Technische Fakultät
Universität Bielefeld

Vorlesung

Softwaretest und -optimierung

Version 2012

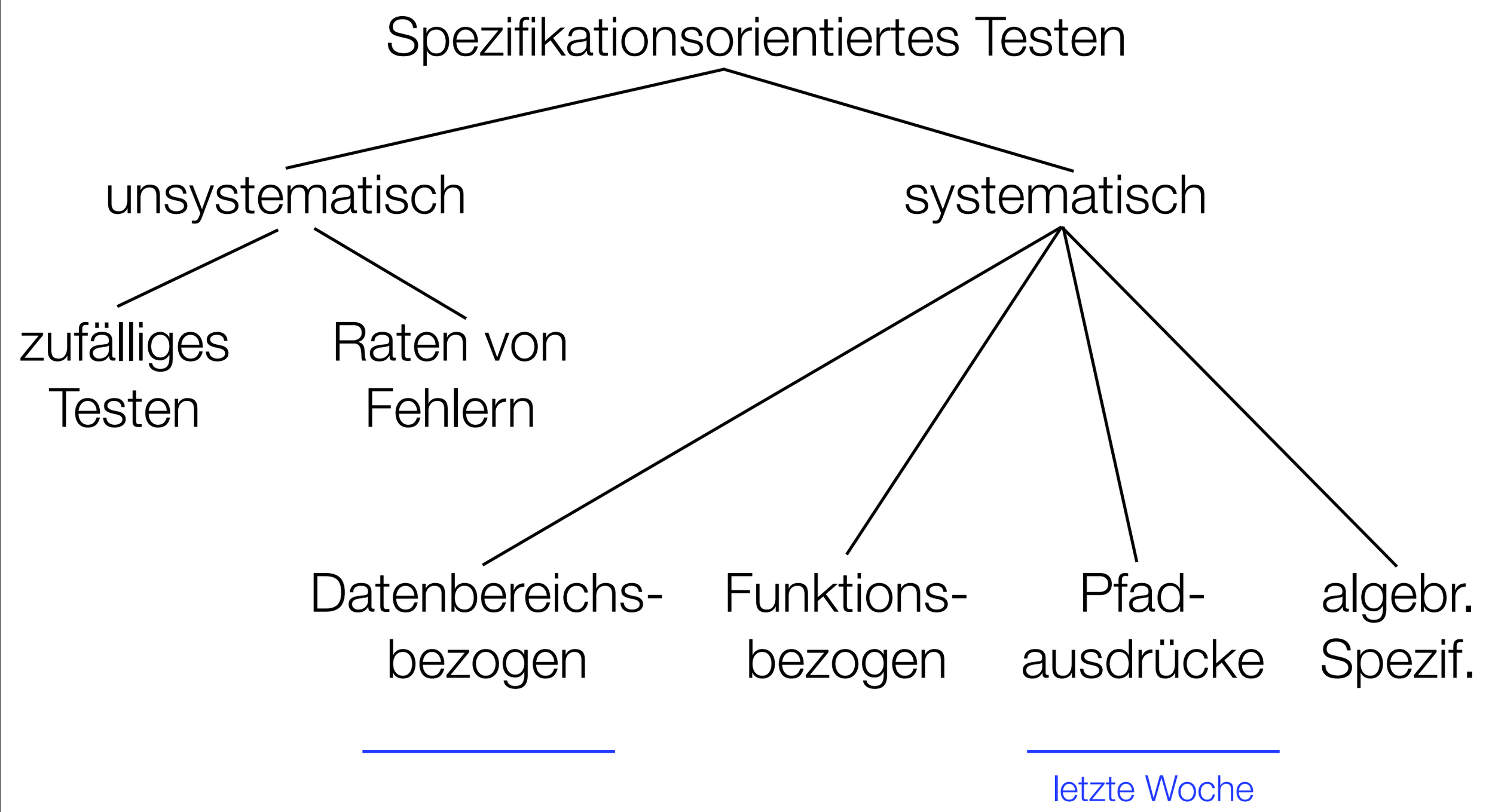
Dr. Carsten Gnörlich

Rechnerbetriebsgruppe

Kap. 4 - Strukturorientiertes Testen

(= Kap. 7 und 9.2 aus Riedemann)

Spezifikationsorientiertes Testen



Grenzen des spezifikationsorientierten Testens

Stichproben aus Äquivalenzklassen bieten gute Fehleraufdeckung

→ wenn Programm als äq. betrachtete Testdaten auch äquivalent behandelt

→ wenn Spezifikation richtig umgesetzt wurde

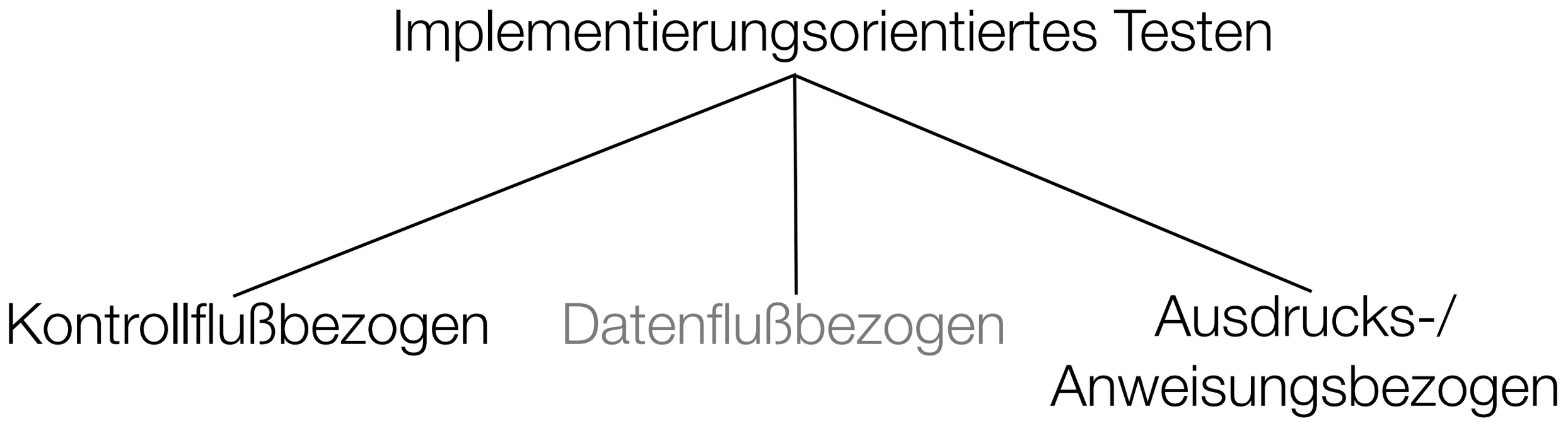
Beispiel: absichtliche Defekte werden nur zufällig aufgedeckt:

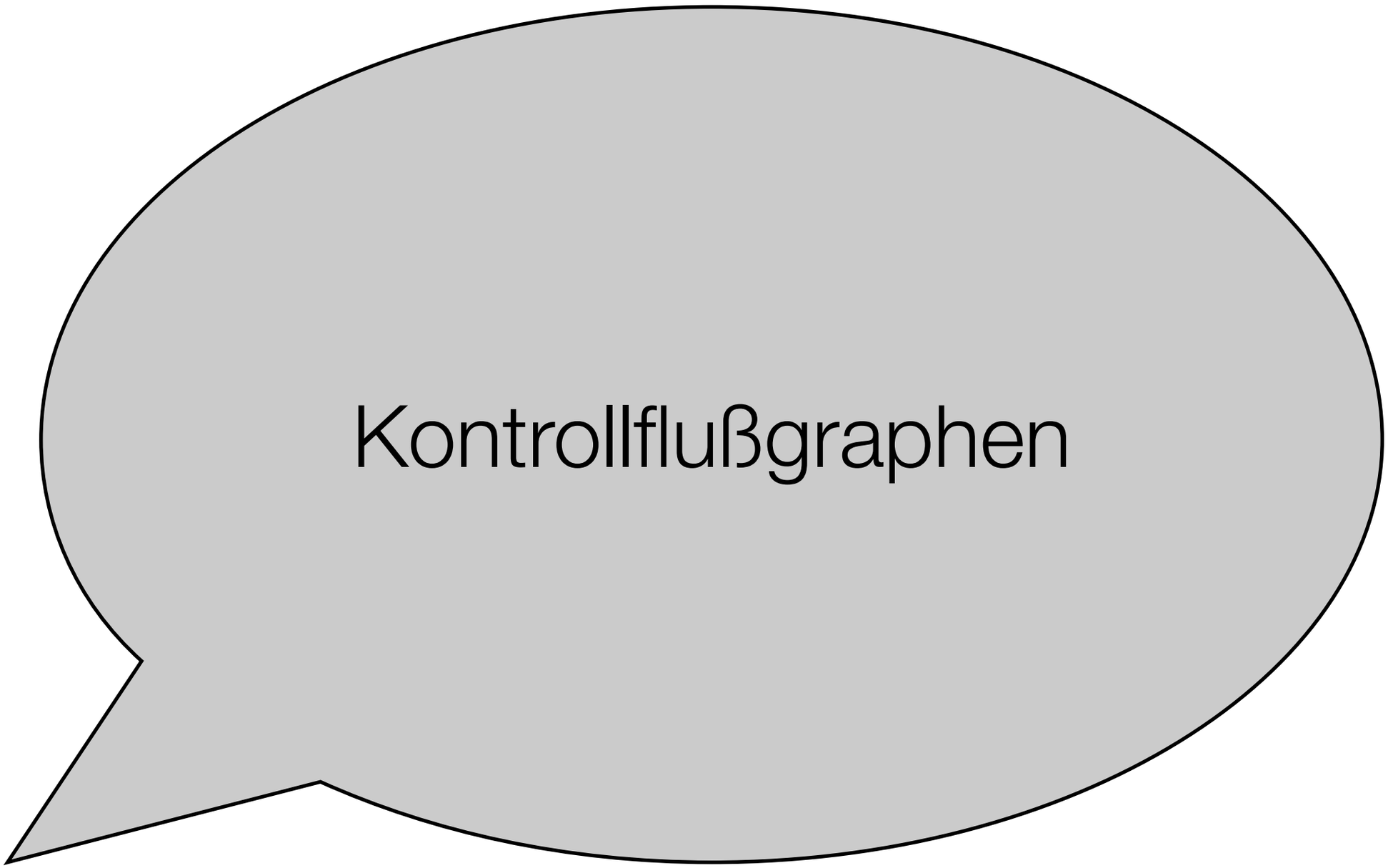
```
if trans_id == 5110 then überweise_geld_in_die_schweiz
```

Implementierungsorientiertes Testen

Grundidee:

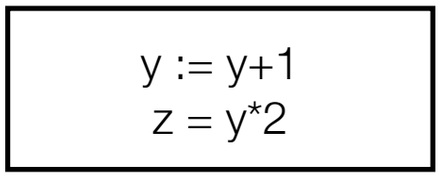
- Betrachten der Programmstruktur
- Annahme daß jede Programmanweisung defekt sein kann



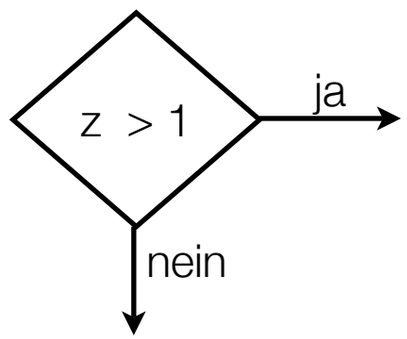

Kontrollflußgraphen

Kontrollflußgraphen

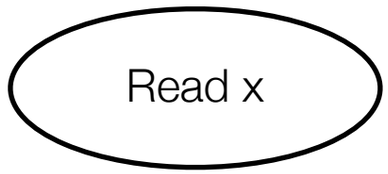
- gerichteter Graph
- Knoten
  - Folge von Anweisungen *ohne* Entscheidungen



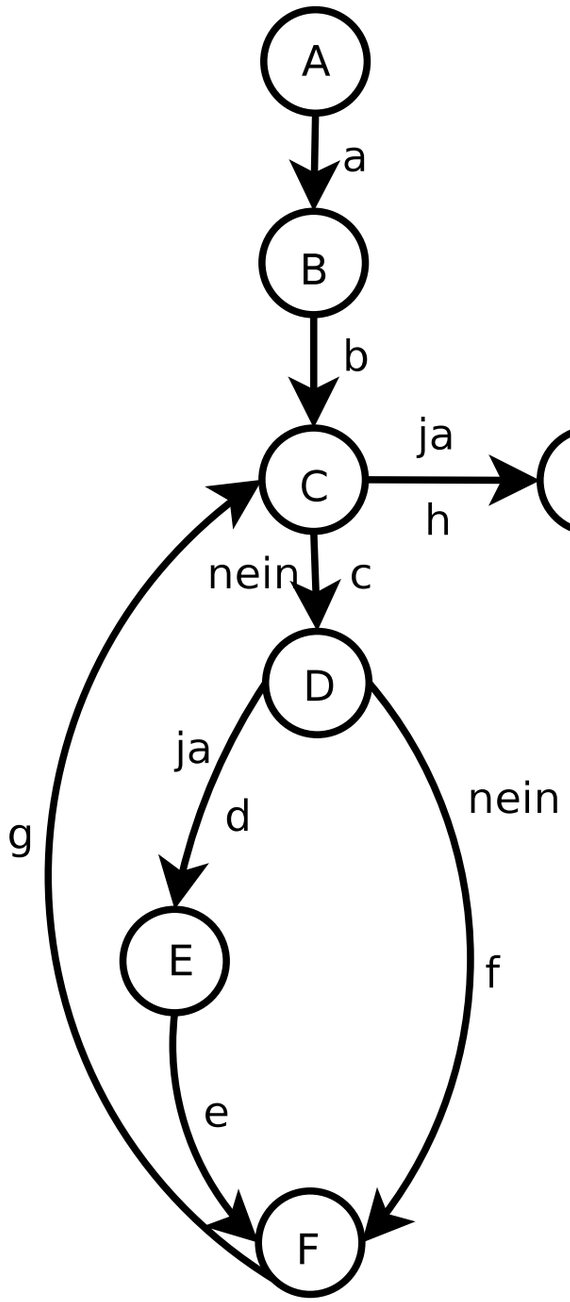
  - Entscheidungsprädikate



- Kanten: Kontrollfluß (bei Entscheidungen beschriftet)
- Konvention: Genau ein Anfangs- und Endknoten

