Technische Fakultät
Universität Bielefeld

Vorlesung

Softwaretest und -optimierung

Version 2012

Dr. Carsten Gnörlich

Rechnerbetriebsgruppe

Kap. 3 - Ursache-Wirkungs-Graphen und Pfadausdrücke

(= Kap. 4.2, 5.1 aus Riedemann)

Wiederholung

Datenbereichsbezogenes Testen

- aus Spezifikation der Parameter Testfälle ableiten

Vorgestellte Verfahren

- Zufälliges Testen, Fehlererwartungsmethode
- Äquivalenzklassen
- Grenzwertmethode
- Gleich kommt noch: Ursache-Wirkungs-Graphen

➡ idealen Test durch geschickte Wahl der Testfälle annähern

Heute haben wir folgendes vor

- Ursache-Wirkungs-Graphen
- Weiterführung der spezifikationsorientierten Verfahren
  - Pfadausdrücke (Testen von Reihenfolgebedingungen)

Äquivalenzklassen
+ Ursache-Wirkungsgraphen

Wiederholung

„Ein Päckchen wiegt zwischen 500 und 2000 Gramm“

Äquivalenzklassen

- [500..2000] gültige ÄK
- [-∞..499], [2001..+∞] ungültige ÄK

Grenzwertmethode

- Testdaten T = {499,500,2000,2001}

Motivation

- Fehler treten vielleicht nur bei bestimmten Kombinationen von Ein-/Ausgabewerte-Klassen auf

  aber: exponentielle Explosion bei Test von Kombinationen aller Eingabe-ÄK

  Beispiel: 7 Eingabebedingungen mit je 3 ÄK -> 3^7 = 2187 Kombinationen

Idee:

- Ausgabe-ÄK = Wirkungen
- Eingabe-ÄK = Ursachen (Testfälle!)

➡ Rechne von den Wirkungen so auf die benötigten Ursachen zurück, daß bei fehlerhafter Behandlung einer Ursache die Wirkung nicht eintritt.

1. Spezifikation in bearbeitbare Stücke zerlegen

- typischerweise eine Teilfunktion
- oder ein kleines Modul

➡ Ursache-Wirkungsgraph (UWG) soll nicht zu groß werden

Beispiel: (wird fortgeführt)

Funktion soll Wörter w_1w_2 akzeptieren mit

$w_1 \in \{A, B\}$, $w_2 \in \{0,1,2,...,9\}$

Ausgaben: „gut“ wenn das Wort zulässig ist
„Fehler 1“ wenn w_1 unzulässig ist
„Fehler 2“ wenn w_2 unzulässig ist

2. Ursachen und Wirkungen spezifizieren

- Ursachen sind Eingabebedingungen,
  deren Zutreffen mit einem booleschen Wert gekennzeichnet sind.

  Bsp: Wochentag = „Sonntag“
  0 < Zeilenanzahl < 900

- Wirkungen sind:

  - Ausgabebedingungen

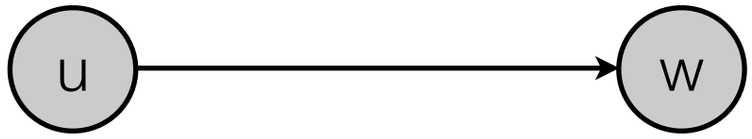
  - Systemtransformationen
    (= innere Zustände, die nicht an den Ausgaben sichtbar werden)

➡ alle Ursachen und Wirkungen mit einer eindeutigen Zahl kennzeichnen

3. Ursache-Wirkungsgraph erstellen

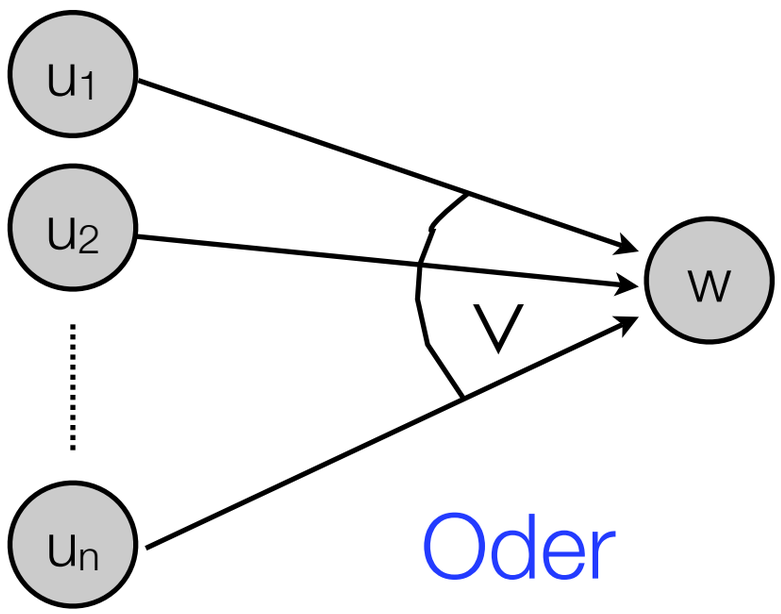
ggf. Zwischenknoten einfügen für komplexe Berechnungen!

