Technische Fakultät
Universität Bielefeld

Vorlesung

Softwaretest und -optimierung

Version 2012

Dr. Carsten Gnörlich

Rechnerbetriebsgruppe

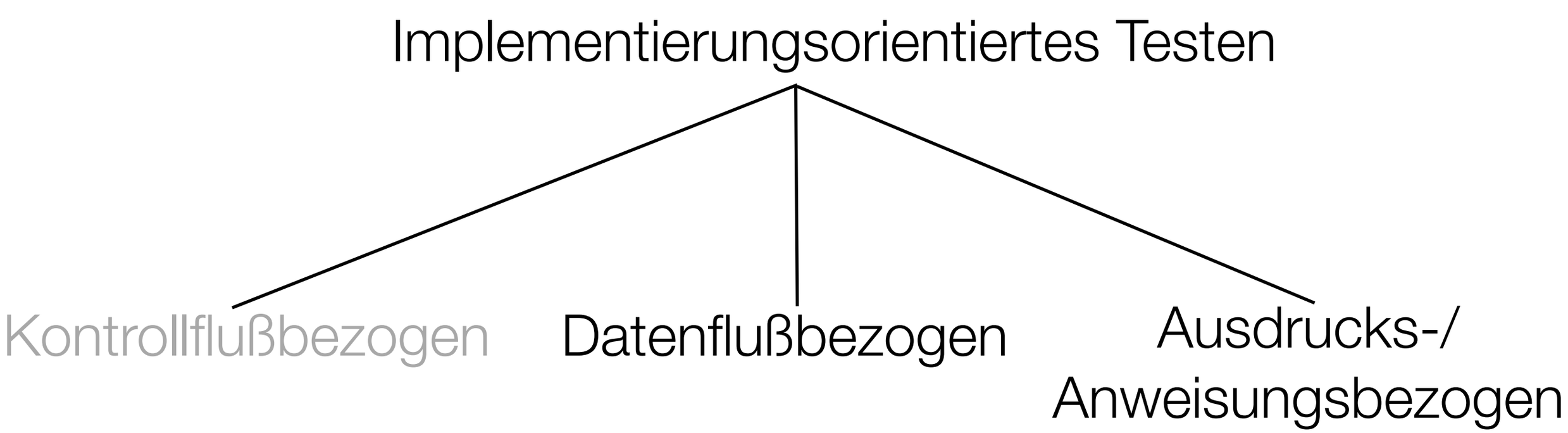
Kap. 5 - Strukturorientiertes Testen II

(= Kap. 9 aus Riedemann)

Implementierungsorientiertes Testen

Grundidee:

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



Fehlermöglichkeiten

- Kontrollfluß (Anweisungen, Wege, boolesche Ausdrücke) ✔
- Anweisungen selbst können fehlerhaft sein:
  - Anweisung falsch
    - → Spezialfälle: boolescher Ausdruck ✔ oder Bereichsfehler
    - → beliebiger Rechenfehler
  - Anweisung korrekt, aber die referenzierten Werte sind falsch
    - → Datenflußbezogener Test

Ausdrucksbezogenes Testen
- Bereichsfehler finden

Motivation von Bereichsfehlern

- richtig wäre: if(A>1 and B=0)
- realisiert wurde: if(A>2 and B=0)

➡ wird beim Testen von booleschen Ausdrücken nur per Zufall gefunden

Beobachtung:

- Kontrollflußweg durch n Variablen eingeschränkt

→ geometrische Interpretation als Teilmenge im n-dimensionalen Raum

→ Bereichsfehler verschiebt die Oberfläche(n) der Teilmenge

→ Spezialfall 2 Variablen: Fläche in (x/y)-Koordinatensystem

Bereichsüberdeckung

Ist ein Bereich durch eine UND-Verknüpfung über <,≤,≥,> beschrieben
und hat der Eingabebereich lineare Grenzen in 2 Variablen,

so teste wie folgt:

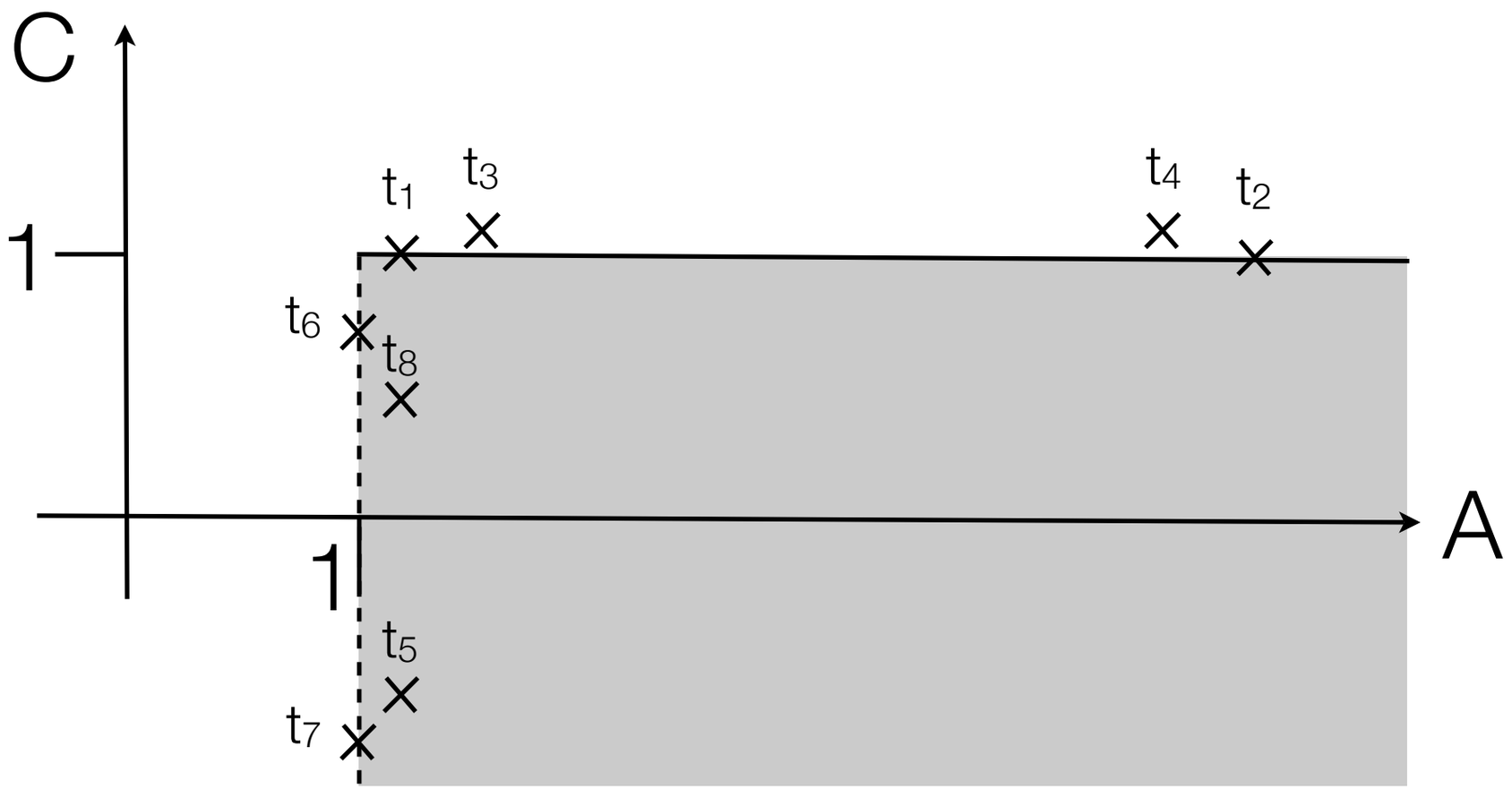
a) je zwei Testdaten auf der Grenze

b1) ein Testdatenpaar außerhalb der Grenzen für ≤,≥

b2) ein Testdatenpaar innerhalb der Grenzen für <,>

Beispiel