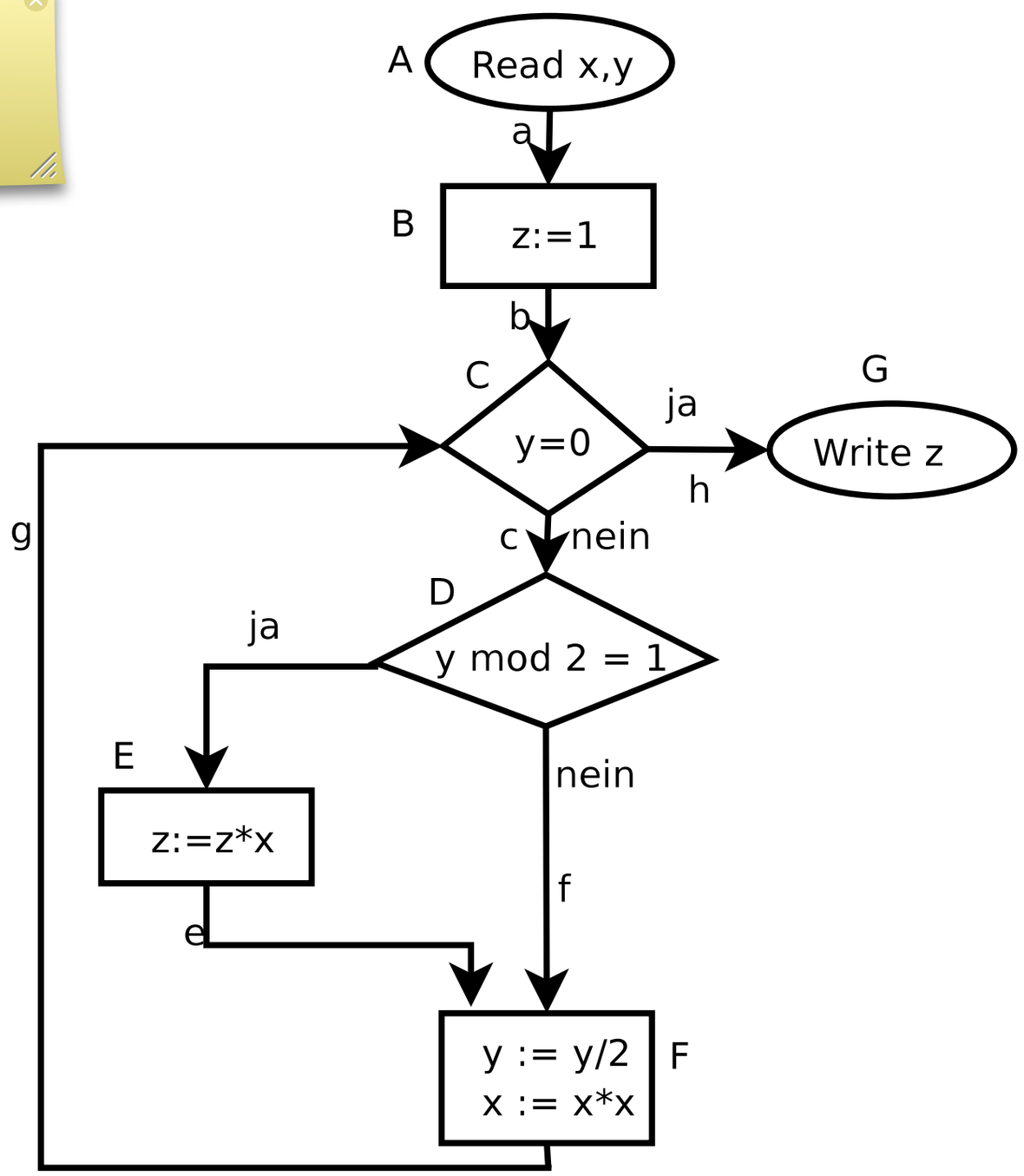


Kontrollflußschema



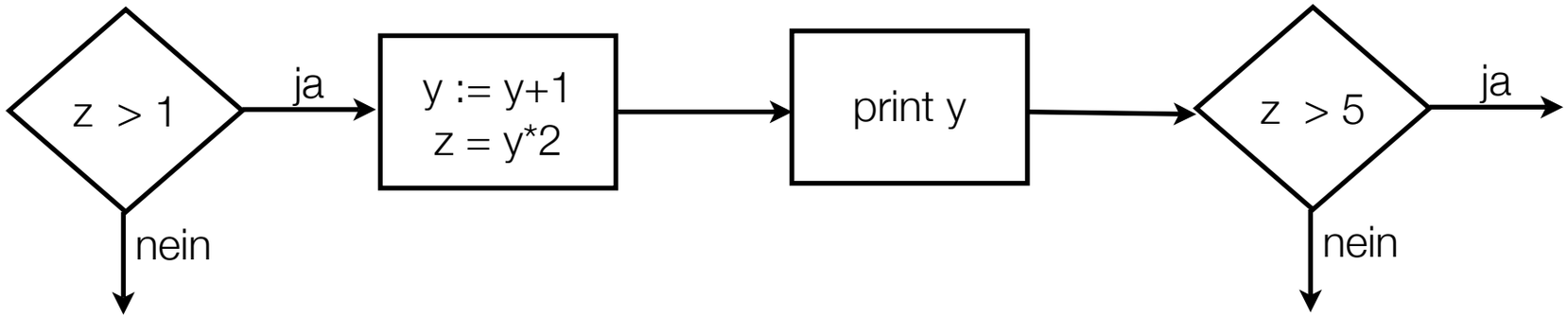
Schema: Abstraktion von konkreten Anweisungen/ Entscheidungen

Kontrollflußgraph



Entscheidungs-Entscheidungsweg (EE-Weg)

- beginnt beim Anfangsknoten oder Entscheidungsknoten
- endet im nächsten Entscheidungsknoten oder Endknoten



Beobachtung: Wird die erste Kante eines EE-Weges ausgeführt,
dann werden alle Kanten des EE-Weges ausgeführt.

Segmente und Vereinigungsknoten

Vereinigungsknoten

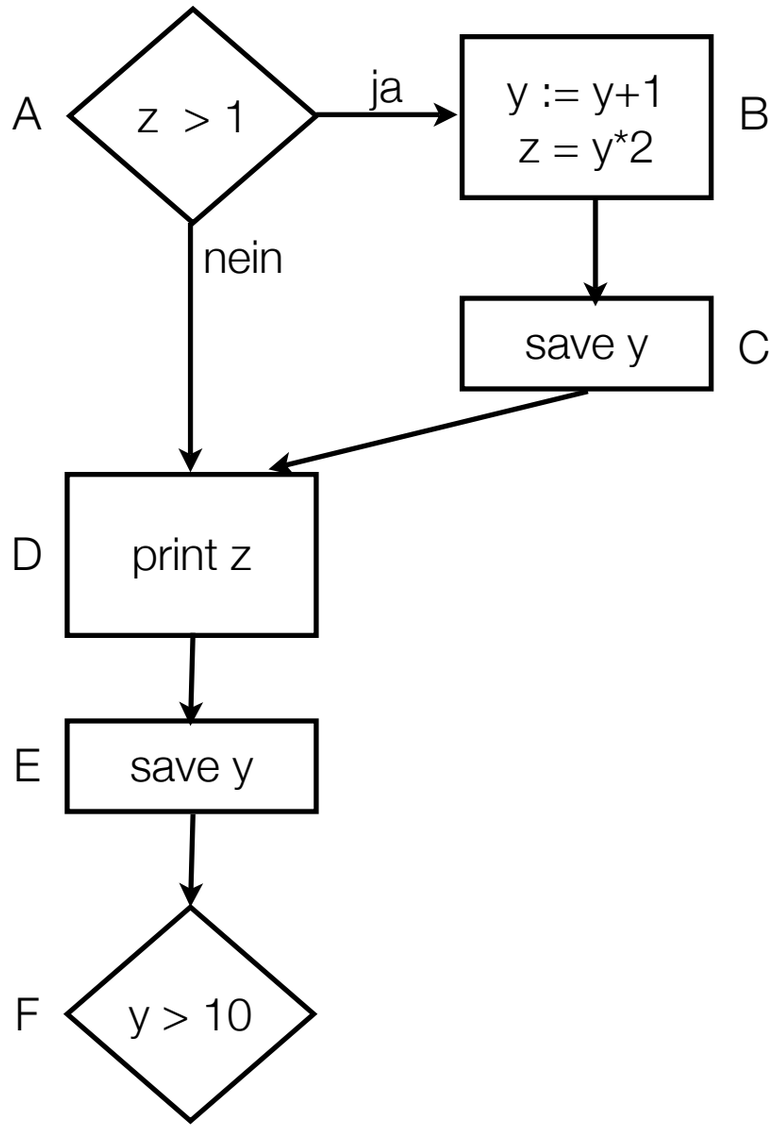
= Überlappung von EE-Wegen

Segment

= EE-Wegteil mit genau einem Eingang
und Ausgang in den inneren Knoten

Beispiel:

ADEF : EE-Weg
D : Vereinigungsknoten
A, DEF : Segmente



Wege im Kontrollflußgraphen

- Zyklus = Weg über mindestens zwei Knoten der an demselben Knoten beginnt und endet.
- einfacher Zyklus = alle Knoten außer Anfang=Ende sind verschieden
- vollständiger Weg
  - Weg vom Anfangs- zum Endknoten
  - unendlich langer Weg beginnend beim Anfangsknoten

- Für Menge T von Testdaten ist Wege(T) die Menge der vollständigen endlichen Wege, die durch die Testdaten $t \in T$ ausgeführt werden.
- $Wege_\infty(T)$: auch unendliche Wege sind zugelassen

Kontrollflußbezogenes Testen
- Anweisungsüberdeckung

Vorgehensweise beim kontrollflußbez. Testen

- Auswahlkriterium für Kontrollflußwege festlegen (welche, wie oft)
- Testwirksamkeitsmaß (TWM) berechnen:
  - Prozentsatz der Weg(stücke), für die zu einer gegebenen Testmenge das obige Auswahlkriterium erfüllt worden ist.
- typische Fehlerarten:
  - Berechnungsfehler: richtiger Weg, aber falsche Zuweisung
  - Bereichsfehler: falscher Weg durch fehlerhafte Abfrage
  - Unterbereichsfehler: ein Kontrollflußweg zu viel oder zu wenig;

    d.h. eine Abfrage fehlt oder ist zu viel

Anweisungsüberdeckung

- Ann.: Jede Anweisung im Programm kann fehlerhaft sein

C_0-Überdeckung

Wähle eine Testdatenmenge T,

bei der jede Anweisung mindestens einmal ausgeführt wird.