- gerichteter zyklenfreier boolescher Graph
- drückt logische Beziehungen zwischen Ursachen und Wirkungen aus

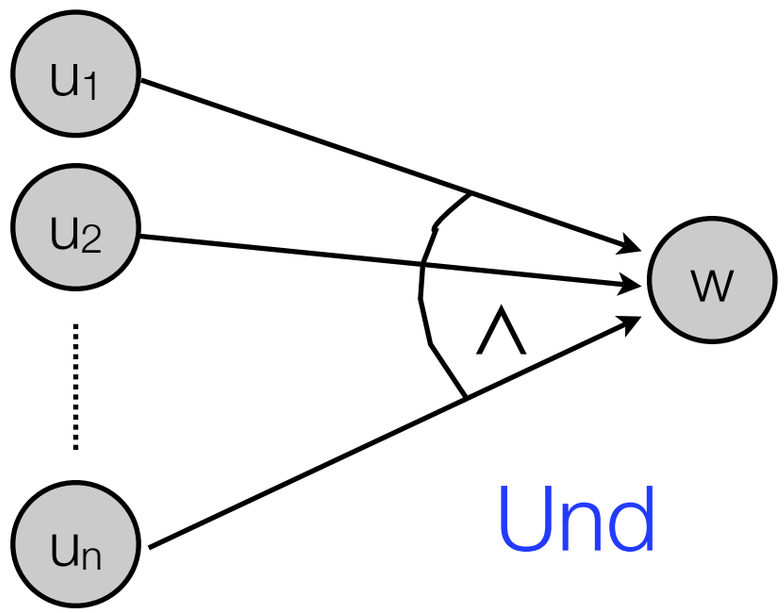


Identität

Negation

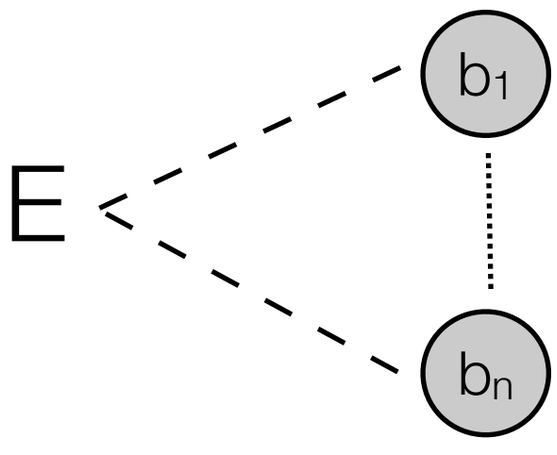


Oder



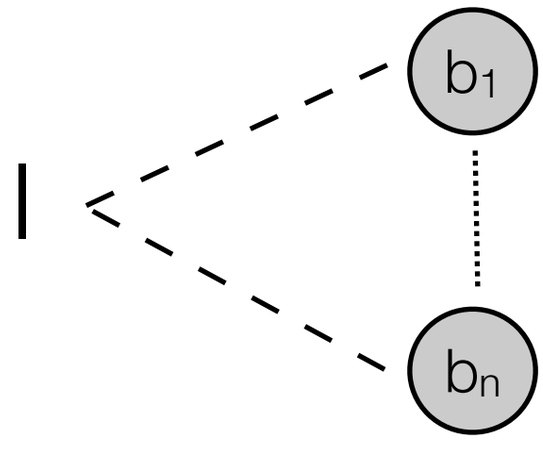
Und

4. Unmögliche Eingabekombinationen ausschließen (1)



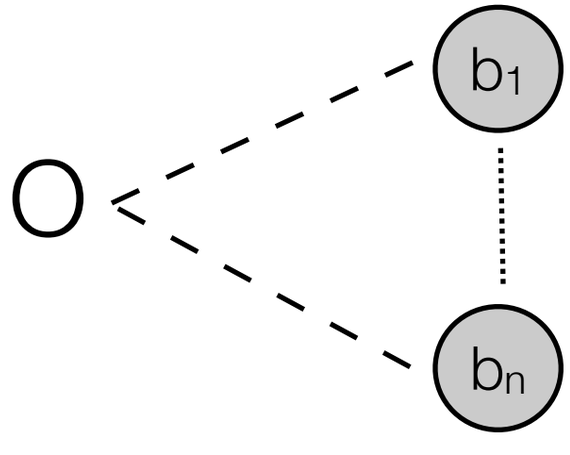
Exklusiv:

Höchstens eine der Bedingungen erfüllt



Inklusiv:

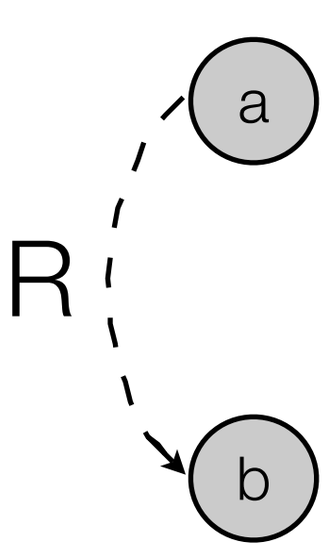
Mindestens eine der Bedingungen erfüllt



Oder:

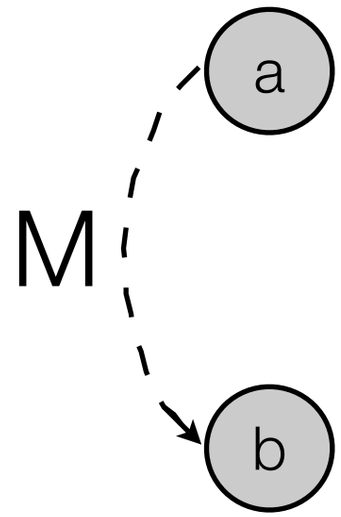
