Universität
Paderborn

Refactoring in eXtreme Programming

Cross the Rubicon:
Extract Method

Jacek Bandyk

Universität Paderborn

Method Extraction - Einführugsbeispiel

```
procedure readAndProcess(days:
   integer, var arr: int_array,
   var sum: integer, process:
   boolean)
var i: integer;
begin
  i := 0;
  sum := i;
  while i < days do begin
    inc(i);
    read(arr[i]);
  end;
  if process = True then begin
    for i := 1 to days do
      sum := sum + arr[i];
  end;
end;
```

- Eingabe und Verarbeitung in einer Funktion eng gekoppelt
- Ziel: Extraktion der Eingabefunktionalität in eigene Funktion
- Ohne automatisierte Unterstützung aufwendig und fehleranfällig

Universität Paderborn

Method Extraction - Einführugsbeispiel

```
procedure readAndProcess(days:
   integer, var arr: int_array,
   var sum: integer, process:
   boolean)
var i: integer;
begin
  readArr(days, arr);
  i := 0;
  sum := i;
  if process = True then begin
    for i := 1 to days do
      sum := sum + arr[i];
  end;
end;
```

```
procedure readArr(days:
   integer, var arr: int_array)
var i: integer;
begin
  i := 0;
  while i < days do begin
    inc(i);
    read(arr[i]);
  end;
end;
```

Universität
Paderborn

Agenda

- Ziele der Methodenextraktion
- Problemstellung und Lösungsansätze
- Program Slicing
- „Tucking“ statements into functions
- Anmerkungen
- Zusammenfassung

Universität Paderborn

Methodenextraktion - Ziele

- Automatische Extraktion ausgewählter Codefragmente in eine Funktion
- Ziele:
  - Konvertierung monolithischer Programme in modularisierte Programme
  - Konzeptionelle Modularisierung verbessert Verständnis
  - Code- Wiederverwendung; *Code scavenging* - „Plünderung“ brauchbarer Programmteile aus bestehender Software
  - Eliminierung von Code- Duplikaten

Universität Paderborn

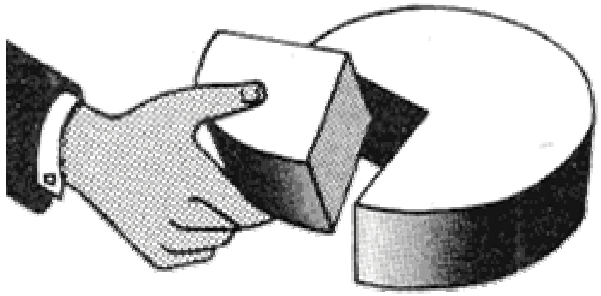
Methodenextraktion - Problemstellung

- Vollständige Identifikation (verstreuter) relevanter Codestellen
  => *Program Slicing*
- Extraktion der Methode selbst
  => *Tucking*
- Semantische Äquivalenz unentscheidbar
  => Nutzung semantikerhaltender Transformationen

Universität Paderborn

Program Slicing - Übersicht

- Notwendig zur automatischen Ergänzung berechnungsrelevanter Codestellen
- Basiert auf statischer Kontroll- und Datenflussanalyse

