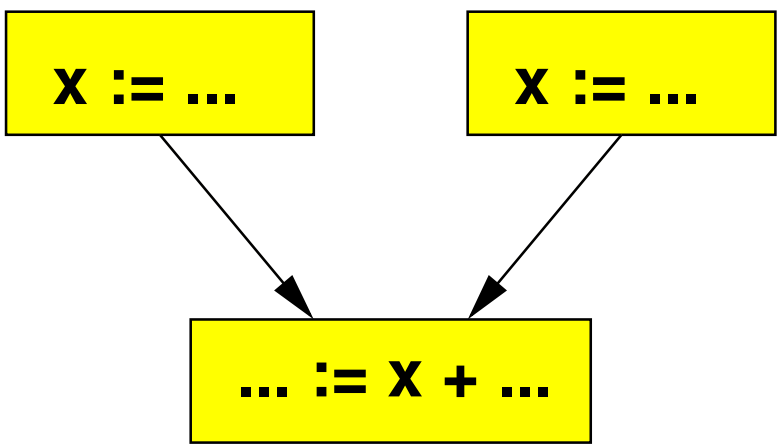
- Zur Compile-Zeit (also statisch)
- Jeder Wert wird an genau eine eigene Variable zugewiesen
  - Erzeuge eindeutige Namen für gleiche Zuweisungsziele
  - Numerierte Variablen sind Wertinstanzen der ursprünglichen Variablen
- Jeder Operand hat somit genau eine Definition in BB
- Letzte Definition ist die aktuelle

Problem

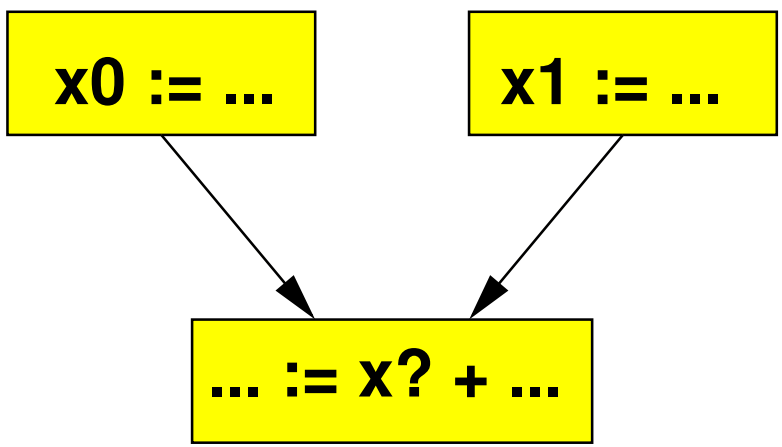
TECHNISCHE UNIVERSITÄT DARMSTADT

- Was, wenn mehrere “letzte” Definitionen? (z.B. then/else-Zweige: mehrere BBs)
- Sogenannte *merge points*
- Zusammenführen von mehreren “letzten” Definitionen

Ursprünglicher CFG



“Letzte” Definition von `x`?



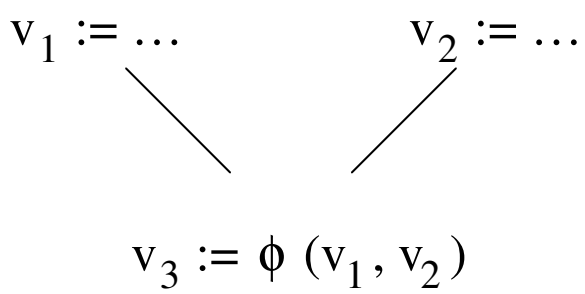
Problem: Kontrollfluß

TECHNISCHE UNIVERSITÄT DARMSTADT

Was passiert, wenn zwei Werte der gleichen Variable aufeinanderstoßen?

- An sogenanntem *merge* oder *join*-Punkten im Kontrollflußgraphen

➥Auflösung über Phi-Funktion

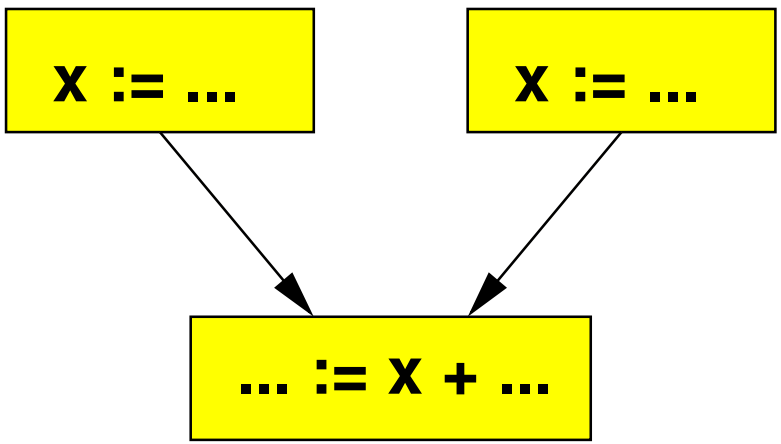


- Für jeden Kontrollzweig einen Parameter
  - Den jeweiligen Wert
- Liefert als Ergebnis den Wert entsprechend der genommenen Kante
  - Von welchem Zweig kamen wir?
  - Welcher Wert ist also der richtige?