Genau eine der Bedingungen erfüllt

4. Unmögliche Eingabekombinationen ausschließen (2)



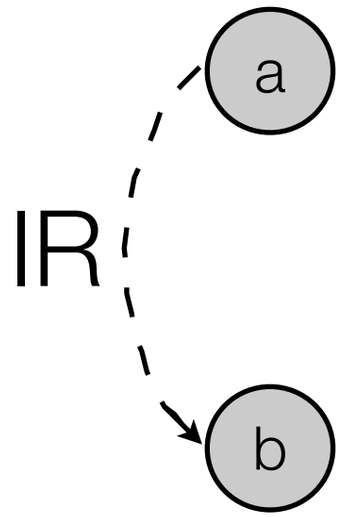
Requires:

Erfüllung von a
erfordert Erfüllung von b



Maskiert:

Erfüllung von a impliziert
Nichterfüllung von b



Irrelevant:

Erfüllung von a impliziert
daß Bedingung b entfällt

5. Leite aus UWG eine Entscheidungstabelle ab

a) Wähle eine Wirkung aus

b) Bestimme durch Zurückverfolgen des UWG alle Komb. von Ursachen, für die

   i) die Wirkung vorhanden ist

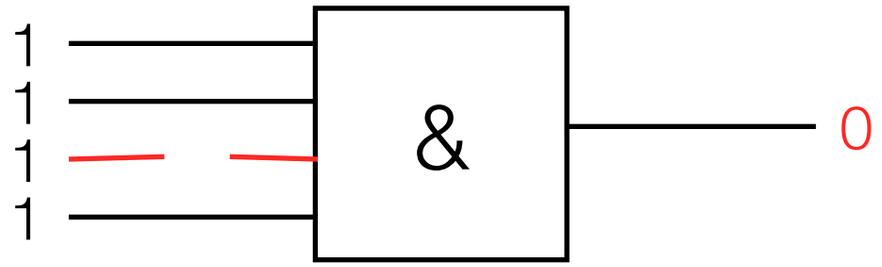
   ii) die Wirkung nicht vorhanden ist.