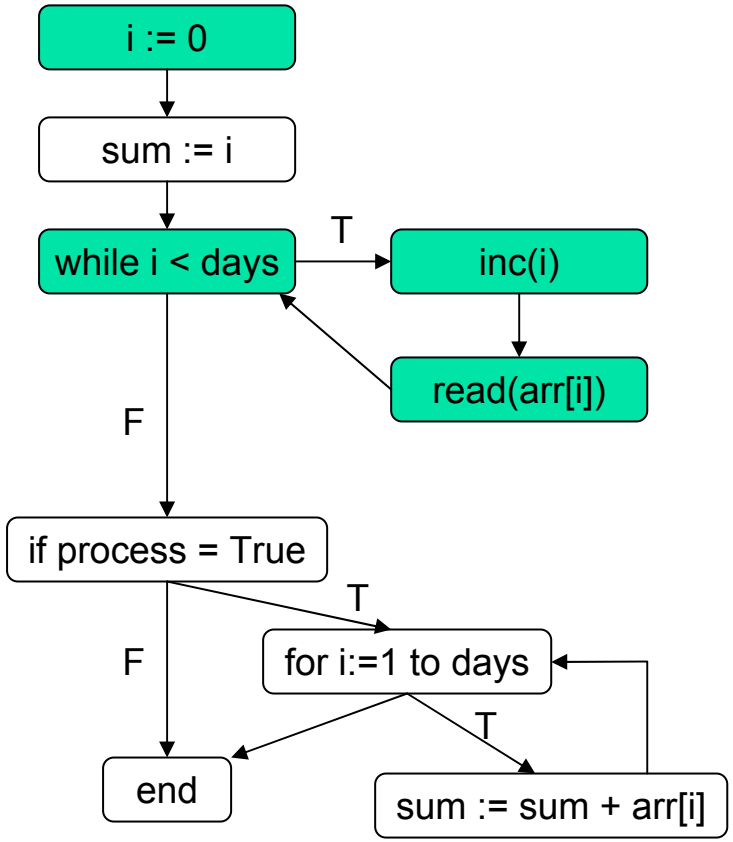


- Eingabe ist modifizierter CFG und *Slice Criterion* C=(Knoten, Variabelenmenge)

Universität Paderborn

Program Slicing - Beispiel

```
procedure readAndProcess(days:
   integer, var arr: int_array,
   var sum: integer, process:
   boolean)
var i: integer;
begin
  i := 0;
  sum := i;
  while i < days do begin
    inc(i);
    read(arr[i]);
  end;
  if process = True then begin
    for i := 1 to days do
      sum := sum + arr[i];
  end;
end;
```

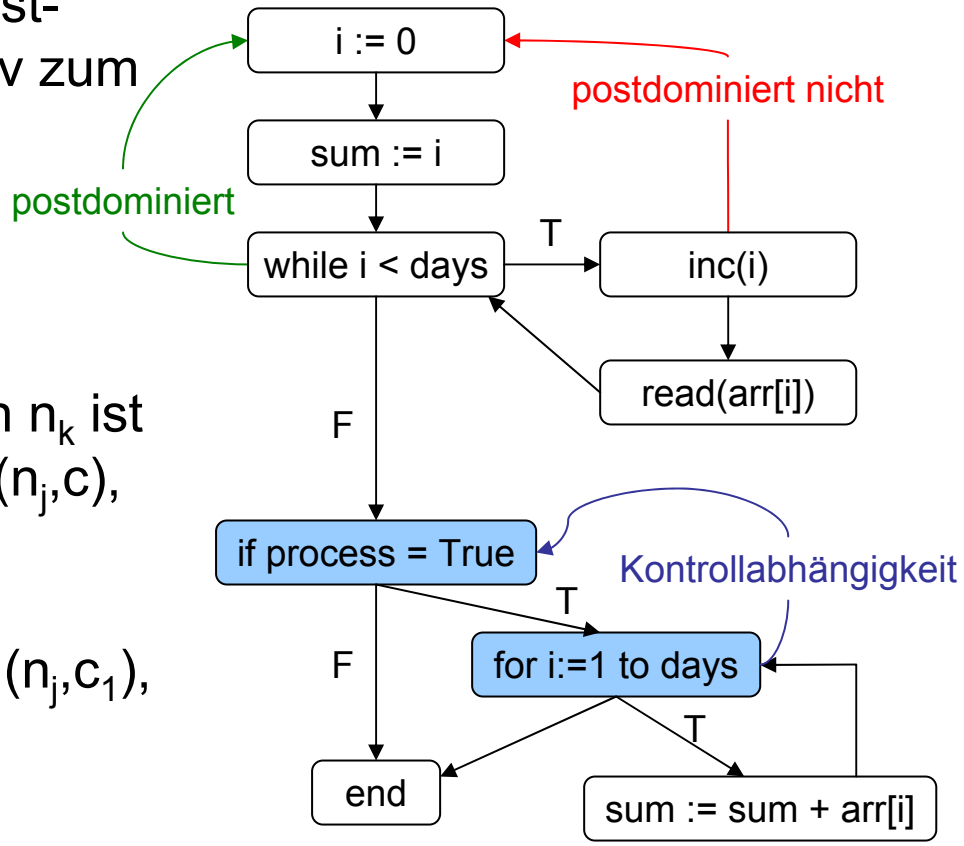


Universität Paderborn

Program Slicing - Kontrollabhängigkeit

- *Postdominator*: Knoten w post-dominiert v, wenn jeder von v zum Endknoten ausgehende Pfad w enthält (auch sich selbst)