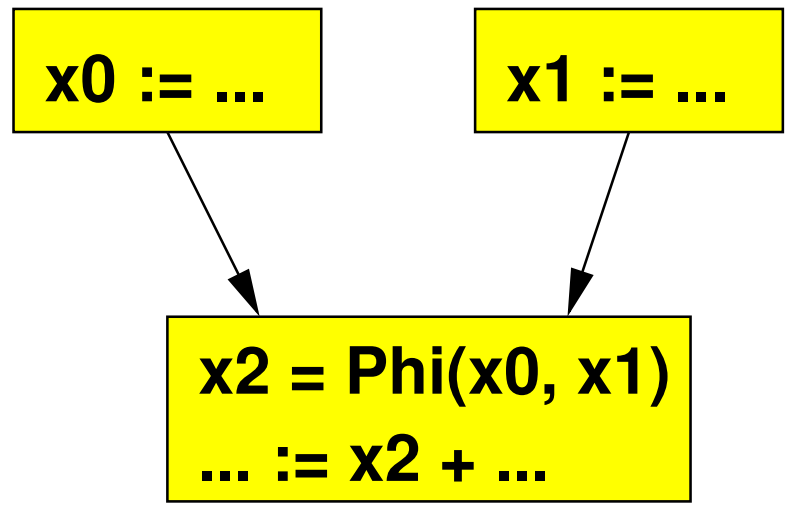
Auflösung des Problems

TECHNISCHE UNIVERSITÄT DARMSTADT

Ursprünglicher CFG



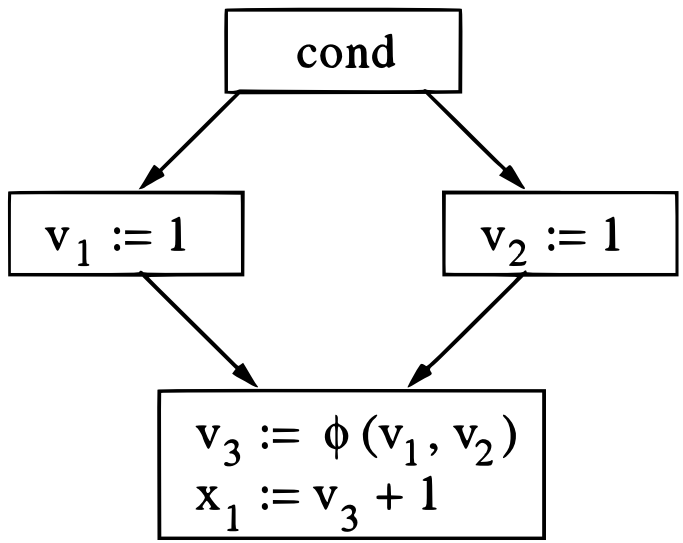
Auflösung durch ϕ-Funktion



Beispiel SSA-Form: IF-Statement

TECHNISCHE UNIVERSITÄT DARMSTADT

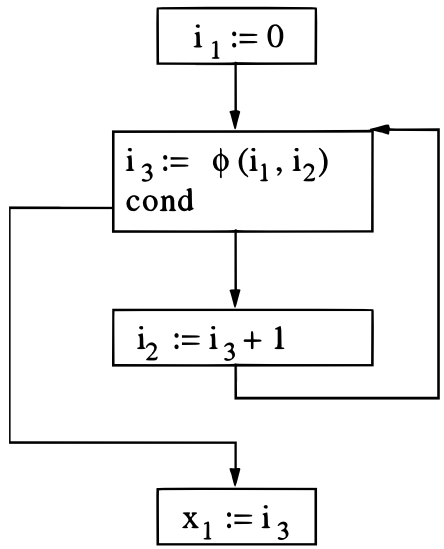
```
IF cond THEN
    v := 1
ELSE
    v := 2
END;
x := v + 1
```



Beispiel SSA-Form: WHILE-Statement

TECHNISCHE UNIVERSITÄT DARMSTADT

```
i := 0;
WHILE cond DO
    i := i + 1
END;
x := i
```



Beachte: Entscheidung, ob Wert von vor oder nach dem Schleifenkörper genommen wird.

Vorteile SSA

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für jeden Wert genau eine Definition
- Jede Zuweisung legt neuen Wert an
- Kein Auslöschen (*kill*) von Werten möglich
- Wenn zwei Ausdrücke textuell gleich sind
- . . . liefern sie das gleiche Ergebnis

Transformation in SSA-Form 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Drei Teilprobleme

1. Eindeutige Namen für Werte
   - Einfach durchnumerieren
2. Einfügen von Phi-Funktionen
   - Holzhammermethode
     - An jedem join-Point für alle Variablen Phi-Funktionen einfügen
   - Erzeugt sehr viele Phi-Funktionen, die meisten unnötig
3. Umbenennen von benutzten Variablen in passende Werte
   - Wieder recht einfach
   - Referenziert letzte Definition

Transformation in SSA-Form 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Allgemeine Lösung

- Cytron et. al. 1991
- Vorgehen: Berechnen von Dominatorgrenzen
- “Gerade nicht mehr” von Knoten X dominierte Knoten
- Hier nicht mehr klar, ob Definitionen aus X noch gelten
- Einfügen von Phi-Knoten nur für die Variablen, bei denen entschieden werden muß
  - Aufeinandertreffen von verschiedenen Definitionen an Dominatorgrenzen
- Algorithmus nicht trivial . . .

Sonderfall: Strukturierte Programmiersprachen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Keine GOTOs
- Nur strukturierte Anweisungen
  - IF
  - CASE
  - WHILE
  - REPEAT
  - FOR

➥Viel einfacheres und schnelleres Vorgehen möglich

➥Brandis/Mössenböck 1994

Unser Ansatz für Triangle!

Einschränkungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Aus Zeitgründen in der Vorlesung keine detailierte Behandlung von

- Arrays
- Records
- Prozeduraufrufen
- Verschachtelten Geltungsbereichen

Alles handhabbar
. . . aber aufwändig und lenkt von Kernideen ab.

Bei Interesse (oder Bedarf!): Cytron et al., Abschnitt 3.1

Benennen von Werten in Basisblöcken

TECHNISCHE UNIVERSITÄT DARMSTADT

Assignments (original form)	*Assignments (SSA form)*	*Current values* v	x
		v_0	x_0
$v := 0;$	$v_1 := 0;$	v_1	x_0
$x := v + 1;$	$x_1 := v_1 + 1;$	v_1	x_1
$v := 2$	$v_2 := 2$	v_2	x_1

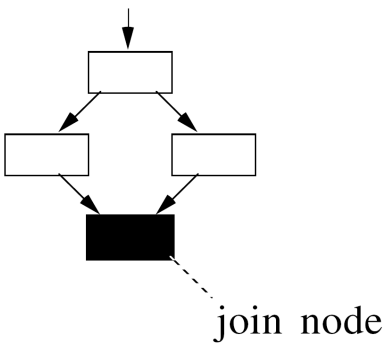
- Jede Zuweisung an v erzeugt neuen Wert v_i
- Nach Zuweisung ist v_i aktueller Wert von v
- Ersetze alle folgenden Verwendungen von v durch v_i
- Verwaltung z.B. in extra Tabelle während Umformung

Join-Knoten 1

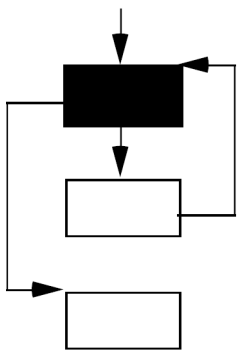
TECHNISCHE UNIVERSITÄT DARMSTADT

Bei strukturierten Programmiersprachen:
Alle Join-Knoten sind durch Konstrukte bereits vorgegeben

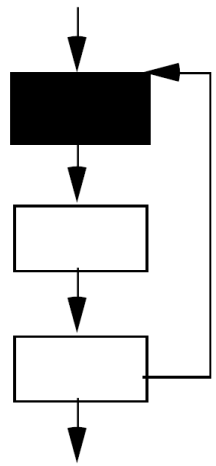
IF, CASE



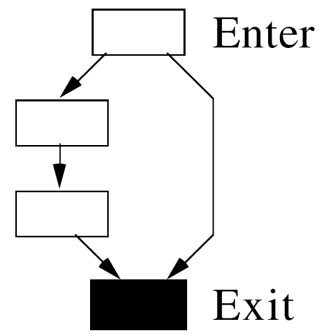
WHILE, FOR



REPEAT



procedure



Platzhalter
(siehe später!)

Join-Knoten 2

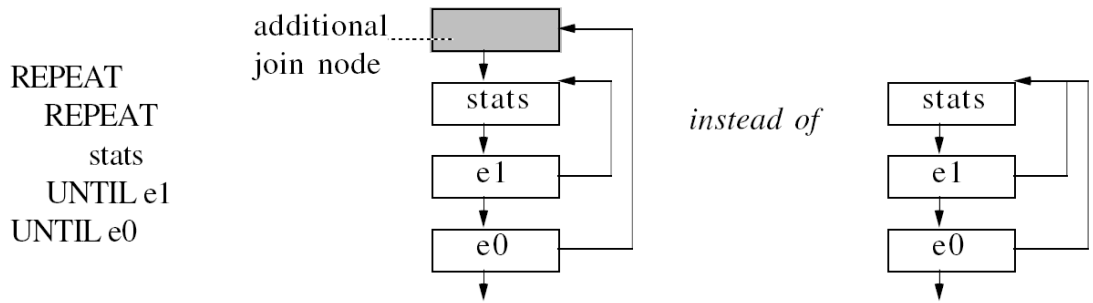
TECHNISCHE UNIVERSITÄT DARMSTADT

- Strukturen können verschachtelt sein
  - Bearbeite von innen nach aussen
  - Innerster Join-Knoten ist aktueller Join-Knoten
- Erzeuge keine speziellen Knoten für Joins
- Verwende bisherige Blöcke weiter

Join-Knoten 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausnahme: Verschachtelte REPEAT-Anweisungen