Testwirksamkeitsmaß

$$TWM_0 = \frac{\text{Anzahl mindestens unter T ausgeführter Anweisungen}}{\text{Anzahl aller Anweisungen im Programm}}$$

Durchführen der Strategie

- TWM_0 z.B. durch Instrumentierung mit Zählern ermitteln

  (nutze Eigenschaften des EE-Weges!)

1. Teste das Programm mit allen $t \in T$ und ermittle TWM_0

2. Solange $TWM_0 < 1$, weitere Testfälle hinzunehmen

Eigenschaften der Strategie

a) $TWM_0 = 100\%$ nicht immer erreichbar

- Entscheidbarkeitsproblem; in der Praxis nur 90% Abdeckung erreichbar

→ besser: *begründen*, warum Teile nicht erreichbar sind

→ noch besser: Programm umstrukturieren für 100% Abdeckung

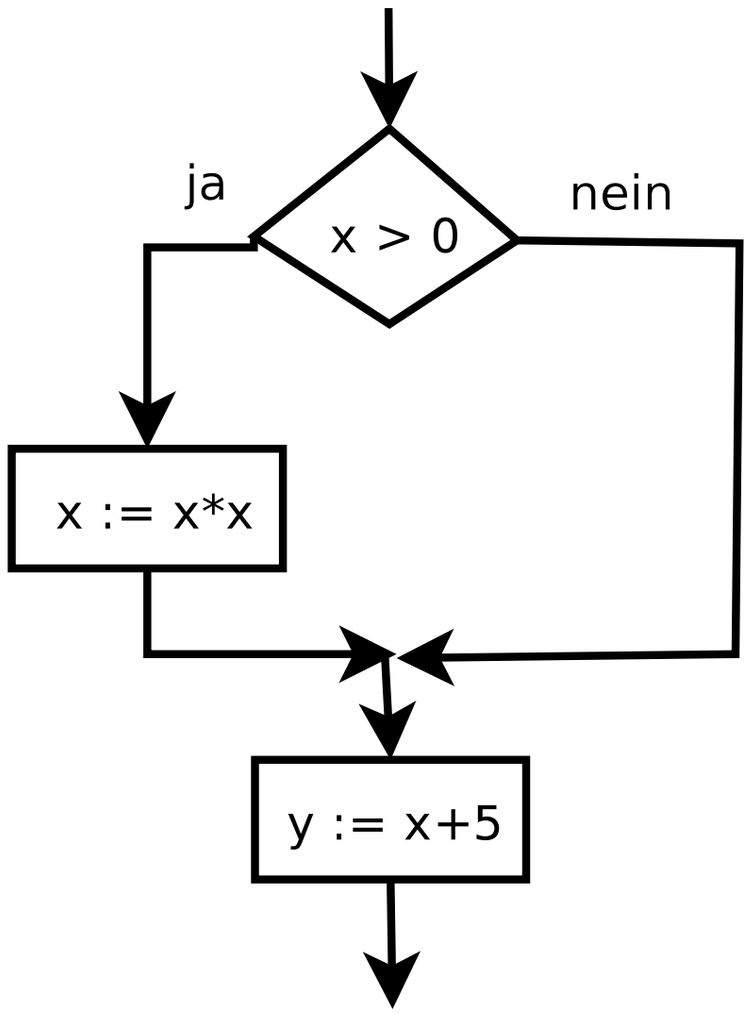
b) Auswahl der Testfälle (ÄK, Grenzwerte etc.) ?

c) Abdeckung aller Anweisungen ist notwendig, aber nicht hinreichend

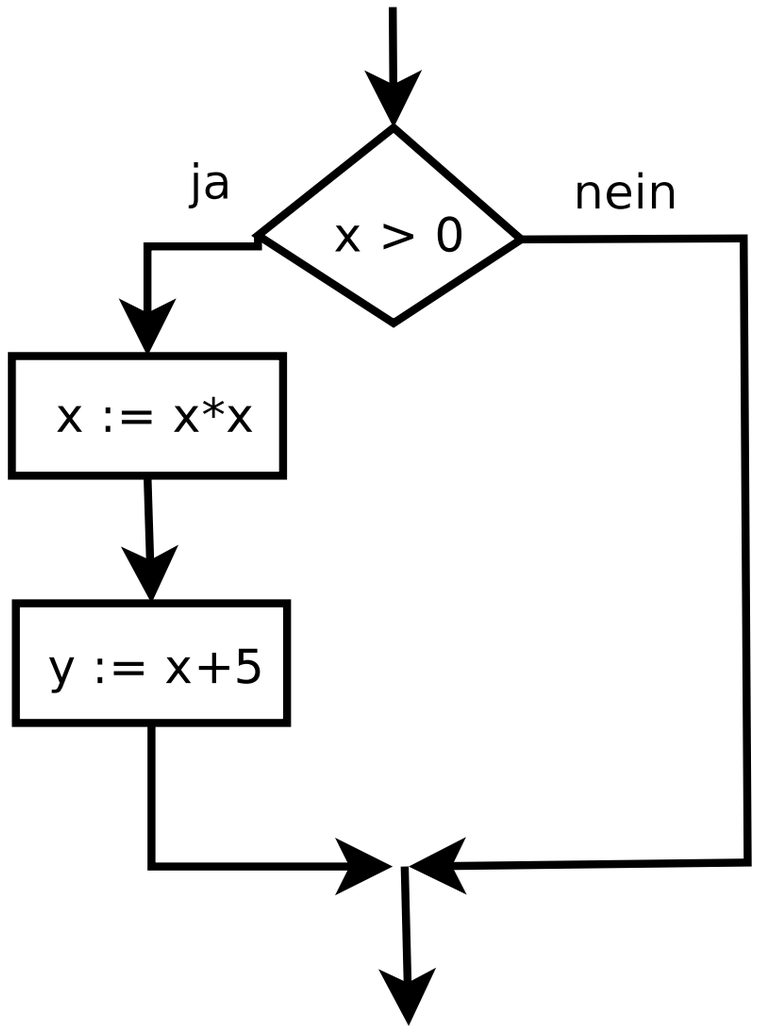
→ Kontrollflußfehler werden nicht erkannt (s. nächste Folie)

Kontrollflußfehler - Beispiel

Richtiges Programm



Falsches Programm



für x=5: $TWM_0 = 100\%$

Kontrollflußbezogenes Testen
- Zweigüberdeckung

Anweisungsüberdeckung

C_1-Überdeckung

Wähle eine Testdatenmenge T,

bei der jeder Zweig im Programm mindestens einmal ausgeführt wird.