- *Kontrollabhängigkeit*: Knoten n_k ist kontrollabhängig von Kante (n_j,c), wenn
  1. n_k postdominiert $Ziel(n_j,c)$
  2. es ex. eine weitere Kante (n_j,c_1), so dass n_k $Ziel(n_j,c_1)$ nicht postdominiert

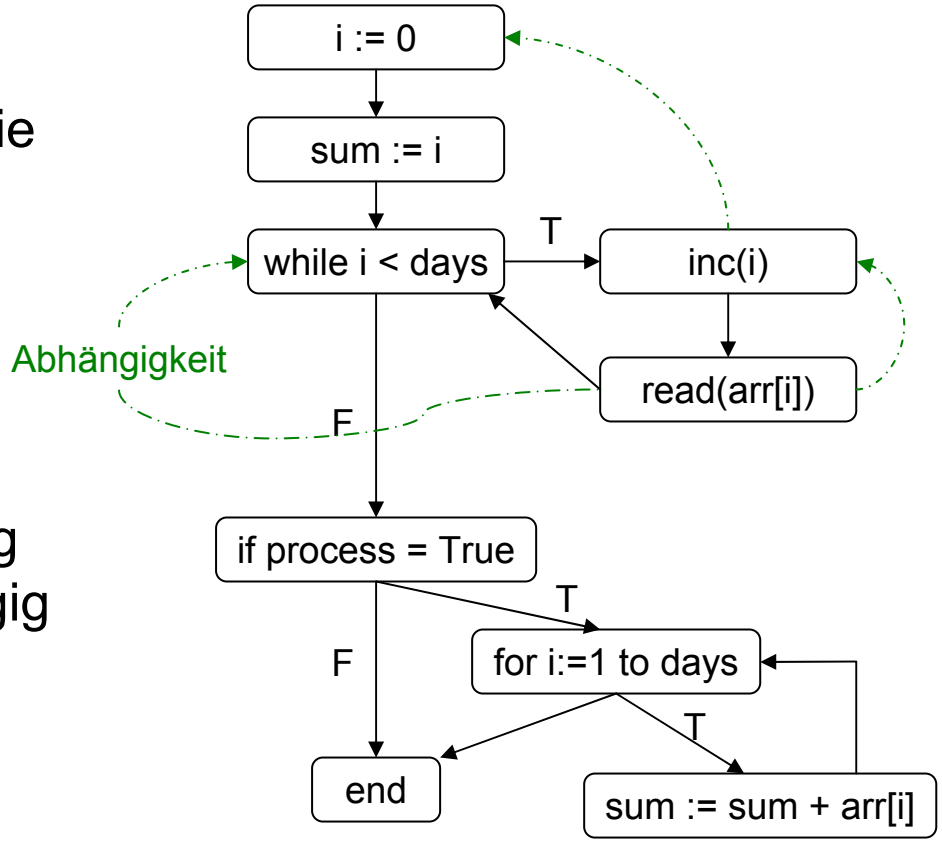


Universität Paderborn

Program Slicing – Abhängigkeit

- *Datenabhängigkeit*: n_k ist datenabhängig von n_j :
  1. n_k nutzt eine Variable v, die in n_j definiert ist
  2. es gibt einen Pfad von n_j nach n_k, auf dem v nicht definiert wird

- *Abhängigkeit*: n_k ist abhängig von n_j wenn es datenabhängig von n_j oder kontrollabhängig von (n_j, c) ist