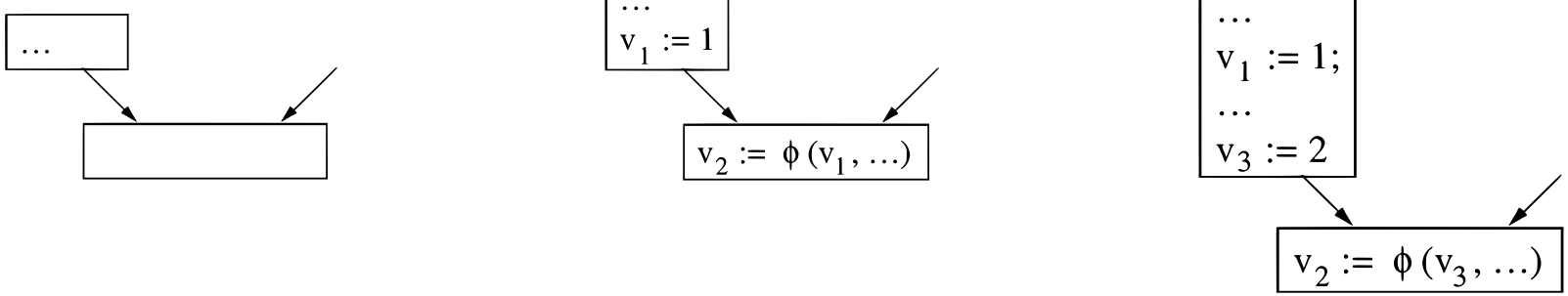
- Für spätere Optimierung hilfreich
- Sonst kein Ziel für aus der inneren Schleife bewegte Berechnungen

Phi-Knoten Erzeugen 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jede Zuweisung gehört zu einem Zweig des Kontrollflußgraphen
- Jede Zuweisung erzeugt einen neuen Wert
  - Ggf. auch bei Prozeduraufruf (`var`, global, nicht-lokal)
- Irgendwann trifft der Wert auf einen Join-Knoten
- Dort Unterscheidung zwischen allen Werten für diese Variable

➥Jede Zuweisung erzeugt oder modifiziert Phi-Funktion für Variable



Phi-Knoten Erzeugen 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Phi-Operand entsprechend dem bearbeiteten Zweig
- . . . wird jeweils auf letzten aktuellen Wert gesetzt
- Phi-Funktionen treten selber in Zuweisungen auf
- Erzeugen also selber neue Werte
- Führen zu weiteren Phi-Funktionen in nächstäußerem Join-Knoten
- Ende bei Erreichen des Exit-Knotens

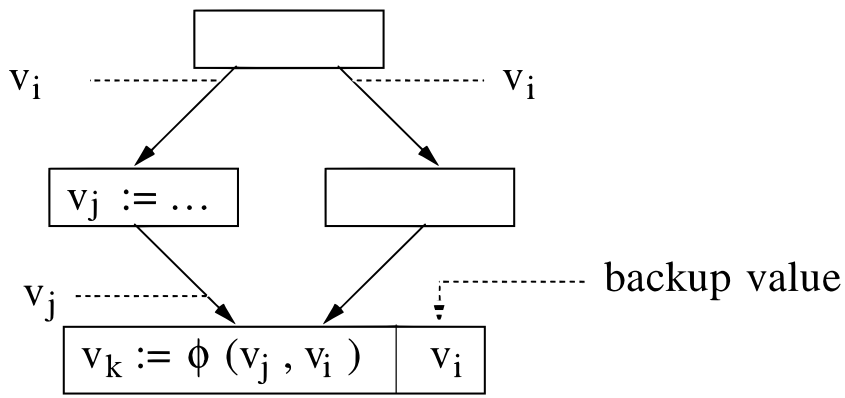
Vorgehen: Erzeugen eines CFGs in SSA-Form je Prozedur durch Traversieren des ASTs

Könnte aber auch direkt beim Parsen geschehen.

Phi-Knoten für IF-Anweisungen 1

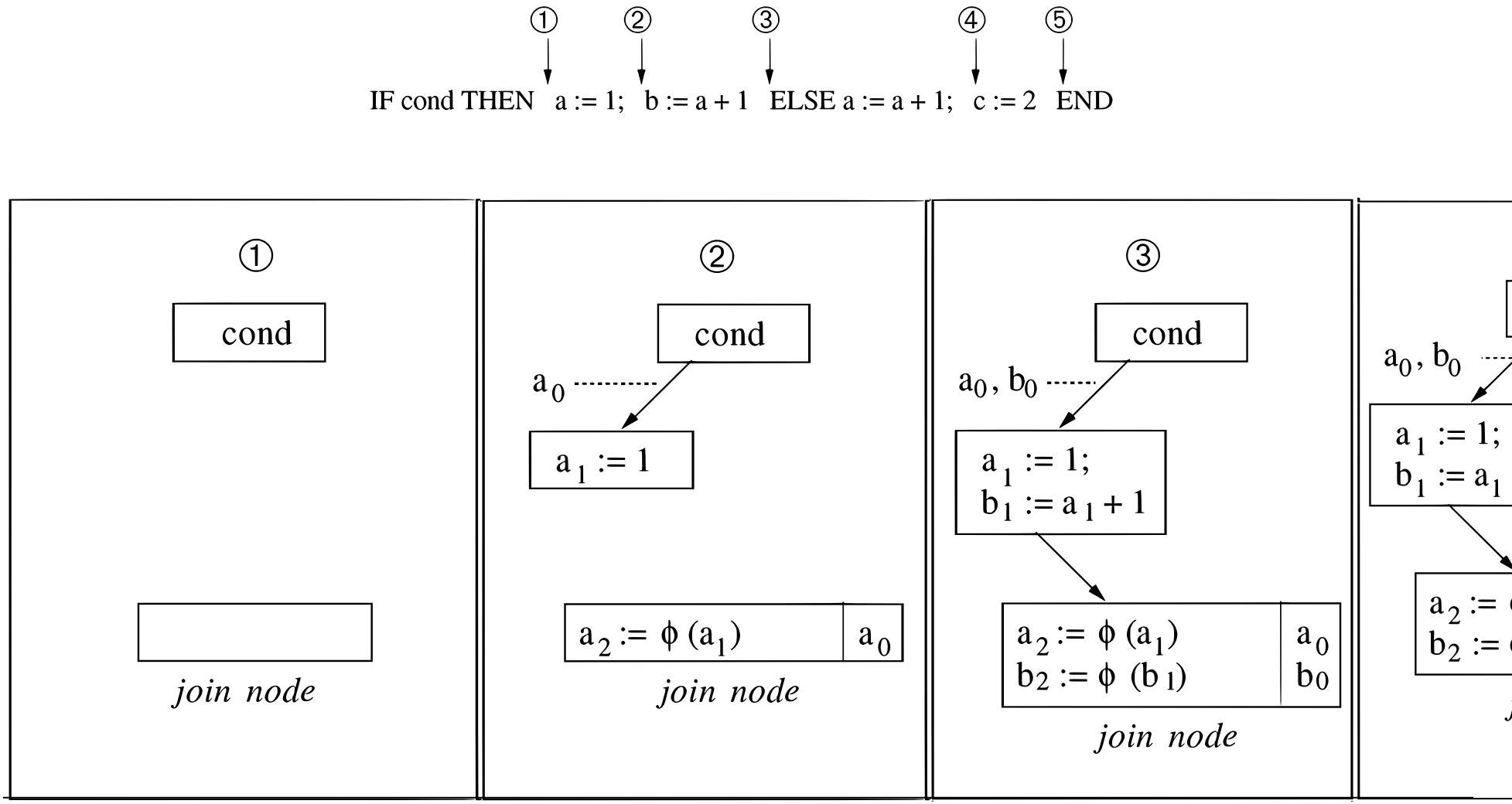
TECHNISCHE UNIVERSITÄT DARMSTADT

1. Bei Erreichen von IF: Erzeuge neuen Join-Knoten
   - Wird Phi-Funktionen aus THEN/ELSE enthalten
   - Wird später in den CFG eingehängt
2. Bearbeite THEN-Zweig, für eine Zuweisung an v
   - 1. Mal: Lege leere Phi-Funktion (Identität) für v an, sichere Wert v_i vor IF zusammen mit Phi-Funktion
   - Sonst: Setze Phi-Operand auf jeweils aktuellen Wert v_j
3. Bearbeite ELSE-Zweig
   - Setze aktuelle auf gesicherte Werte (pre-IF) zurück
   - Dann gleiches Vorgehen wie im THEN-Zweig