Testwirksamkeitsmaß

$$TWM_1 = \frac{\text{Anzahl mindestens einmal ausgeführter Zweige unter T}}{\text{Anzahl aller Zweige im Programm}}$$

Satz 7.2.1 (und Folgerungen daraus)

Testmenge T erfüllt C_1-Überdeckung

$\Leftrightarrow$ jedes Segment wird unter T ausgeführt

$\Leftrightarrow$ jeder EE-Weg wird unter T ausgeführt

$\Rightarrow$ TWM_0 erfüllt

$\Rightarrow$ bessere Vorgaben zum Generieren der Testfälle

Bemerkung:

C_1-Überdeckung ist ebenfalls notwendig aber nicht hinreichend

$\rightarrow$ siehe die nachfolgenden Kriterien

Kontrollflußbezogenes Testen
- Kombination von Segmenten

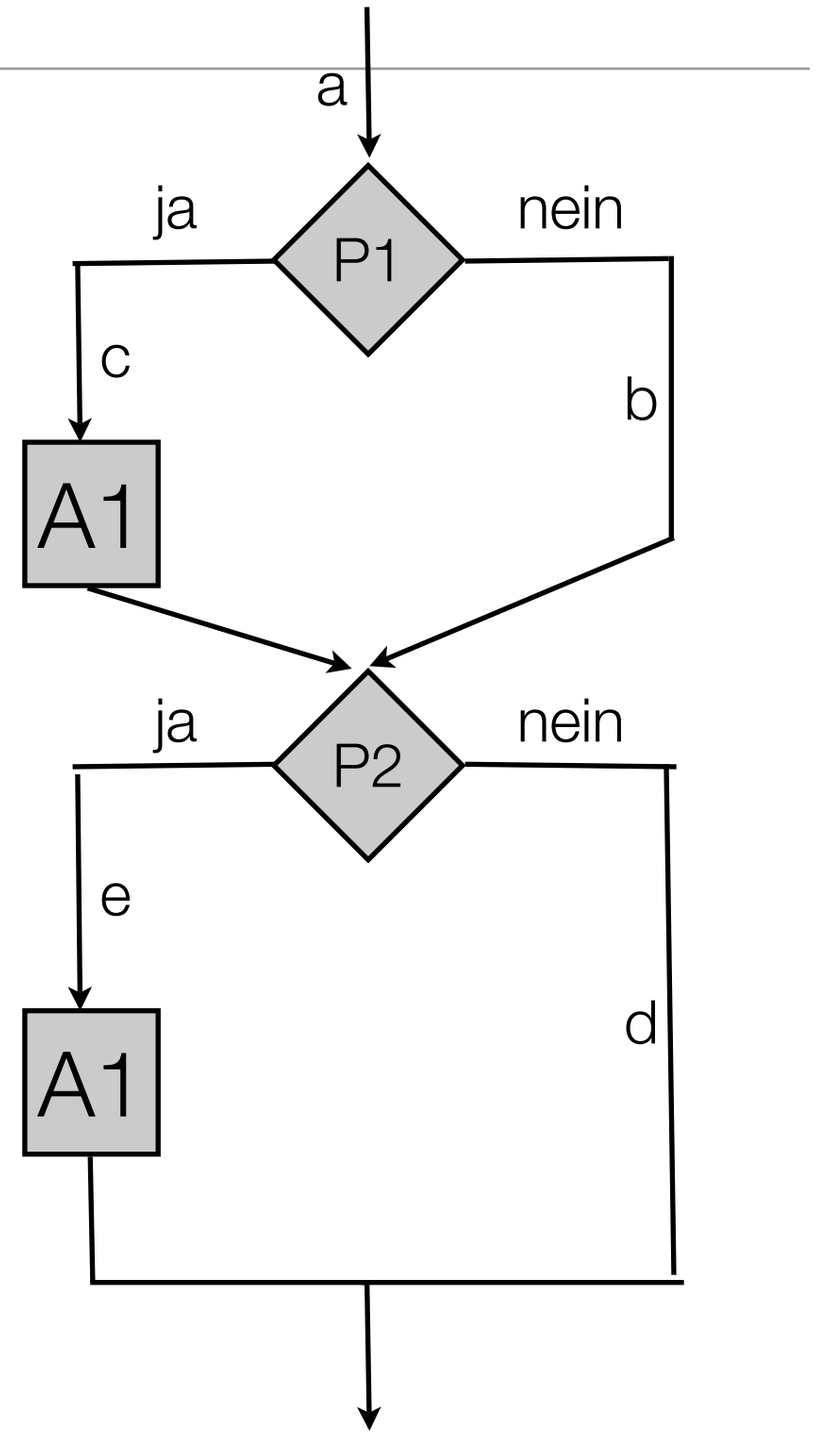
Motivation:

- für C_1-Überdeckung reicht aus:
  - t_1: P1=false, P2=true
  - t_2: P1=true, P2=false

➡ Segment-Reihenfolgen ce bzw. bd werden nicht getestet

➡ wir haben schon gesehen daß auch dies wichtig ist

(Anweisung kann in falschem Zweig liegen)



Pfadüberdeckung

C_p-Überdeckung

Wähle eine Testdatenmenge T,

bei der jeder Weg im Programm mindestens einmal ausgeführt wird.

Probleme:

- unendlich viele Wege bei *Schleifen* ohne Iterations-Obergrenze
- bei k aufeinanderfolgenden *Verzweigungen*: 2^k Möglichkeiten

besser handhabbar: Segmentpaarüberdeckung

C_{sp}-Überdeckung

Wähle eine Testdatenmenge T,

bei der jedes Paar von EE-Wegen p und q,
die im Kontrollflußgraph direkt aufeinander folgen,
mindestens einmal ausgeführt wird.

$C_s(n)$-Überdeckung (Verallgemeinerung)

... bei der jede Folge von n EE-Wegen $S_1,...,S_n$,
die im Kontrollfluß direkt aufeinander folgen,
mindestens einmal ausgeführt wird.

Kontrollflußbezogenes Testen
- Kriterien zur Schleifenabdeckung

Allgemeinfall: unendlich viele Wege durch Schleifen

→ formale Verifikation

oder

→ Bildung von endlich vielen Äquivalenzklassen / repräsentativen Wegen

Grenze-Inneres-Methode

Klasse von C_{GI}-Überdeckungen

Wähle eine Testfallmenge T mit folgenden Eigenschaften:

- T erfüllt das C_0-Kriterium (Anweisungsüberdeckung)

  und jede Schleife wird getestet für

- k = 0 Iterationen (Grenzfall: keine Schleifenausführung)
- k = 1 Iteration (Grenzfall: genau eine Iteration)
- k ≥ 2 Iterationen (Inneres; für weitere aber endlich viele k)

→ Aufwandsproblematik erfordert Abschwächung des Kriteriums

k-Äquivalenz von Wegen

Zwei Wege im Kontrollflußgraphen heißen k-äquivalent, wenn sie

- schleifenfrei und identisch sind, oder wenn sie
- die Schleife wie folgt durchlaufen:

1. weniger als k-mal und sie sind identisch / für innere Schleifen k-äquiv.

2. mindestens k-mal, wobei

   (a) die ersten k-1 Durchläufe identisch / für innere Schleifen k-äq. sind

   (b) der k-te Durchlauf identisch / für innere Schleifen k-äq ist, aber die Kante zum Schleifenanfang fehlt wenn es genau k Iterationen gibt

   (c) weitere Durchläufe (k+1, k+2, ...) beliebig sind oder fehlen.

strukturierte Pfadüberdeckung

$C_i(k)$-Überdeckung

Wähle eine Testdatenmenge T,

die alle Wege mit bis zu k Iterationen enthält.

Formal: Wege(T) enthält mindestens einen Weg
aus jeder Klasse von k-äquivalenten Wegen.

Komplexität: wird schnell sehr hoch

→ kleine k verwenden, z.B. k=2, oder

→ Kriterium der k-Äquivalenz noch weiter abschwächen

Schwache 2-Äquivalenz

Zwei *vollständige* Wege im Kontrollflußgraphen heißen schwach 2-äquivalent, wenn sie:

- schleifenfrei und identisch sind, oder wenn sie
- die Schleife wie folgt durchlaufen:

1. höchstens einmal
   und sie sind identisch / in inneren Schleifen schwach 2-äquivalent,

2. mindestens zweimal, wobei der erste Durchlauf identisch oder
   in inneren Schleifen schwach 2-äquivalent ist

Schwächere Überdeckungskriterien

starke C_{GI}-Überdeckung

Wähle eine Testfallmenge T, die $C_i(2)$-Überdeckung erfüllt

(→ alle Iterationen bis zu einer Tiefe 2)

C_{GI}-Überdeckung

Wähle eine Testfallmenge T so daß Wege(T) mindestens einen Weg aus jeder Klasse von schwach 2-äquivalenten Wegen enthält.