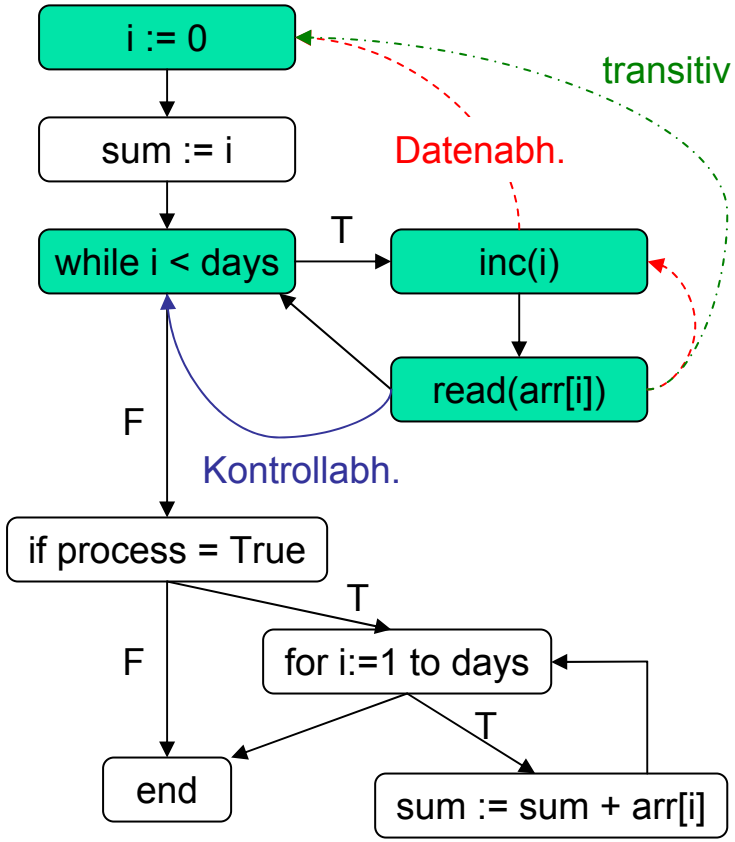


Universität Paderborn

Slice - Definition

- *Slice* für C=(Knoten n_k, Variablenmenge):

  Menge aller Knoten, von denen n_k für gegebene Variablen transitiv *abhängig* ist



Universität Paderborn

Program Slicing - Anmerkungen

- Explizite Angabe der Variabelenmenge im Slice- Kriterium
- Dadurch präzisere Angabe des Berechnungszieles
- Beispiel: *kursiv* gekennzeichnete Codefragmente sind Slice für C=(11, total)

```
1.  begin
2.    read(x, y);
3.    total := 0.0;
4.     sum := 0.0;
5.     if (x <= 1)
6.       then sum := y
7.       else begin
8.         read(z);
9.         total := x * y;
10.      end;
11.    write(total, sum);
12. end.
```

Universität
Paderborn

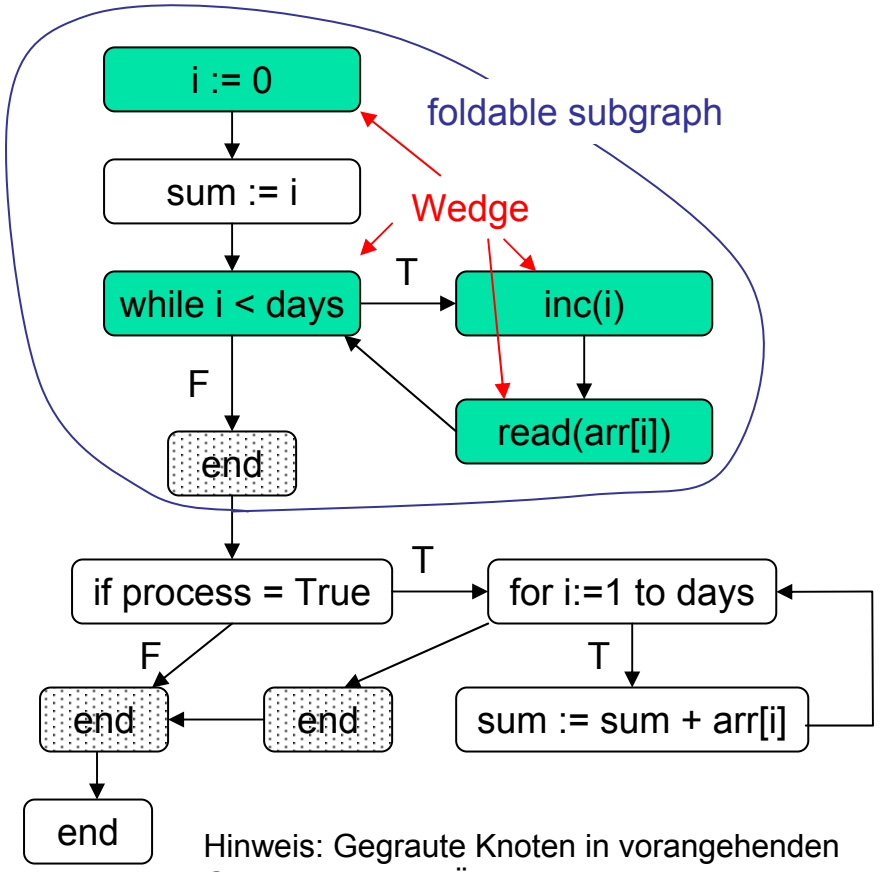
Tucking statements into functions

- *tuck*: to gather and fold
- 3 Schritte: *Wedge*, *Split* and *Fold*
- Beinhaltet Behandlung von Variablen
- Semantik bleibt nach der Operation erhalten
- Extraktion durch Graphsubstitution unter Beibehaltung der Abhängigkeiten