Phi-Knoten für IF-Anweisungen 2

TECHNISCHE UNIVERSITÄT DARMSTADT



Phi-Knoten für IF-Anweisungen 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Nach Abarbeiten von THEN und ELSE-Zweigen:
Festlegen des Join-Blocks (*commit*)

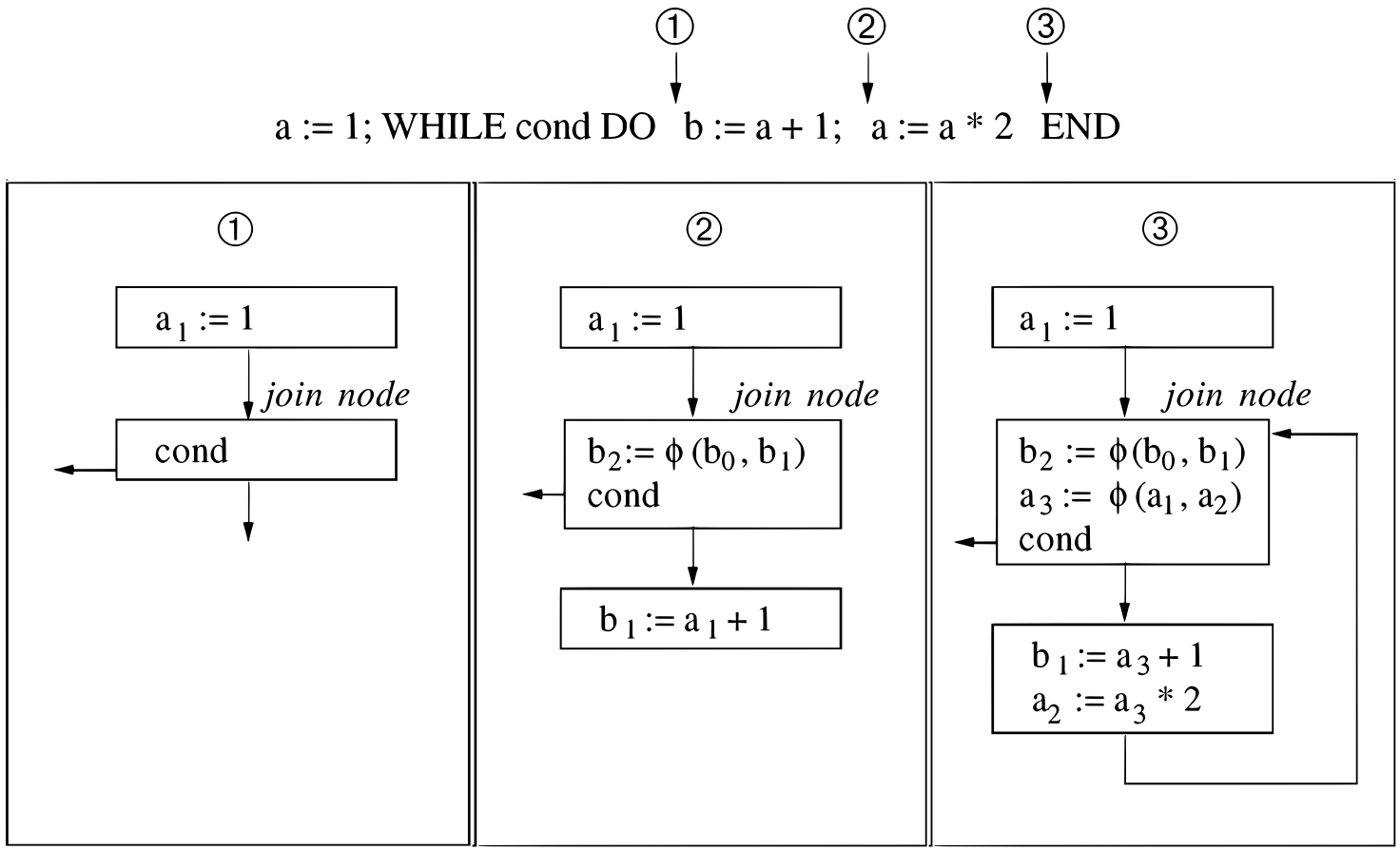
- Join-Block selber bearbeiten
- Werte Zuweisungen von Phi-Funktionen aus
- Trage neue Phi-Funktionen in nächstäußeren Join-Block ein
  - Join-Block der umschließenden Kontrollstruktur
- Trage dort LHS der Phi-Zuweisungen als aktuelle Werte der Variablen ein
- Hänge aktuellen Join-Block in CFG ein

Phi-Knoten in WHILE-Anweisungen 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Join-Knoten von WHILE-Anweisung ist Kopfknoten
  - Zusammentreffen von Schleifeneintritt und Rückwärtskante im CFG
- Bearbeitung des Schleifenkörpers analog zur IF-Anweisung, aber
- Bei Eintragen einer neuen Phi-Funktion in Kopfknoten
- . . . entsteht neuer aktueller Wert
- Alle lesenden Benutzungen der Variable im Schleifenkörper durch aktuellen Wert ersetzen
  - Verwalte Liste aller im Schleifenkörper benutzten Werte
  - Sogenannte *use chain*
  - Kann für schnelle Korrektur (Ändern der Versionsnummer) benutzt werden

Phi-Knoten für WHILE-Anweisungen 2



Beachte: Ersetzung von a_1 durch a_3!

Phi-Knoten für WHILE-Anweisungen 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nach der Bearbeitung des Schleifenkörpers
- . . . Festlegen der Phi-Zuweisungen im Join-Knoten
- Erzeugt neue Phi-Funktionen in nächstäußerem Join-Knoten
- Legt neue aktuelle Werte für nachfolgende Anweisungen fest
  - Im Beispiel: a_3 und b_2