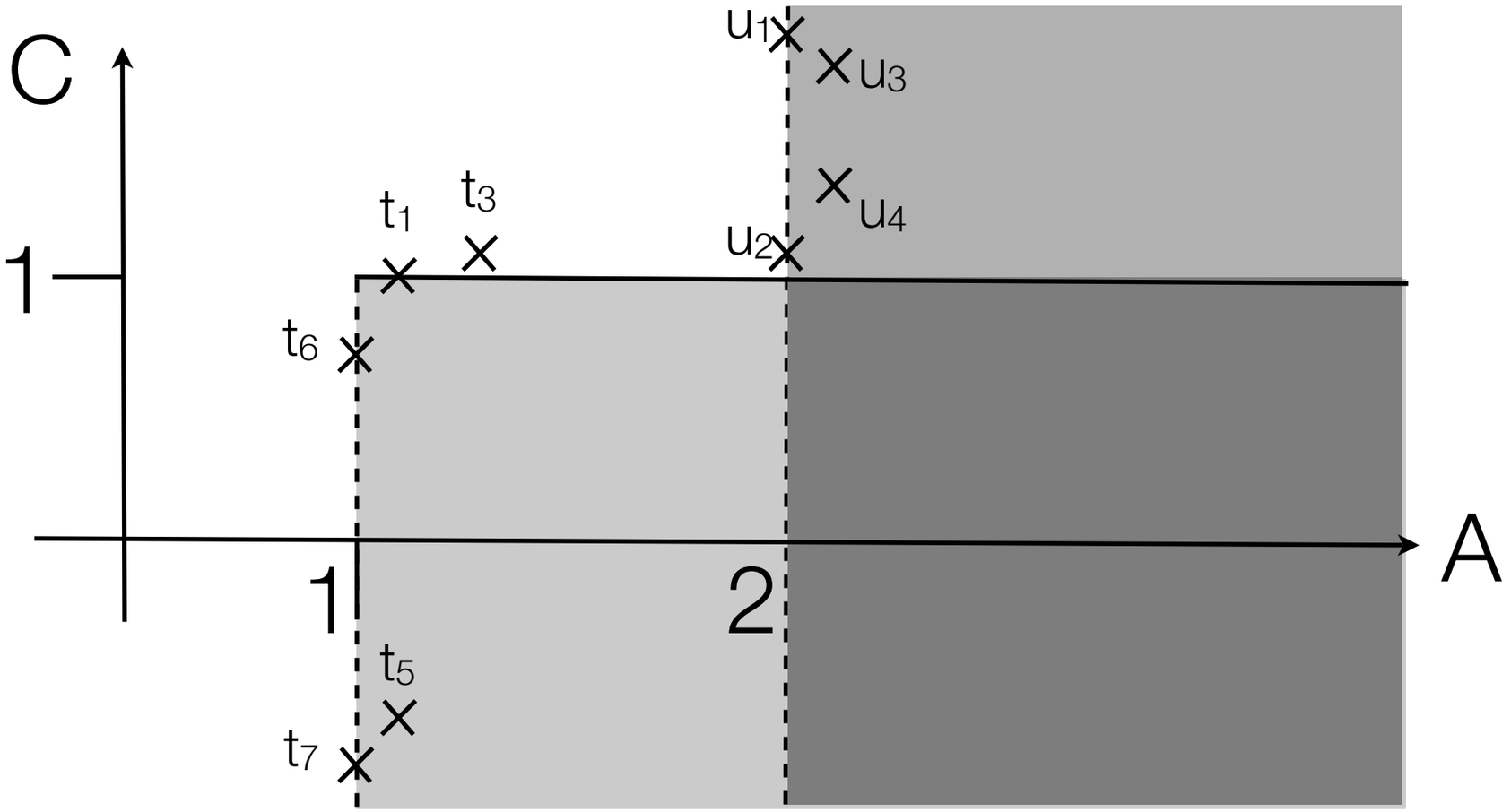
Bedingung „A > 1 and C ≤ 1“



Abstand der außen/innen liegenden Testfälle sei kleinstmöglich
→bei jeder Verschiebung der Fläche wechselt mind. ein Testfall die Seite