Universität Paderborn

Tucking – Wedge

- G_S ist *single entry single exit (SESE) subgraph* von G, wenn er jeweils nur einen Ein- und Ausgangsknoten besitzt
- G_S ist *foldable subgraph*, wenn alle Kanten vom Endknoten aus G_S führen, mit Ausnahme vom Startknoten
- *Wedge*(G_{Old}, G_S, S) = Slice(G_{Old} , S) ∩ N_S
- Angabe von G_S ermöglicht Steuerung des Beginns der Extraktion

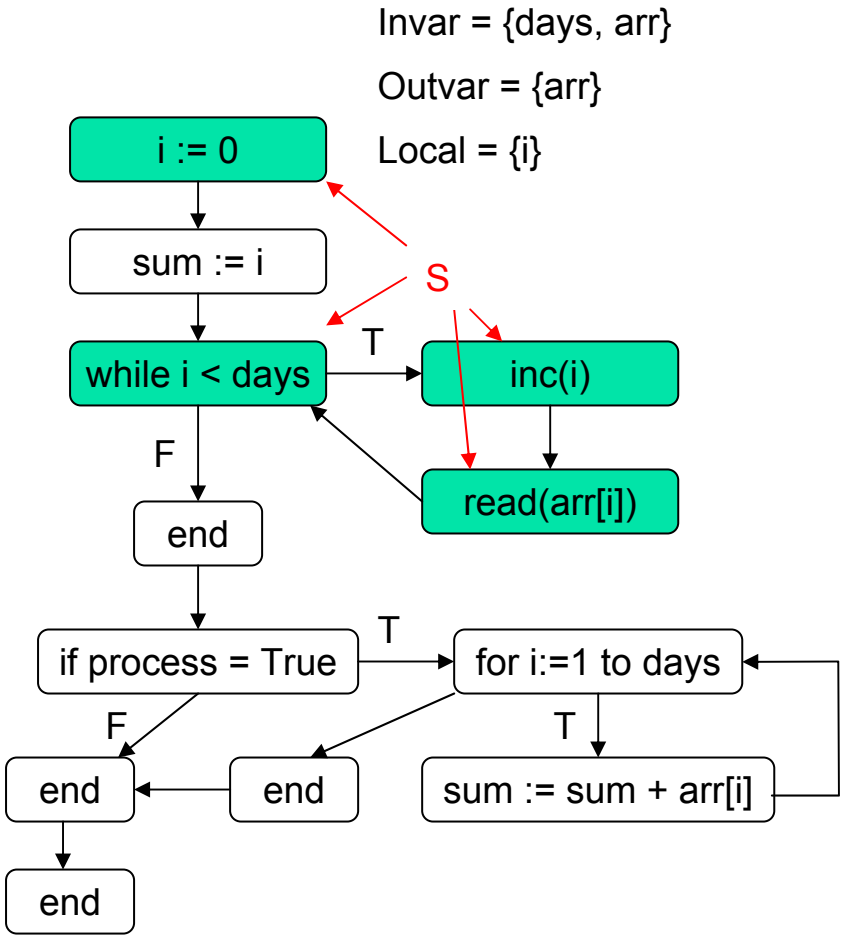


Hinweis: Gegraute Knoten in vorangehenden Graphen zwecks Übersicht nicht dargestellt.