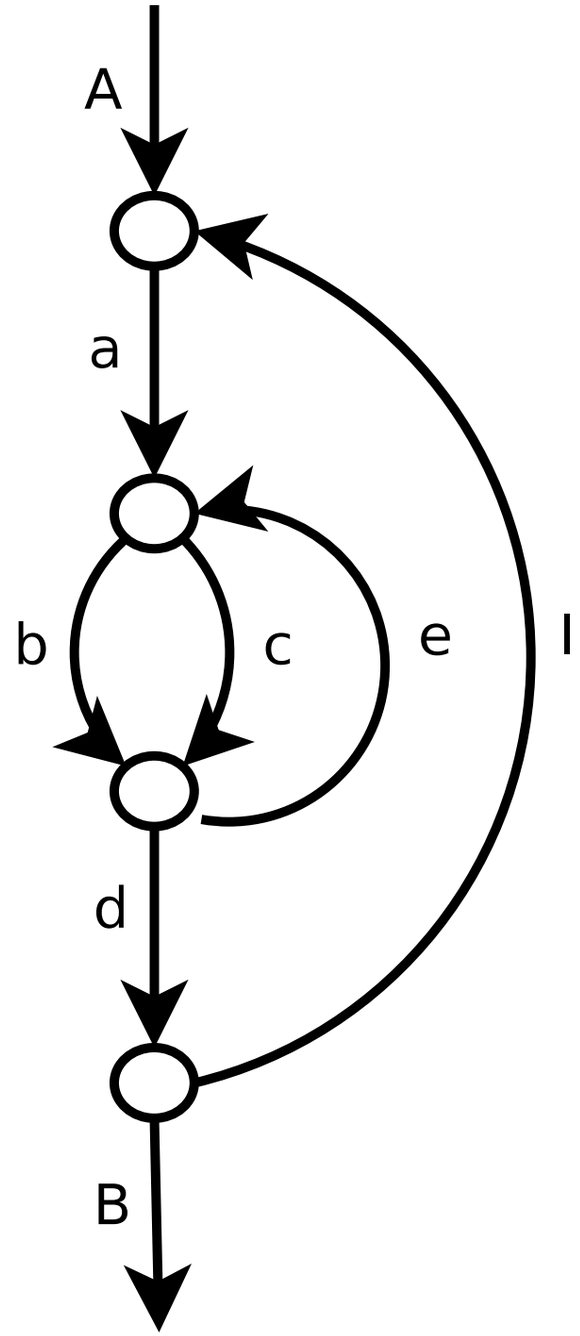
(→ alle Fälle k=0, k=1 und ein weiteres beliebiges k)

schwache C_{GI}-Überdeckung

Wähle eine Testfallmenge T, die $C_i(1)$-Überdeckung erfüllt

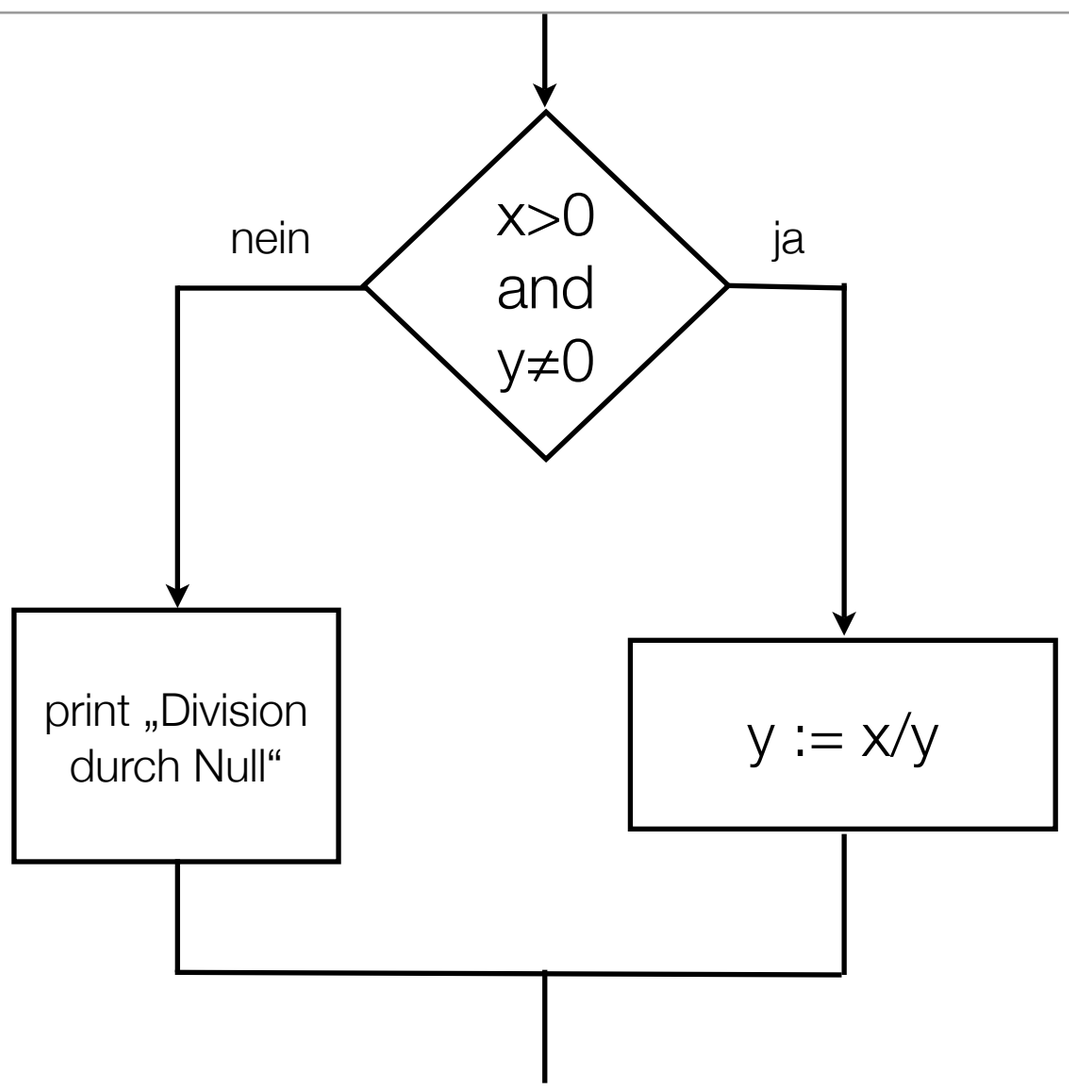
(→ alle Fälle k=0, k=1)