Erweiterung um „ODER“

Bedingung „(A > 1 and C ≤ 1) or A > 2“



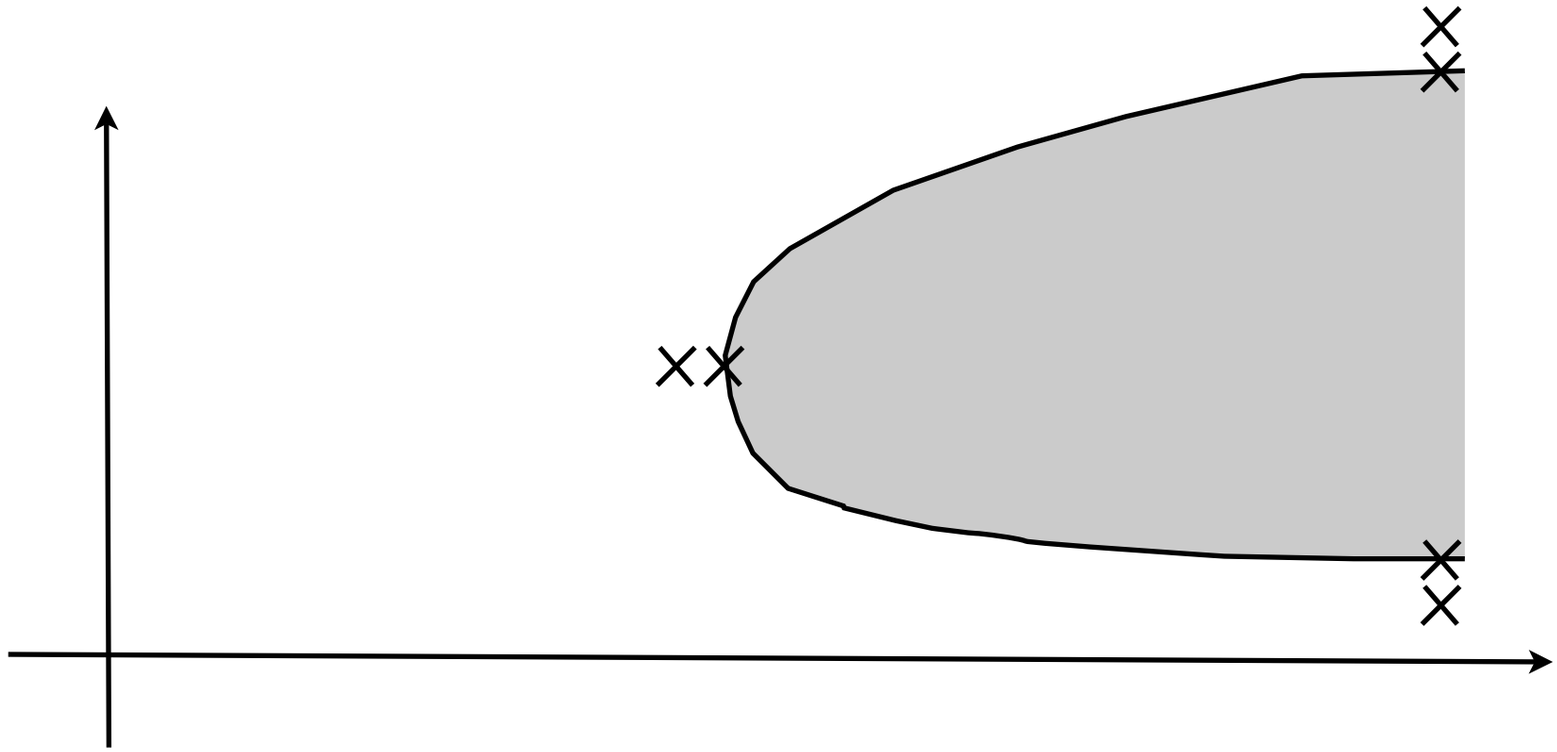
Hinweis: Testdaten mit C ≤ 1 und A = 2 + ε werden durch A>2 maskiert

Verallgemeinerungen

- für n Variablen spannen die Grenzen eine n-dimensionale Hyperebene auf

→ n „linear unabhängige“ Testpunkte wählen

- für nicht-lineare Bereiche Extrema testen:



Ausdrucksbezogenes Testen
- allgem. Programmanweisungen

Berechnungsfehleraufdeckung

1. Aufstellen eines Fehlermodells für die Funktion/Anweisung f
   - F_f = Menge aller Fkt. die durch Fehler aus f entstehen können
   - außerdem sei $f \in F_f$

   | kompetente Programmier-Hypothese

2. Bestimme einen verläßlichen Test T für F_f und f:

   ∀ f‘ aus F_f gilt:
   $\forall t \in T: f'(t)=f(t) \Rightarrow f' = f$

   Bemerkung:

   - nicht alle Fehler können getestet werden (Erfahrung → Fehlermodell)