CASE und FOR würden analog zu IF und WHILE bearbeitet

Phi-Knoten für REPEAT-Anweisungen 1

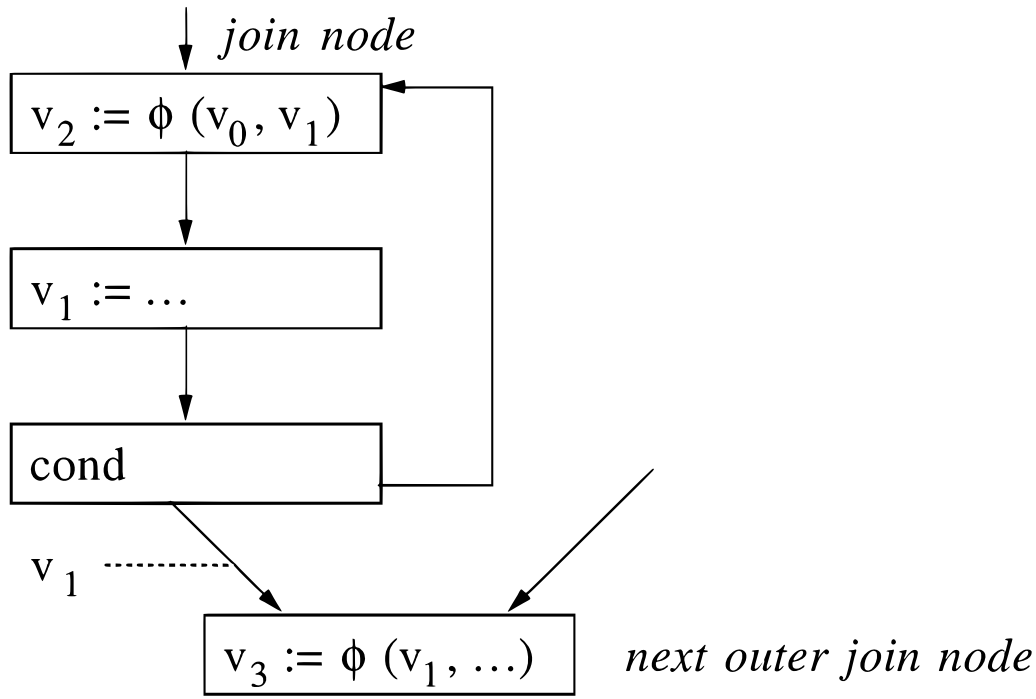
TECHNISCHE UNIVERSITÄT DARMSTADT

- Sonderfall!
- Konstrukt wird nicht über Join-Knoten verlassen
- Analog zu WHILE: Join-Knoten ist Schleifenkopf
  - Hier auch Phi-Zuweisungen untergebracht
- Aber Unterschied beim Festlegen des Join-Blocks!
- Aktueller Wert ist nicht Ziel der Phi-Zuweisung im Kopf (wie bei WHILE)
- . . . sondern Wert zugeordnet der Rückwärtskante
  - Sonst wären Änderungen nach genau einem Schleifendurchlauf nicht sichtbar
- Gleichen Wert auch für Operanden nächstäußerer Phi-Funktion verwenden

Phi-Knoten für REPEAT-Anweisungen 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
REPEAT
    v := ...
UNTIL cond
```



Beachte: Weiterverwendung von v_1, nicht von v_2

Kernalgorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

- INSERTPHI
  - Erzeugt neue oder modifiziert bestehende Phi-Zuweisung in Join-Knoten b
  - Aufruf: INSERTPHI(b, i, v_i, v_{old})
    - Zur Bearbeitung von Zuweisung v_i := ...
    - ...die im i-ten, zum Block b führenden Zweig steht
    - v_{old} ist aktueller Wert vor dieser Zuweisung
    - Wird als Sicherheitskopie abgespeichert
- COMMITPHI
  - Legt die Phi-Zuweisungen in einem Join-Knoten b fest
  - Bestimmt aktuelle Werte
  - Propagiert neue Phi-Zuweisungen in nächstäußeren Join-Knoten B, über die Kante l kommend

InsertPhi

TECHNISCHE UNIVERSITÄT DARMSTADT

PROCEDURE InsertPhi (b: Node; i: INTEGER; v_i, v_{old}: Value);
BEGIN
  IF *b contains no ϕ-assignment for v* THEN
    Insert "$v_j := \phi(v_{old}, \ldots, v_{old}) / v_{old}$" in b;
    IF *b is a join node of a loop* THEN
      Rename all mentions of v_{old} in the loop to v_j
    END
  END;
  Replace i-th operand of v's ϕ-assignment by v_i
END InsertPhi;

CommitPhi

TECHNISCHE UNIVERSITÄT DARMSTADT

```
PROCEDURE CommitPhi (b: Node);
BEGIN
   FOR all φ-instructions "v_i := φ (v_0, ..., v_n) / v_old" in b DO
      IF b is a join node of a repeat THEN val := v_n ELSE val := v_i END;
      Make val the current value of v;
      InsertPhi(B, I, val, v_old)
   END
END CommitPhi;
```

Hier Annahme: Letzter Zweig *n* ist Rückwärtskante der REPEAT-Schleife

Hinweise

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hier nicht gezeigt: Rücksetzen auf v_{old} bei Bearbeitung des nächsten Zweiges
- Variablen in Triangle durch Verweise auf Definitionen kennzeichnen
- Keine String-Vergleiche mehr nötig!
- Werte sind dann Tupel (Definition, Versionsnummer)
- Prozeduraufrufe wie Zuweisungen behandeln
  - LHS: `var`-Parameter, geschriebene nicht-lokale und globale Variablen
  - RHS: Parameter (`var` und Wert), gelesene nicht-lokale und globale Variablen

Beispiel Prozeduraufruf

TECHNISCHE UNIVERSITÄT DARMSTADT

```
let
  var f : Integer;
  var g : Integer;
  var n : Integer;
  proc p() ~ begin f := 2*f; g := g+1 end
in begin
   n := 1; f := 2; g := 3;
   while n < 10 do begin
     p();
     n := n + 1
   end;
   putint(f); puteol(); putint(g)
end
```

- Sehe `p()` an als `{f,g} = p {f,g}`
- RHS: Operator `p`, angewandt auf Werte `f` und `g`
- LHS: Erzeuge neue Versionen von `f` und `g`

In SSA-Form

TECHNISCHE UNIVERSITÄT DARMSTADT

- `{f3,g3} = p() {f2,g2}` in Schleife
- Im Kopfknoten nun:

```
f2 = Phi (f1, f3)
g2 = Phi (g1, g3)
n2 = Phi (n1, n3)
n2 < 10
```

- Details in Cytron, Abschnitt 3.1
  - Arrays, Records, Prozeduren und Funktionen
  - Besser als nachlesen: Idee verstanden haben :-)

Rückwandlung aus SSA-Form

Rückwandlung aus der SSA-Form 1

TECHNISCHE UNIVERSITÄT DARMSTADT

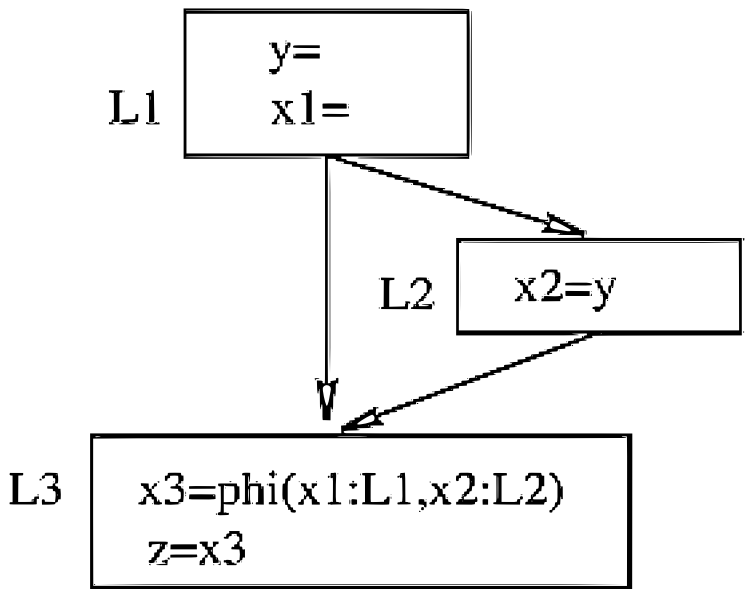
- Normale Prozessoren haben keine Phi-Instruktion
- Phi-Instruktionen müssen entfernt werden

Rückwandlung aus SSA-Form 2

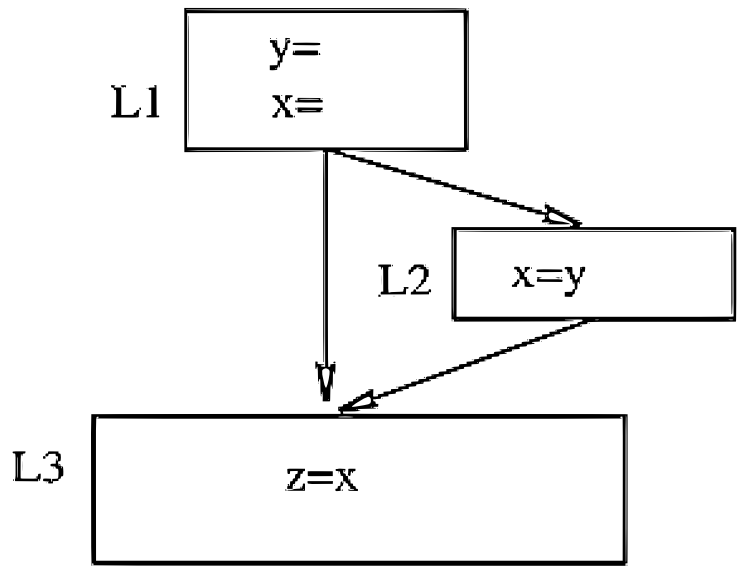
TECHNISCHE UNIVERSITÄT DARMSTADT

Naive Idee: Phi einfach löschen und Wertnummern entfernen

Vorher:



Nachher:



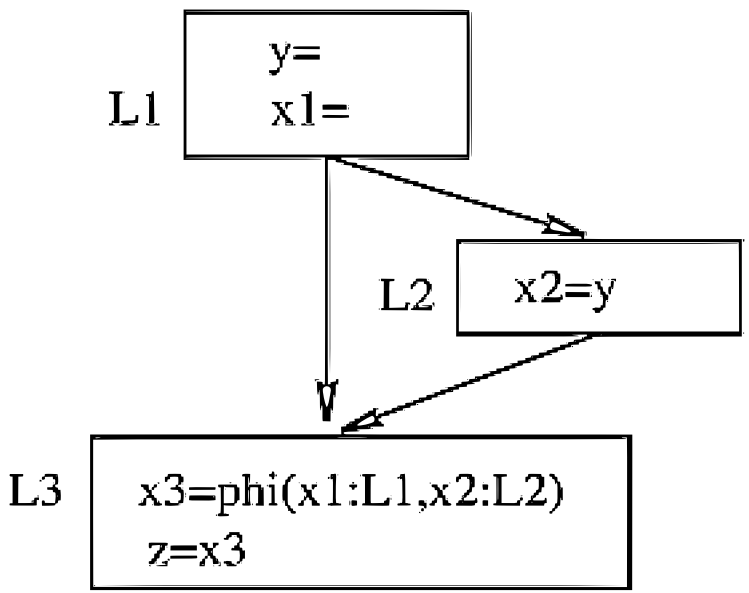
. . . so weit, so gut.

Rückwandlung aus SSA-Form 3

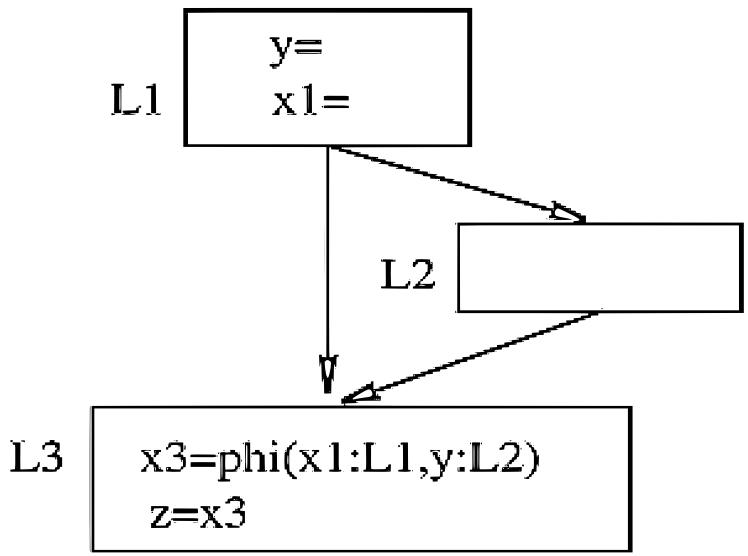
TECHNISCHE UNIVERSITÄT DARMSTADT

Jetzt Annahme: Einfache Optimierung hat stattgefunden

Vor Copy-Propagation



Nach Copy-Propagation



Rückwandlung durch einfaches Löschen . . . geht schief:

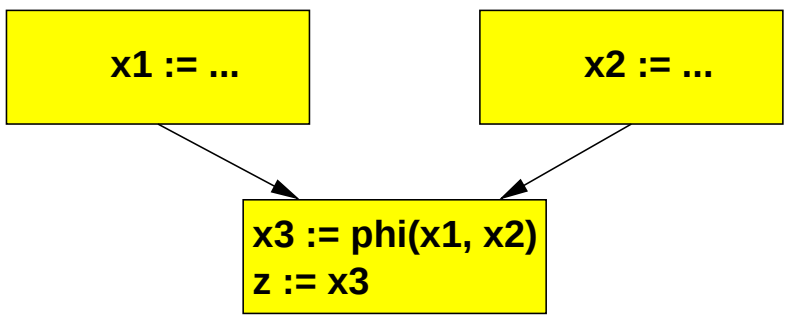
- Phi-Funktion auflösen nach x oder y?

Rückwandlung aus SSA-Form 4

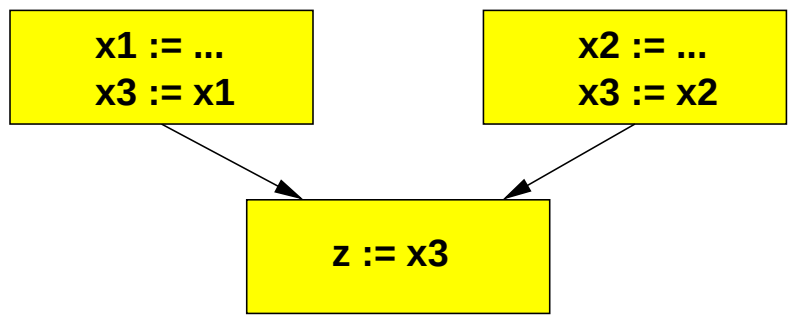
TECHNISCHE UNIVERSITÄT DARMSTADT

Besserer Ansatz: Füge Kopieroperationen in Vorgängerblöcke der Phi-Funktion ein

Vorher



Nachher



Zielführender als naives Löschen!

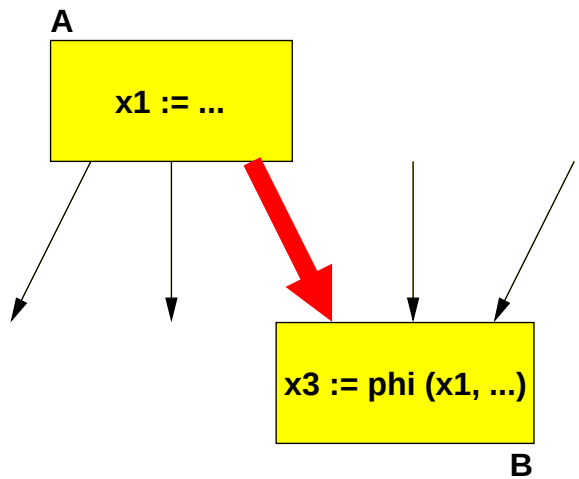
Rückwandlung aus der SSA-Form 5

TECHNISCHE UNIVERSITÄT DARMSTADT

Problemfall: Kritische Kanten

Kritische Kontrollflusskante

Eine kritische Kante im CFG verläuft von einem Block mit mehreren Nachfolgern zu einem Block mit mehreren Vorgängern.

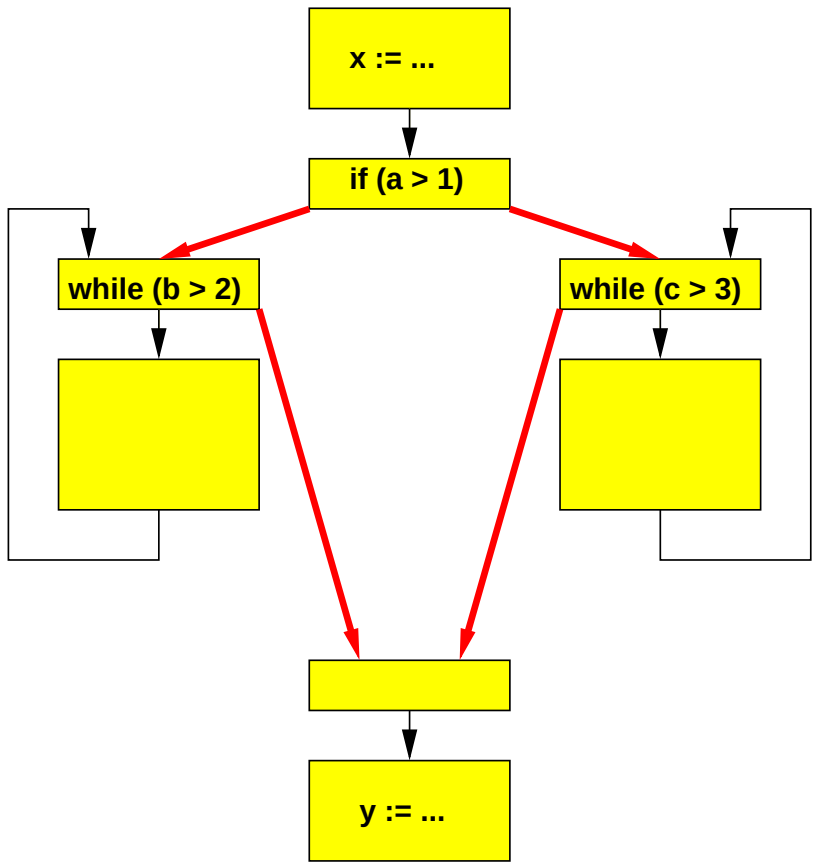


Rückwandlung aus der SSA-Form 6

TECHNISCHE UNIVERSITÄT DARMSTADT

Praktisch: Können kritische Kanten in strukturierten CFGs à la Triangle auftreten?