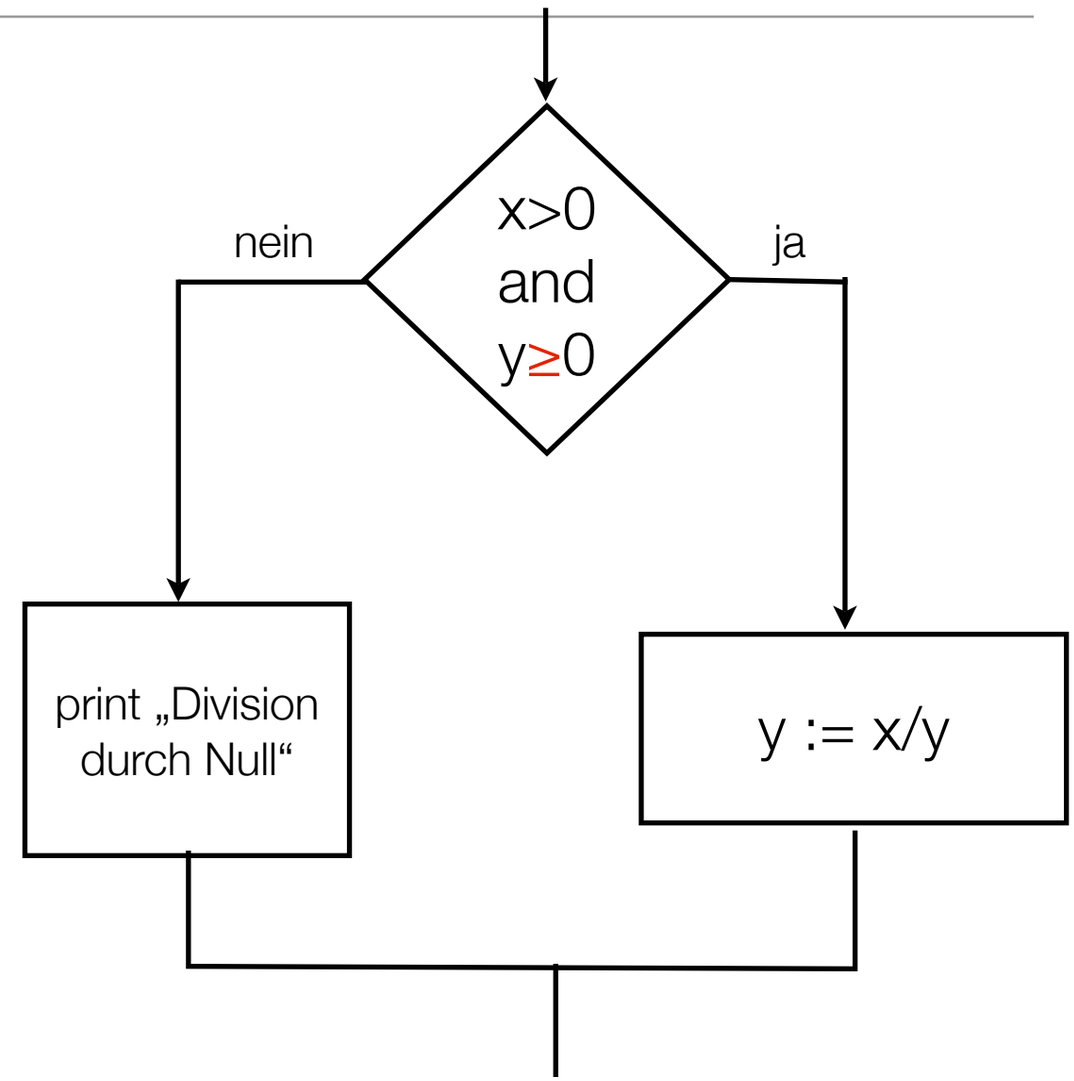
Beispiel: Zwei geschachtelte Schleifen



Ausdrucksbezogenes Testen
- boolesche Ausdrücke

Motivation





- T={(x=1,y=1), (x=-1, y=-1)} erfüllt TWM_1 (Zweigüberdeckung), deckt den Fehler aber nicht auf!

Beobachtung

Untersuchung des Kontrollflusses nicht ausreichend

➡ Anweisungen selbst können fehlerhaft sein:

- Anweisung korrekt, aber die referenzierten Werte sind falsch

  → Datenflußbezogener Test, nächste Woche

- Anweisung falsch

  →beliebiger Rechenfehler

  →Spezialfall: boolescher Ausdruck oder Bereichsfehler

atomare Prädikatterme

Atomare Prädikatterme

- enthalten Relationssymbole wie <,≤,=,≠,...
- enthalten boolesche Variablen
- dürfen keine logischen Operatoren (and, or, xor, ...) enthalten

Beispiel:

if A>1 and (B=2 or C>1) and D then ... else ...

atomare Terme: A>1, B=2, C>1, D

atomare Bedingungsüberdeckung

C_2-Überdeckung

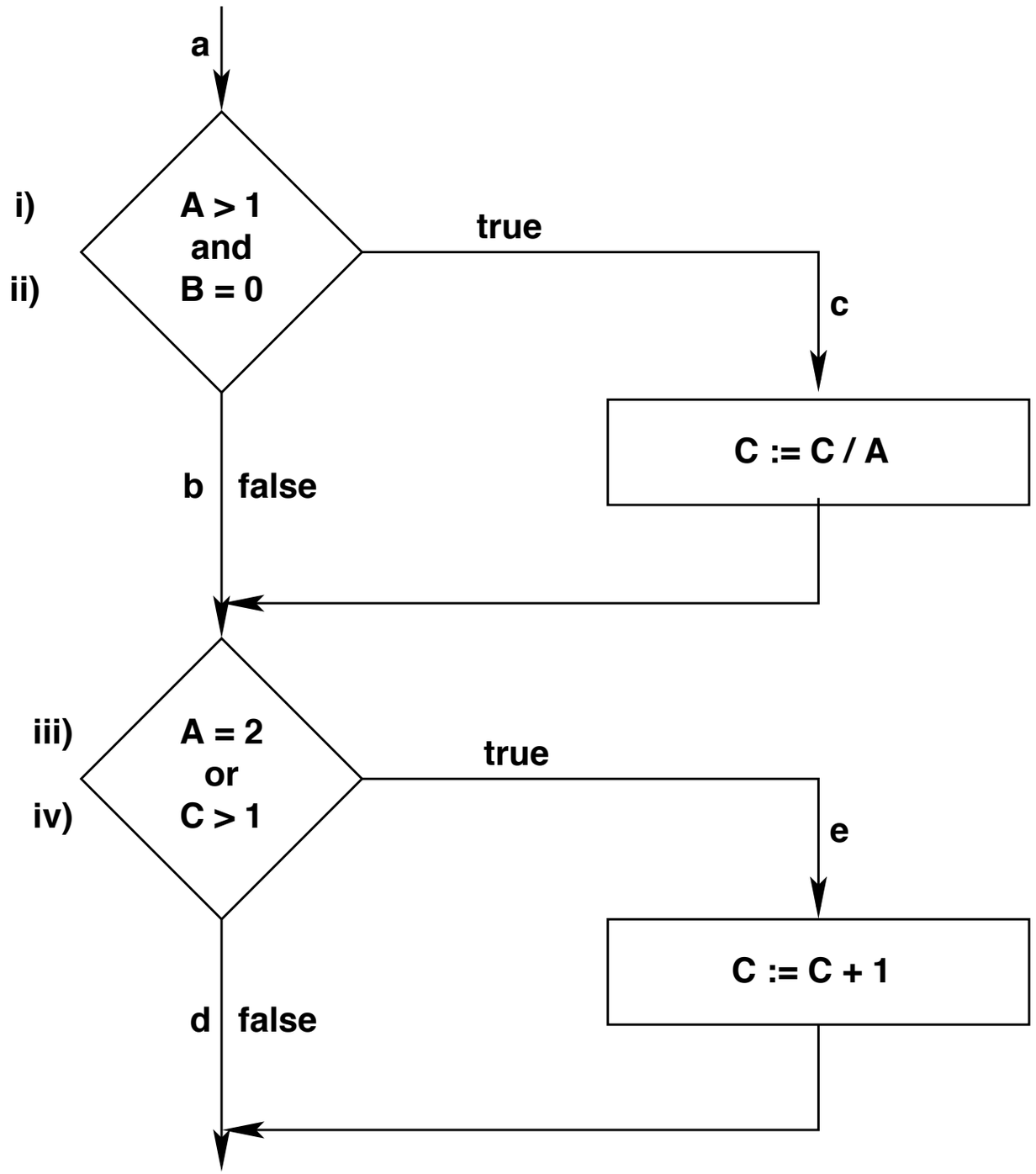
Wähle eine Testmenge T,

so daß es für jeden atomaren Prädikatterm p

jeweils mind. einen Testfall t gibt sodaß p=true und p=false erreicht werden.

→ C_2 garantiert keine C_0-Abdeckung (alle Anweisungen!)