Betrachtete Fehlermodelle

1. Datenzugriff
2. Datenspeicherung
3. arithmetischer Ausdruck
4. arithmetische Relation
5. Boolescher Ausdruck

Datenzugriffs-Fehler

Fehlerart: Zugriff auf eine falsche Variable

Testdaten: Alle Variablen für die zu testende Zuweisung mit verschiedenen Werten belegen

Beispiel:

```
int binary_search(char *key, int n)
{  int npos=0;
   int low=1;
   int high=n;
   int mid=(low+high)/2;

   /* Suche mit Hilfe von low, mid, high */
```

n ≥ 5 wählen

Datenspeicherungs-Fehler

Fehlerart: Speichern eines Wertes in eine falsche Variable

Testdaten: Die zu testende Zuweisung muß alle Variablen mit Werten belegen die sich von den vorherigen Werten unterscheiden

Beispiel:

```
int *liste = malloc(n*sizeof(int));

for(i=0; i<n; i++)
   liste[i] = 42;   /* initialer Wert */

...

if(...) liste[j| = y;   /* Wähle y != 42 um diese Anweisung zu testen */
```

Fehler in arithmetischen Ausdrücken

Fehlerart: Ausdrücke über +,-,*,/ falsch zusammengesetzt

Testdaten:

a) für einfache additive/multiplikative Fehler: neutrale Elemente vermeiden

Bsp: g(x) = ax + b → a,x ≠ {0,1} und b ≠ 0 nehmen

b) ansonsten mathematische Eigenschaften des Ausdrucks nutzen

Bsp: Für ein Polynom vom Grad n reichen n+1 Testfälle

c) für endliche / „böse“ Konstrukte gegen Referenzimplementierungen testen

Bsp: Primkörper gegen Maple/Mathematica testen

- Polynom hat max. n Nullstellen und damit max. n gleiche Werte zu Polynom gleichen Grades (es sei denn der Fehler ändert den Grad - kompetente Programmierhypothese!)

Fehler in arithmetischen Relationen

Fehlerart: a) falsches Symbol aus ≤<=>≥≠ gewählt

b) falscher Bereich: (E+k) r 0 statt (E+k‘) r 0, r∈{≤<=>≥≠}

Testdaten: Teste jeweils die Grenzwerte, an denen der Wert sich ändert.

Beispiel: if(a - 7.5 == 0.0) {...} → a = 7.5 ± ε testen

Nebenbei: Test von Floats auf Gleichheit ist eine schlechte Idee

Besser: if(fabs(a-7.5) < EPSILON) { } mit geeignet kleinem Wert für EPSILON

Fehler im booleschen Ausdruck

Fehlerart: boolescher Ausdruck falsch konstruiert

Testdaten:

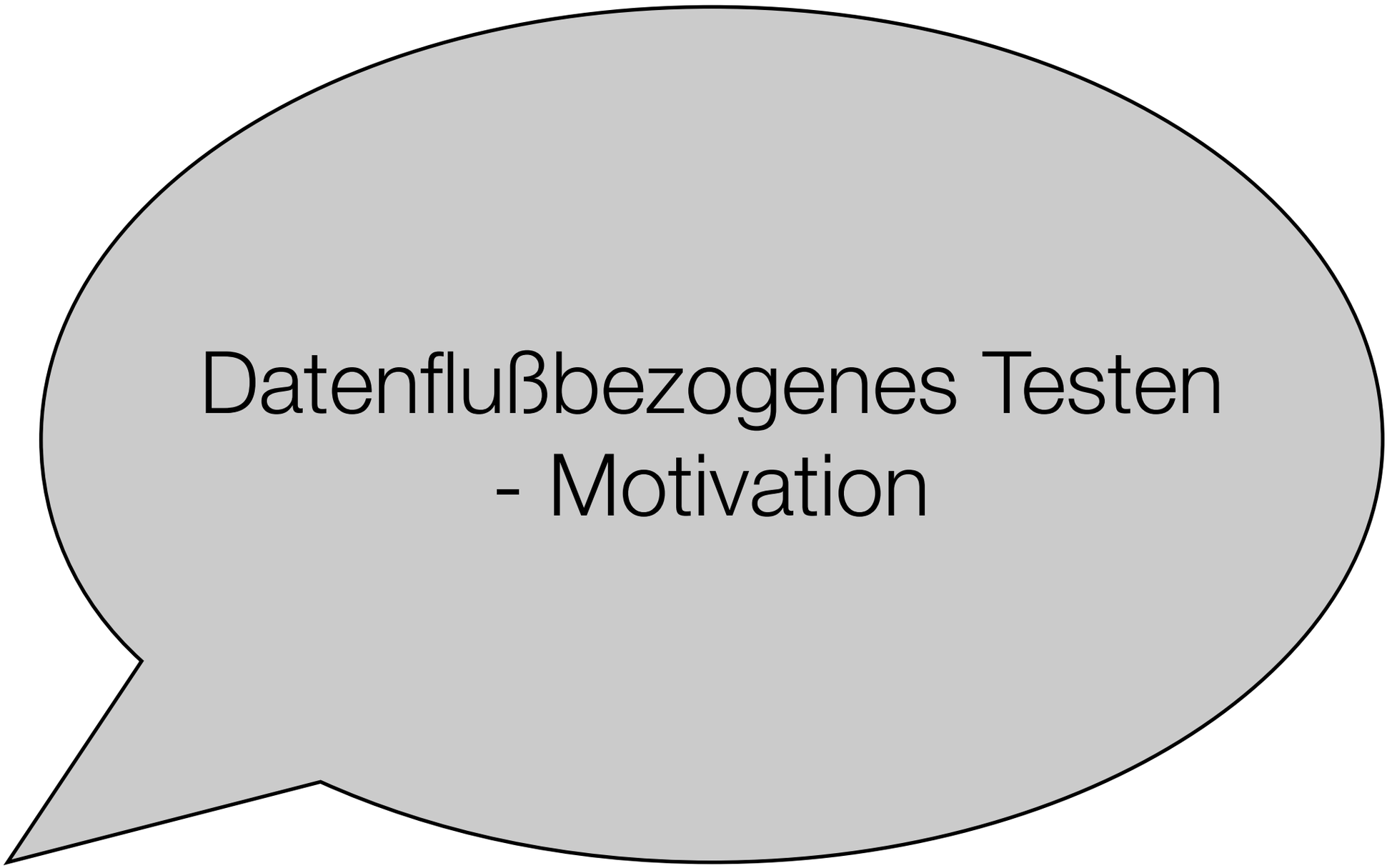
a) falls boolescher Ausdruck Kontrollfluß steuert

→ C_2(M)-Überdeckung, siehe letzte Vorlesung

(Mehrfachbedingungsüberdeckung)

b) falls Ergebnis des booleschen Ausdrucks Rückgabewert ist

→ analoges Vorgehen; Rückgabewert ist Ausgabe des Programmteils


Datenflußbezogenes Testen
- Motivation

Motivation und Ansatz

Die Anweisungen selbst sind korrekt, aber

- die referenzierten Werte werden vorher falsch berechnet

Teile Anweisungen in zwei Klassen auf:

- Anweisungen die den Wert einer Variablen berechnen (definieren)
- Anweisungen die diesen Variablenwert benutzen (referenzieren)

→ verfolge die Interaktion der jeweiligen Anweisungspaare

→Analyse des „Datenflusses“

Datenflußgraph

Ein Datenflußgraph ist ein Kontrollflußgraph,
dessen Knoten *k* mit folgenden Mengen annotiert sind:

DEF(k): Menge der Variablen *x*, denen innerhalb des Knotens ein Wert zugewiesen wird und nicht wieder später undefiniert wird.

UNDEF(k): Menge der Variablen, die innerhalb des Knoten undefiniert und nicht wieder erneut belegt werden.

REF(k): Menge der Variablen, die innerhalb des Knotens referenziert werden (und vorher nicht undefiniert werden).

Bem: Referenz von undef. Variablen → Datenflußanomalie (später)

+ die Einschränkungen auf der nächsten Seite

Einschränkungen für Knoten im Datenflußgraphen

- keine lokalen Datenflüsse innerhalb des Knotens

  (die gleiche Variable definieren und danach referenzieren)

  → Knoten bei Bedarf aufteilen

- Module mit Ein-/Ausgabe erhalten zusätzlich Knoten K_{ein} und K_{aus}

- Entscheidungsknoten dürfen Variablen nur referenzieren

  also Seiteneffekte wie if(a++ >0) in zwei Knoten aufteilen

Datenflußbezogenes Testen
- einfache Kriterien

Alle defs-Kriterium

Wähle eine Testdatenmenge T,

die für jede Variable X und jede Definition von X

mindestens einen Weg enthält auf dem X referenziert wird.