```
x := ...
if (a > 1) then {
  while (b > 2) do {
  }
} else {
  while (c > 3) do {
  }
}
y := ...
```

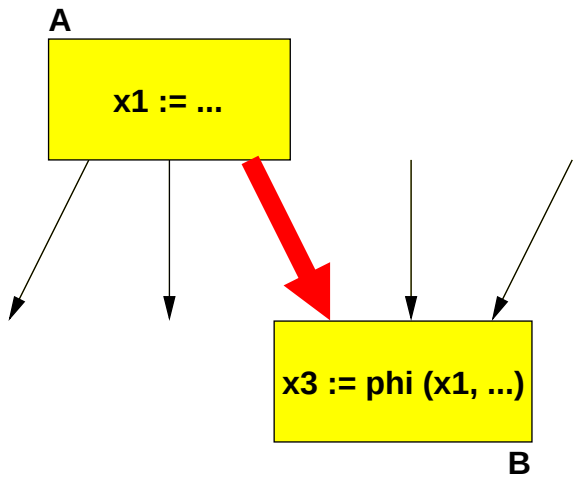


Rückwandlung aus der SSA-Form 7

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem bei kritischen Kanten

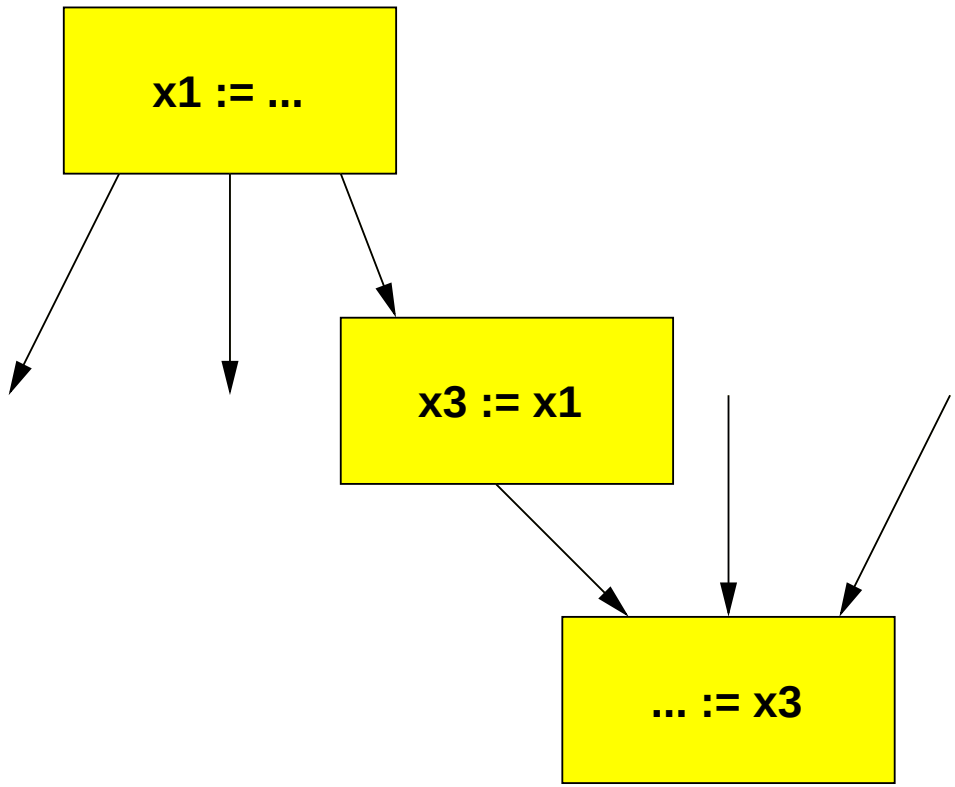
- Wo Kopierzuweisungen von A bei Auflösen der Phi-Funktion in B unterbringen?
- Am Ende von A?
  - Nicht effizient (unnötige Anweisungen für Nachfolger außer B)
- Am Anfang von B?
  - Geht nicht, da dann alle Vorgänger von B die Kopie von A bekommen!



Rückwandlung aus der SSA-Form 8

TECHNISCHE UNIVERSITÄT DARMSTADT

Einfache Lösung:
Kante aufspalten und neuen Block einfügen!



Funktioniert immer!

Rückwandlung aus der SSA-Form 9

TECHNISCHE UNIVERSITÄT DARMSTADT

Nachteil: Verlangsamt möglicherweise Programm

- Beispiel: Zusätzliche Sprunganweisung bei REPEAT/UNTIL

Abhilfe: Gezielteres Einfügen von Kopien

- Briggs 1998 oder Sreedhar 1999

Kommt noch in eigener Vorlesung!

Rückwandlung aus SSA-Form 10

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aber nicht alle kritischen Kanten sind relevant
- Nur solche vor Blöcken mit phi-Funktionen

Damit einfache Vorgehensweise zur Rückwandlung

- Teile phi-Funktion in Kopieranweisungen auf
- Lege Kopieranweisung am Ende des entsprechenden Vorgängerknotens ab
- Es sei denn, dass Kante zum Vorgänger kritisch ist
- Dann Kante aufspalten, Kopieranweisung in eingefügten Knoten legen

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Berechnung von Dominatoren

Berechnung von Dominatoren 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Muß bei Cytron et al. bei der SSA-Umformung gemacht werden