Universität Paderborn

Tucking - Variablen

- IN?(G, S, v): v ist Eingabevariable in S, wenn v eine Definition außerhalb von S benutzt
- OUT?(G, S, v): v ist Ausgabevariable von S, wenn Definition von v in S außerhalb von S genutzt wird
- LOC?(G, S, v): v ist weder Ein- noch Ausgabevariable
- Invar(G, S), Outvar(G, S) und Local(G, S) sind entsprechende Variablenmengen

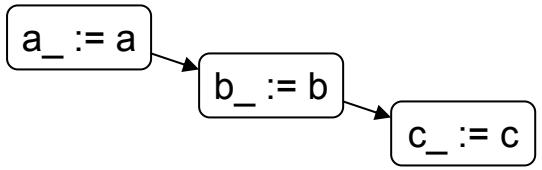


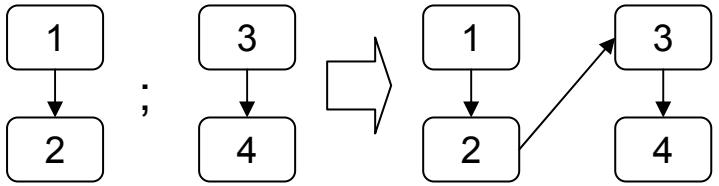
Universität Paderborn

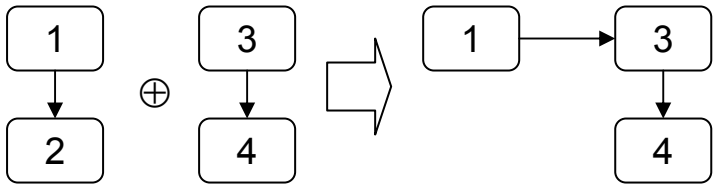
Tucking – Split, Graphdefinitionen

- [V]: Menge neuer, temporärer Variabelennamen v‘ zu jeden v in V
- I[V]: Neuer Teilgraph mit Zuweisungen der Form v‘ := v für alle v in V
- $G_1;G_2$: Sequentielle Graphkomposition durch hinzufügen von Kanten
- G1 ⊕ G2: Sequentielle Graphkomposition durch ersetzen des Endknotens

V = {a, b, c}
[V] = {a_, b_, c_}







Universität Paderborn

Tucking – Split

Split(G_{old}, G_S, S) = G_{new}:

X = Wedge(G_{old}, G_S, S)

X‘ = N_S – Exitnode(G_S) – X

Y = Wedge(G_{old}, G_S, X‘)