Bemerkungen:

- es kann mehr als 1 Referenz zu einer Definition geben,
  aber nur eine wird getestet

alle Def/Ref-Interaktionen

Wähle eine Testdatenmenge T,

die alle Paare von Definitionen und Referenzen einer Variablen testet.

Bemerkung:

- unter Umständen werden nicht alle Entscheidungskanten ausgeführt

alle Referenzen

Wähle eine Testdatenmenge T,

die zu allen Paaren von Definitionen und Referenzen auch die unmittelbaren Nachfolgeknoten des Referenzknotens ausführt.

Motivation: Referenzen in Entscheidungsbäumen besser bewerten

- if-Anweisung selbst hat keinen Effekt
- erst die Aktion im then... / else...-Teil generiert meßbares Ergebnis

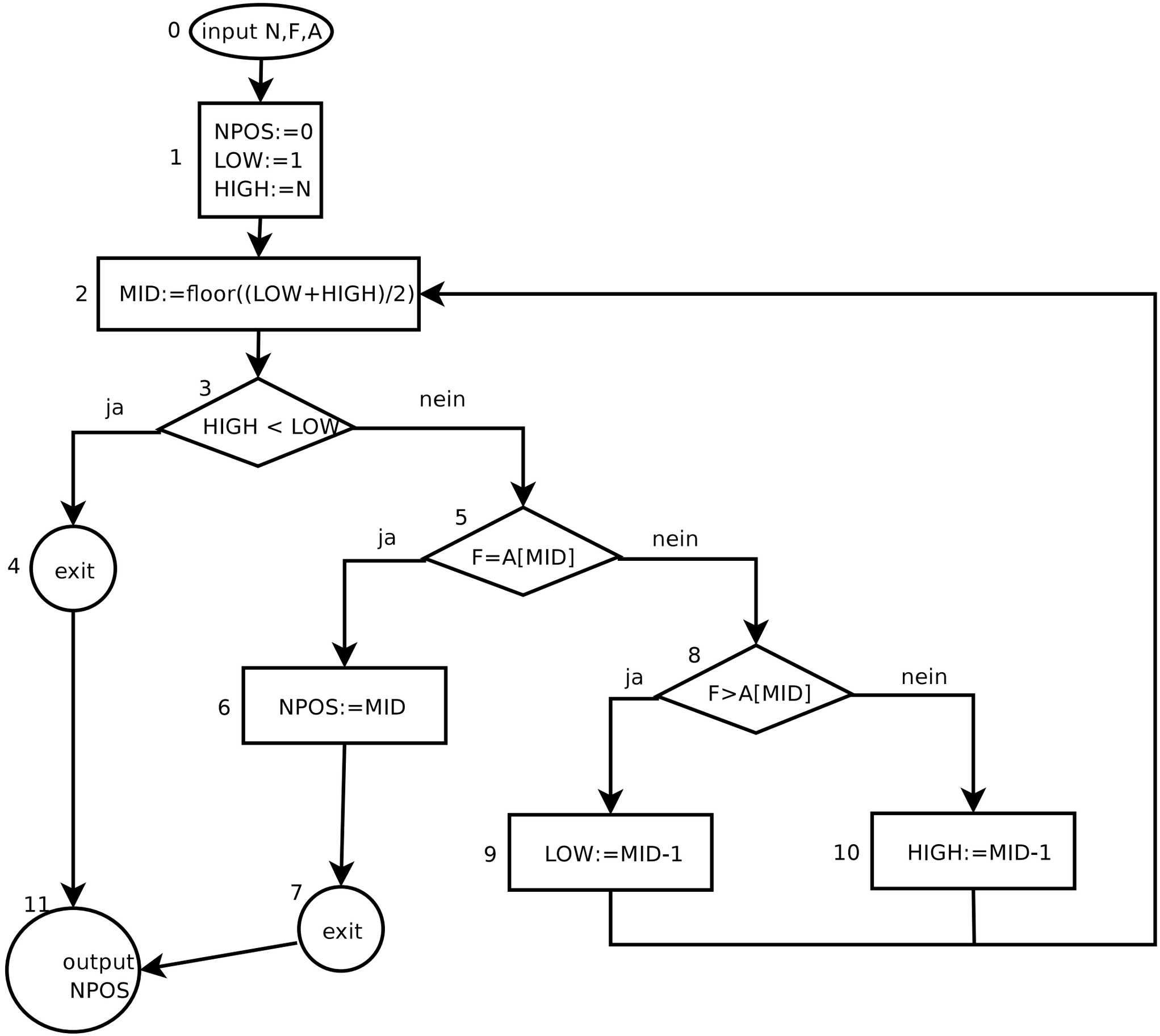
Bemerkung

Testen von Def/Ref-Paaren geschieht jeweils nur auf einem Weg:

- *x* wird auf allen Wegen zwischen Def und Ref nicht verändert

➡ Datenfluß-Fehler bzgl. *x* können auf diesen Wegen nicht entstehen

Datenflußbezogenes Testen
- Verkettung von Datenflüssen


0
input N,F,A
1
NPOS:=0
LOW:=1
HIGH:=N
2
MID:=floor((LOW+HIGH)/2)
3
HIGH < LOW
ja
nein
4
exit
5
F=A[MID]
ja
nein
6