Hauptwerkzeug: Pfadsensitivierung aus der Chip-Fertigung

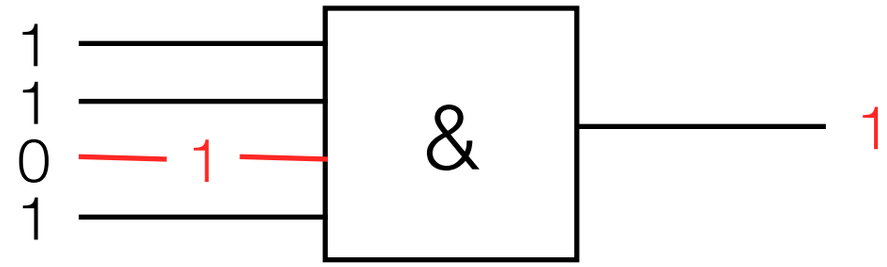
- wähle für jeden Knoten n-1 Eingänge so, daß ein Fehler im n-ten Eingang einen Fehler am Ausgang produziert

➡ statt 2^n Kombinationen nur n Testfälle!

Pfadsensitivierung für UND-Knoten

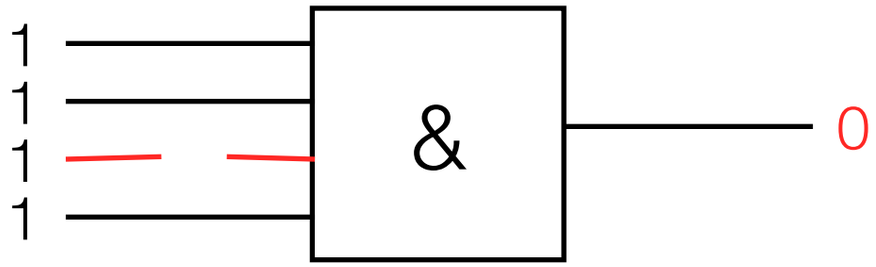


Leitung unterbrochen („stuck at 0“)

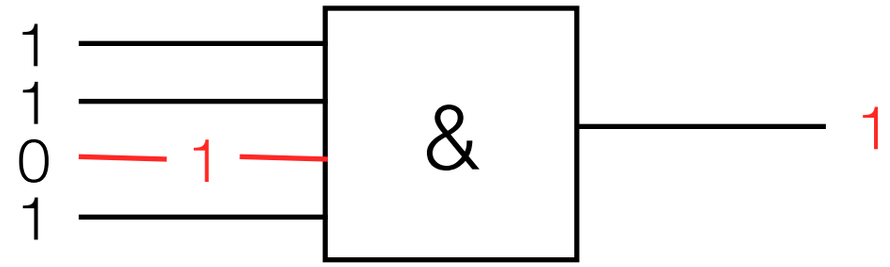


Leitung kurzgeschlossen („stuck at 1“)

Pfadsensitivierung für UND-Knoten



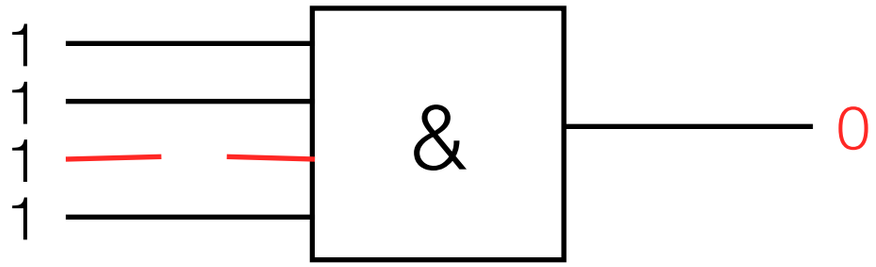
Leitung unterbrochen („stuck at 0“)



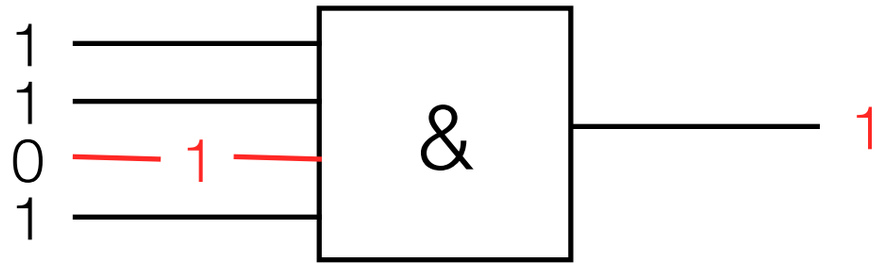
Leitung kurzgeschlossen („stuck at 1“)

x_1	x_2	x_3	...	x_n	Eigenschaft	y (fehlerfrei)
1	1	1	...	1	alle Eingänge 1	1
0	1	1	...	1	genau ein	0
1	0	1	...	1	Eingang 0,	0
1	1	0	...	1	alle anderen	0
⋮	⋮	⋮	...	1	Eingänge 1	⋮
1	1	1	...	0		0