- War hier nicht nötig
- Dominatoren sind aber nach wie vor nützlich
- Wie sind sie hier berechenbar?
- Viel einfacher als im allgemeinen Fall!

Berechnung von Dominatoren 2

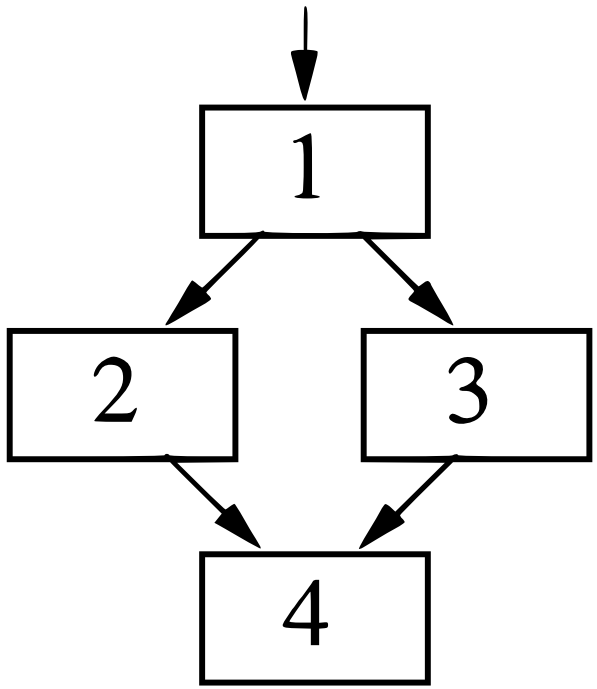
TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch hier Berechnung in einem Pass möglich
  - Über Quelltext oder AST
- Dominatorbaum
  - Vater eines Blocks ist dessen unmittelbarer Dominator IDOM
- Idee hier: Sub-CFGs der Konstrukte IF/WHILE/FOR/REPEAT/CASE
- . . . haben einen Eintrittspunkt und einen Austrittspunkt
- Der Eintrittspunkt dominiert alle Knoten des Konstrukts
- Unmittelbare Dominatoren können immer nach dem gleichen Schema bestimmt werden
- Dann Hochhangeln für weiter entferne Dominatoren

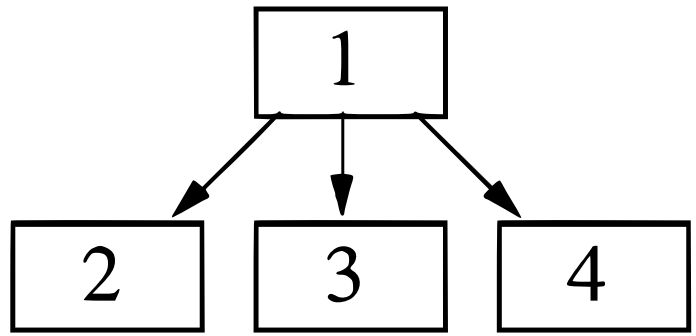
Berechnung von Dominatoren für IF, CASE

TECHNISCHE UNIVERSITÄT DARMSTADT

CFG



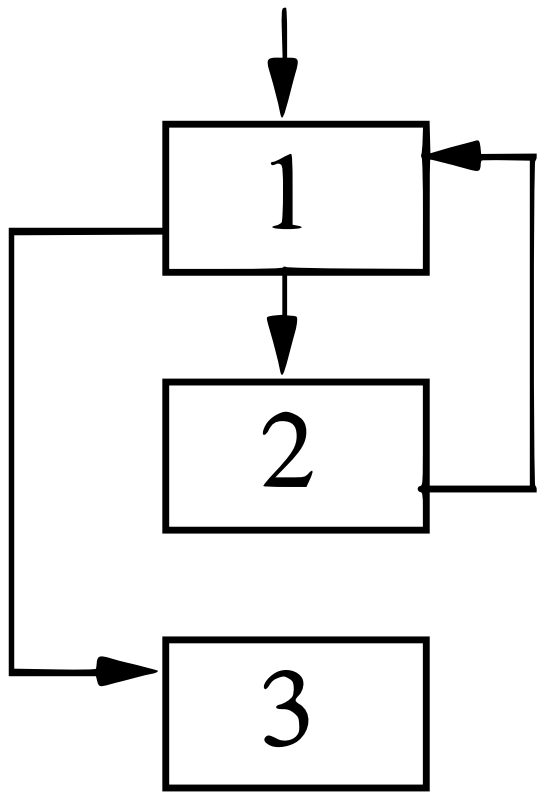
Dominatorbaum



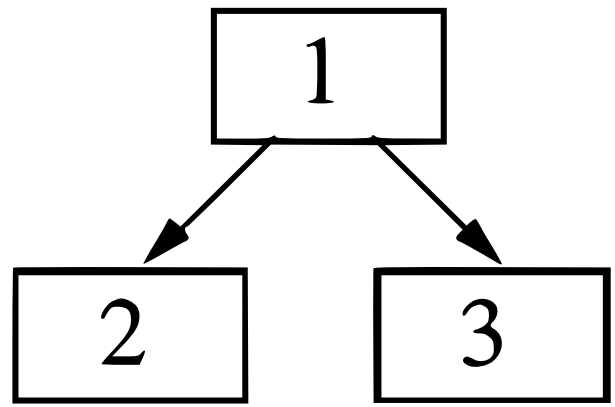
Berechnung von Dominatoren für WHILE, FOR

TECHNISCHE UNIVERSITÄT DARMSTADT

CFG



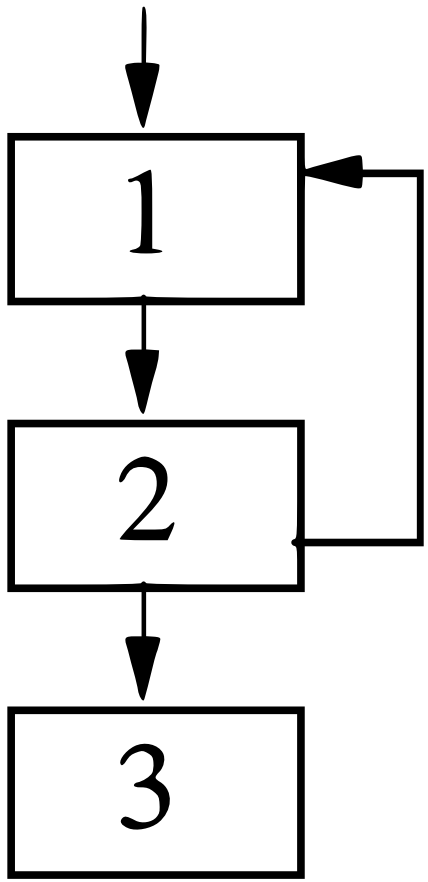
Dominatorbaum



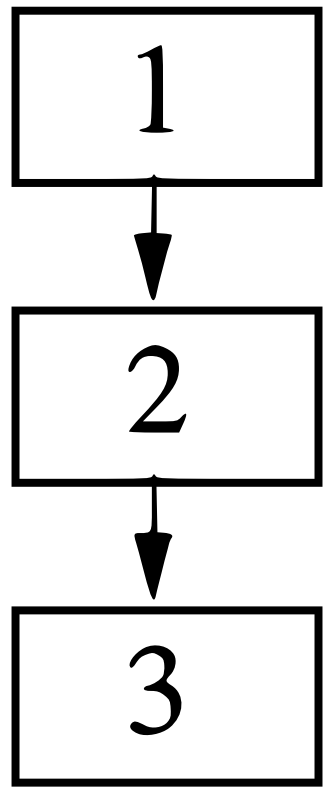
Berechnung von Dominatoren für REPEAT

TECHNISCHE UNIVERSITÄT DARMSTADT

CFG



Dominatorbaum



Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kontrollflussgraphen
- Versionsnummern für Variablen
- Aufbau der SSA-Form
- Transformation in SSA-Form
- Allgemeiner Fall (aus dem Orbit)
- Sonderfall: Strukturierte Programmiersprachen
- Rückwandlung aus der SSA-Form (einfaches Verfahren!)
- Berechnung von Dominatoren