Beispiel



t_1: A=2
B=1
C=4

t_2: A=1
B=0
C=1

Zweig-/Bedingungsüberdeckung

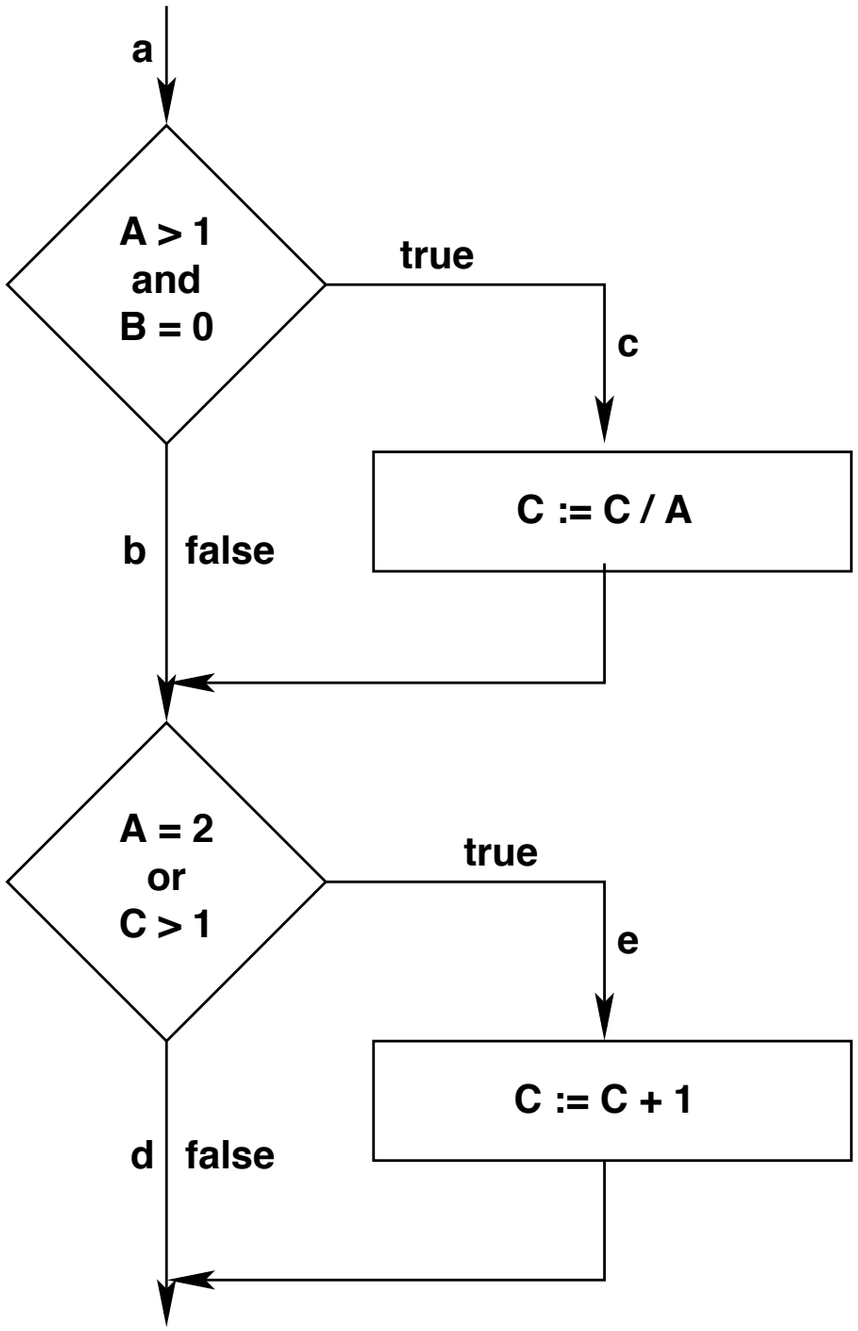
Eine Testdatenmenge T erfüllt die **Zweig/Bedingungsüberdeckung**

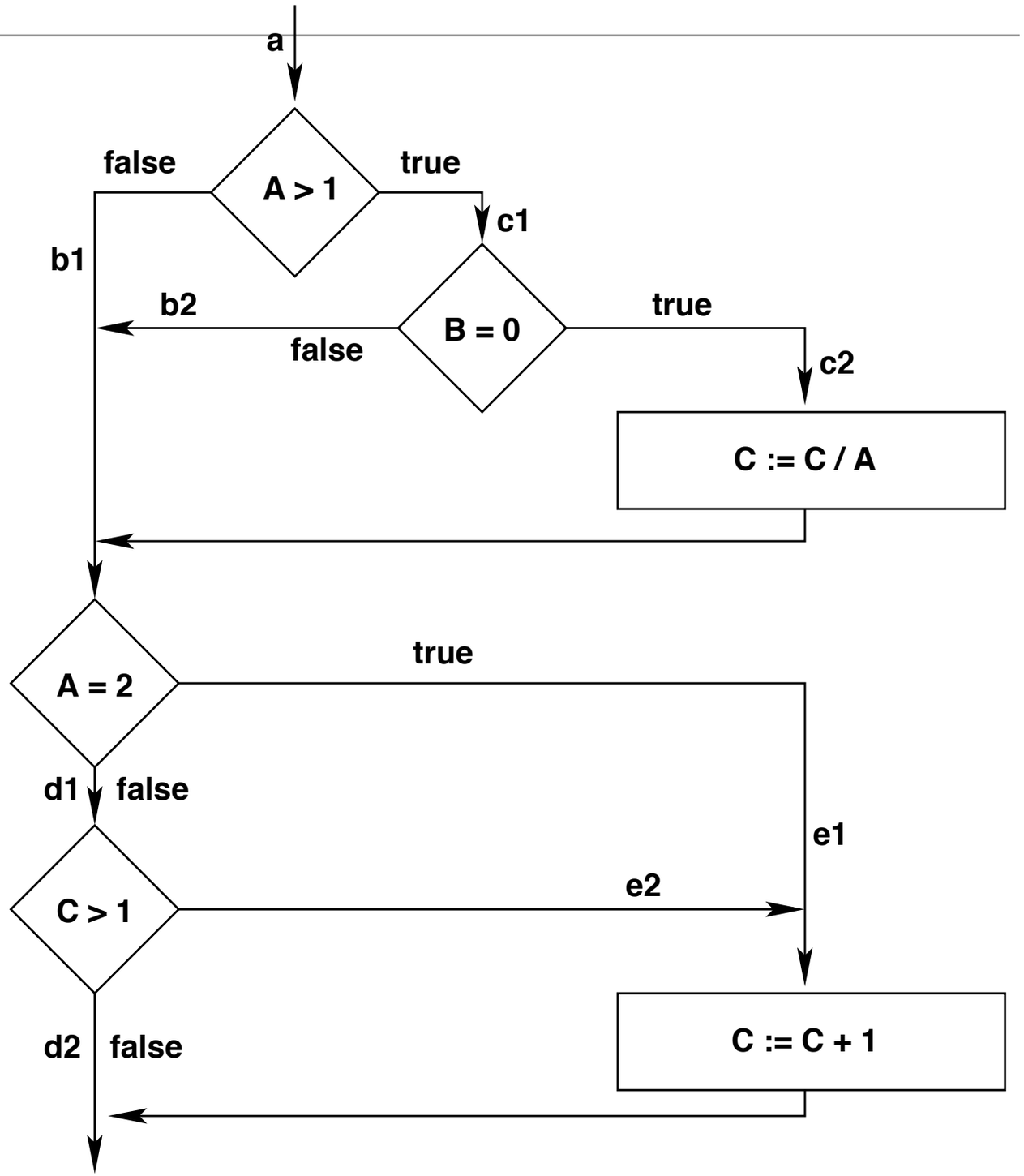
⇔ T erfüllt C_2-Überdeckung und C_1-Überdeckung

→ immer noch nicht schön:

- keine C_1-Überdeckung von Entscheidungszweigen auf Maschinen-Ebene
- Maskierung von falschen atomaren Bedingungen

Beispiel: Unterschiede auf Maschinen-Ebene





Minimale Mehrfachbedingungsüberdeckung

C_2(mM)-Überdeckung

Erstelle eine Testfallmenge T derart, daß

- für jeden booleschen Ausdruck des Programms
- jede Kombination von Wahrheitswerten berechnet wird,

  bei denen die Änderung des Wahrheitswertes eines atomaren Prädikats

  der Wahrheitswert der gesamten logischen Verknüpfung ändern würde

➡ vgl. Pfadsensitivierung bei Ursache-Wirkungs-Graphen

➡ C_2(mM) impliziert C_1-Überdeckung

Mehrfachbedingungsüberdeckung

C_2(mM) erkennt keine falschen log. Operatoren (z.B. XOR statt OR)

C_2(M)-Überdeckung

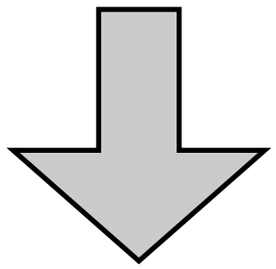
Erstelle eine Testfallmenge T mit derart, daß

- für jeden booleschen Ausdruck des Programms
- jede Kombination von Wahrheitswerten für seine atomaren Prädikate berechnet wird

Anmerkungen zum Testen boolescher Ausdrücke

- Es ist nicht entscheidbar ob alle Ausgänge ausführbar sind
- Instrumentierung des Programmkodes wird schwieriger:

  if A=2 or X>1 then ...



if A=2 then ZählerAT++; else ZählerAF++;

if X>1 then ZählerXT++; else ZählerXF++;

if A=2 or X>1 then ...

Ende der heutigen Vorlesung

Danke fürs Zuhören!

Bis nächste Woche :-)