Pfadsensitivierung für UND-Knoten



Leitung unterbrochen („stuck at 0“)



Leitung kurzgeschlossen („stuck at 1“)

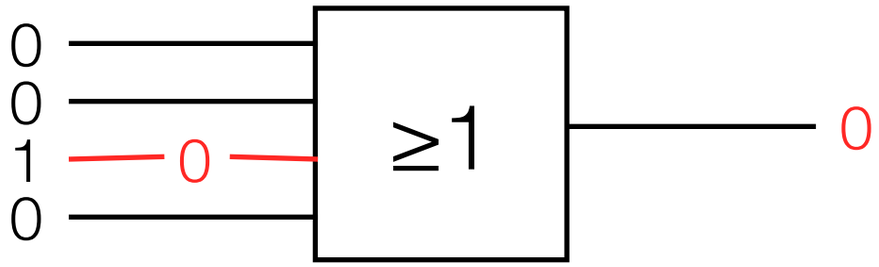
x_1	x_2	x_3	...	x_n	Eigenschaft	y (fehlerfrei)
1	1	1	...	1	alle Eingänge 1	1
0	1	1	...	1	genau ein	0
1	0	1	...	1	Eingang 0,	0
1	1	0	...	1	alle anderen	0
⋮	⋮	⋮	...	1	Eingänge 1	⋮
1	1	1	...	0		0

y=1 erwartet ⇒ alle „1“-Eingänge rekursiv zurückverfolgen

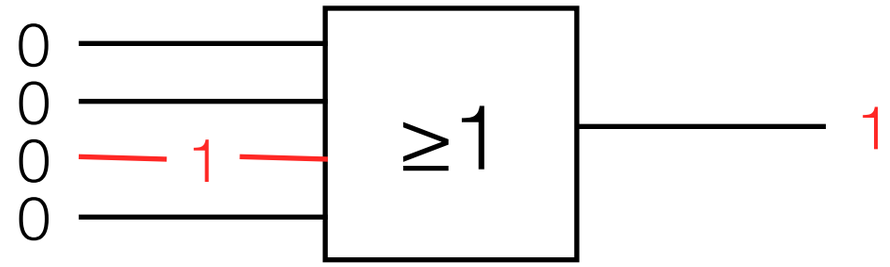
y=0 erwartet ⇒ für den „0“-Eingang alle Möglichkeiten zurückverfolgen

⇒ für die „1“-Eingänge eine beliebige Möglichkeiten nehmen

Pfadsensitivierung für ODER-Knoten

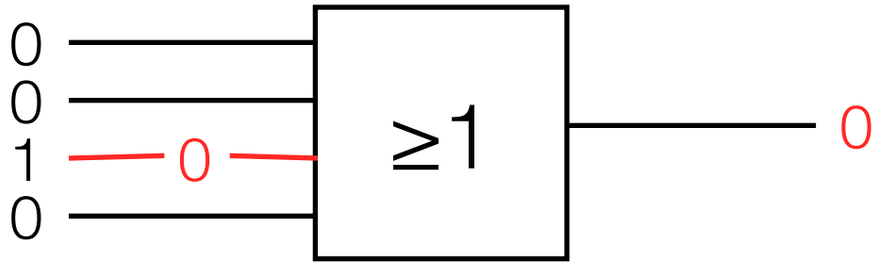


Leitung unterbrochen („stuck at 0“)