Y‘ = N_S – Exitnode(G_S) – Y

if Outvar(G_{old}, X) ∩ Outvar(G_{old}, Y) ≠ ø then

conflict

else

G_X = G_S / X‘

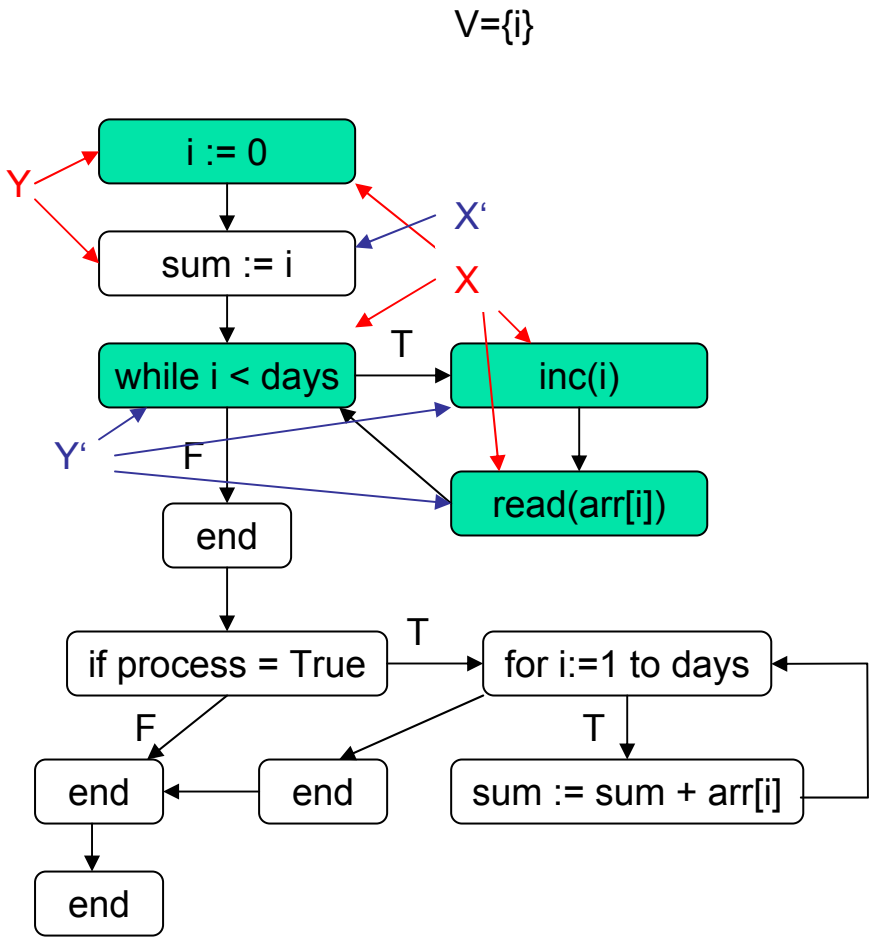
G_Y = G_S / Y‘

V = Local(G_{old}, Y) ∪

(Invar(G_{old}, Y) - Outvar(G_{old}, Y))

G_{XY} = I[V]; G_X ⊕ [V]G_Y

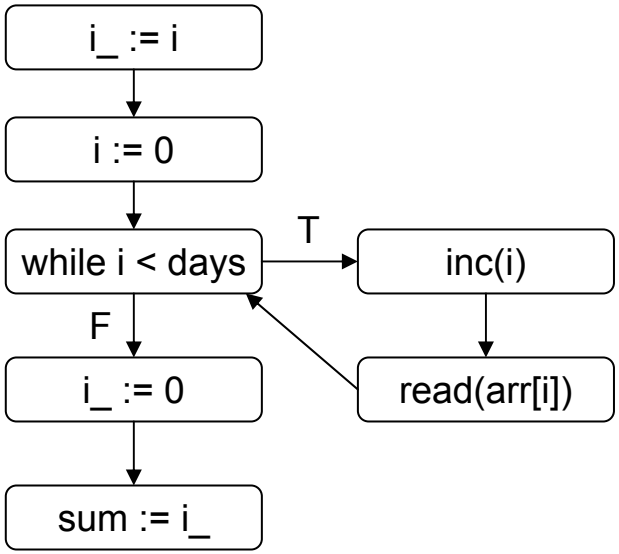
G_{new} = [G_{XY} / G_S] G_{old}



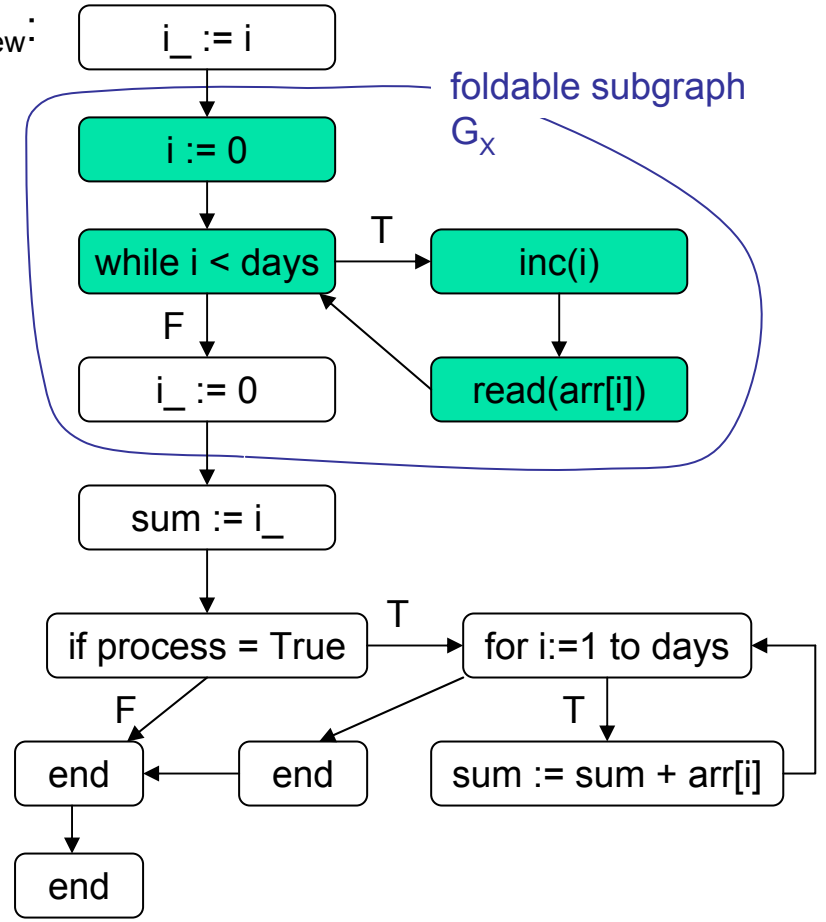
Universität Paderborn

Tucking – Split, Fortsetzung

$G_{XY} = I[V]; G_X \oplus [V]G_Y$:



G_{new}:



Universität Paderborn

Tucking - Fold

- *Split* liefert G_{new}; definiert zusätzlich G_X
- *Fold*: Ersetze G_X in G_{new} durch einen Aufruf *call(G_2, parameter);* G_2 ist G_X mit neuen Start- und Endknoten
- *Parameter* der neuen Funktion*:*
  - Ref(G_2) = Outvar(G_{new}, G_X)
  - Val(G_2) = Invar(G_{new}, G_X) / Outvar(G_{new}, G_X)
  - Local(G_2) = Local(G_{new}, G_X)

Universität Paderborn

Tucking – Beispielextraktion

1. Kursive Codezeilen werden zur Extraktion lokalisiert (*Wedge*)
2. Restrukturierung mit Hilfe von temporären Variabelenzuweisungen (*Split*)
3. Bestimmung der Parameter; Aufruf der neuen Funktion (*Fold*)

```
procedure readAndProcess(days:
   integer, var arr: int_array,
   var sum: integer, process:
   boolean)
var i: integer;
begin
  i := 0;
  sum := i;
  while i < days do begin
    inc(i);
    read(arr[i]);
  end;
  if process = True then begin
    for i := 1 to days do
      sum := sum + arr[i];
  end;
end;
```

Universität Paderborn

Tucking – Beispielextraktion

```
procedure readAndProcess(days:
   integer, var arr: int_array,
   var sum: integer, process:
   boolean)
var i: integer;
begin
  readArr(days, arr);
  i := 0;
  sum := i;
  if process = True then begin
    for i := 1 to days do
      sum := sum + arr[i];
  end;
end;
```

```
procedure readArr(days:
   integer, var arr: int_array)
var i: integer;
begin
  i := 0;
  while i < days do begin
    inc(i);
    read(arr[i]);
  end;
end;
```

Universität Paderborn

Tucking - Bemerkungen

- Variabelenumbenennung temporär; wird nach *Fold*- Schritt zurückgesetzt
- Bei komplexen Abhängigkeiten Extraktionsmöglichkeiten nur eingeschränkt
- Eingabe von G_S ist Sache des Benutzers; hier Verfahren/ Werkzeuge zur Identifikation und Auswahl von faltbaren Teilgraphen nötig

Universität Paderborn

Semantikerhaltende Methodenextraktion

- Vorgestelltes Tucking- Verfahren kann komplexe Abhängigkeiten nicht auflösen
- Erweitertes Verfahren von Raghavan Komondoor und Susan Horwitz
- Nutzt Abhängigkeits- Polygraphen zur semantikerhaltenden Restrukturierung
- Ordnungsregeln sichern Semantikerhalt

Universität Paderborn

Anmerkungen – benötigte Vorbehandlung

- Statische Analyse vor der Extraktion liefert Nutzungs- und Definitionsinformationen
- Eliminierung nichtinitialisierter Variablen
- Eliminierung toter Variabelendefinitionen
- Nutzungsprüfung für Zeiger bei interprozeduralen Zugriffen
- Behandlung sprachspezifischer Konstrukte

Universität Paderborn

Zusammenfassung

- Methodenextraktion praktisch oft benötigt und daher sehr nützlich
- Automatisierte Verfahren basieren auf statischer Datenflussanalyse
- Näherungslösungen für das Problem semantischen Äquivalenz
- Implementierungen vorhanden; z.B. Xrefactory für java und C

Universität Paderborn

Literaturangaben

- *Restructuring prgrams by tucking statements into functions*, A. Lakhotia and J. Deprez
- *Extracting Reusable Function by flow Graph- Based Program Slicing*, F. Lanubile and G. Visaggio
- *Semantics- Preserving Procedure Extraction*, R. Kommondoor and S. Horwitz
- *Xrefactory*, Refactoring Browser for Emacs, XEmacs and jEdit
  Verfügbar unter: http://www.xref-tech.com/

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?