NPOS:=MID
7
exit
8
F>A[MID]
ja
nein
9
LOW:=MID-1
10
HIGH:=MID-1
11
output
NPOS

k-DR-Interaktion

Für $k \geq 2$ heißt die Folge $(n_1,x_1,n_2,x_2, \ldots ,x_{k-1},n_k)$ eine k-DR-Interaktion gdw:

1. $n_1,n_2,\ldots,n_k$ paarweise verschiedene Knoten des Datenflußgraphen sind, mit der möglichen Ausnahme $n_1=n_k$

2. $x_1,\ldots,x_{k-1}$ Variable sind (gleiche oder verschiedene)

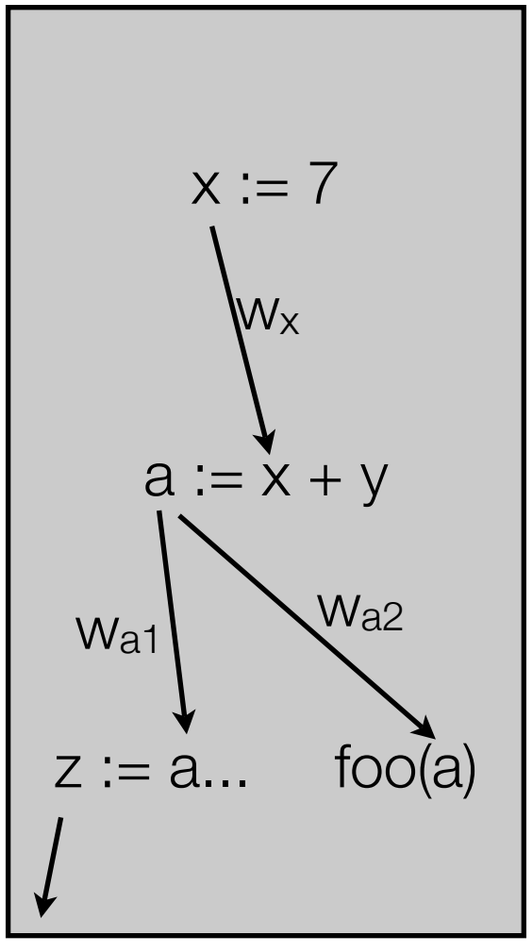
3. für jedes i mit $1 \leq i \leq k$ gibt es einen Weg w_i so daß gilt:

   - x_i wird in Knoten n_i definiert

   - über w_i wird in Knoten n_{i+1} eine Referenz von x_i erreicht

Einschränkung: nur eine Zuweisung pro Knoten
(Knoten ggf. aufteilen)

Anschaulich: „Übergabekette“ von Variablenwerten verfolgen



alle k-DR-Interaktionen-Kriterium

Wähle eine Testmenge T,

die alle k-DR-Interaktionen bis zu der Länge k ausführt

und in den Wegen jeweils auch noch der Nachfolgeknoten des letzten Interaktionselementes enthalten ist.

Bemerkungen

- für k=2: äquivalent mit „alle Referenzen“

Ende der heutigen Vorlesung

Danke fürs Zuhören!

Bis nächste Woche :-)