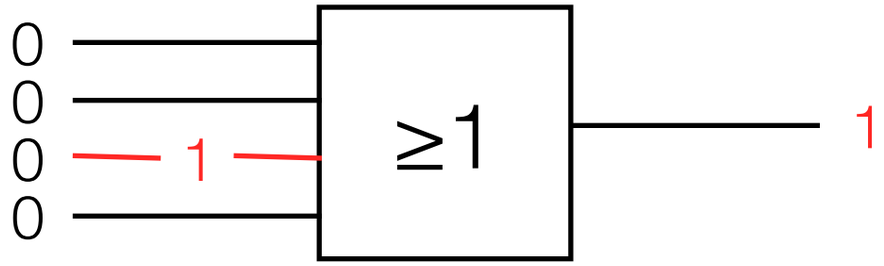


Leitung kurzgeschlossen („stuck at 1“)

Pfadsensitivierung für ODER-Knoten



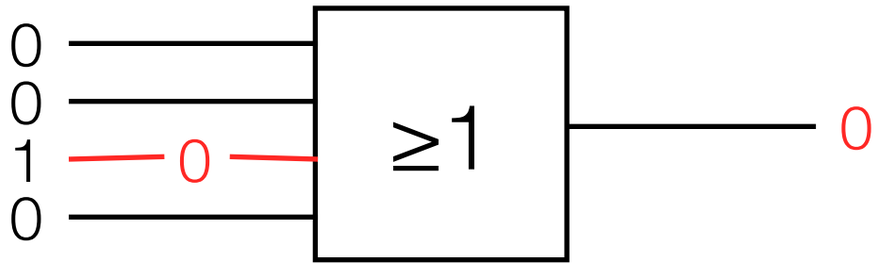
Leitung unterbrochen („stuck at 0“)



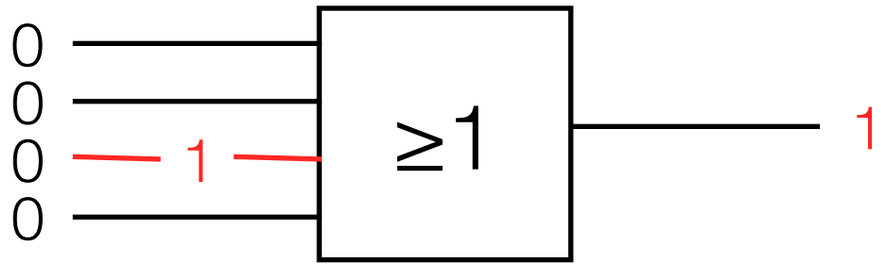
Leitung kurzgeschlossen („stuck at 1“)

x_1	x_2	x_3	...	x_n	Eigenschaft	y (fehlerfrei)
0	0	0	...	0	alle Eingänge 0	0
1	0	0	...	0	genau ein	1
0	1	0	...	0	Eingang 1,	1
0	0	1	...	0	alle anderen	1
⋮	⋮	⋮	...	0	Eingänge 0	⋮
0	0	0	...	1		1

Pfadsensitivierung für ODER-Knoten



Leitung unterbrochen („stuck at 0“)



Leitung kurzgeschlossen („stuck at 1“)

x_1	x_2	x_3	...	x_n	Eigenschaft	y (fehlerfrei)
0	0	0	...	0	alle Eingänge 0	0
1	0	0	...	0	genau ein	1
0	1	0	...	0	Eingang 1,	1
0	0	1	...	0	alle anderen	1
⋮	⋮	⋮	...	0	Eingänge 0	⋮
0	0	0	...	1		1

y=0 erwartet ⇒ alle „0“-Eingänge rekursiv zurückverfolgen

y=1 erwartet ⇒ für den „1“-Eingang alle Möglichkeiten zurückverfolgen

⇒ für die „0“-Eingänge eine beliebige Möglichkeit nehmen

Fortführung des Beispiels

erste Etappe: Wirkung (10) erreichen

es gilt: (10) = (3) $\wedge$ (9)

also: Pfadsensitivierung für UND:

erwartet für (10)	´(3)	´(9)	
1	1	1	
0	1	0	
0	0	1	

Fortführung des Beispiels

erwartet für (10)	´(3)	´(9)	Rekursiv weiter:
1	1	1	alle Wege zu „1“
0	1	0	
0	0	1	

wie erreicht man (3) = (9) = 1?

(3) ist Eingabe → trivial

(9) = (1) ∨ (2) = 1:

erw. für ´(9)	´(1)	´(2)
1	0	1
1	1	0

→

zusammen:

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

Fortführung des Beispiels

erwartet für (10)	´(3)	´(9)	
1	1	1	
0	1	0	alle Wege zu „0“ plus
0	0	1	einen beliebigen zu „1“

wie erreicht man (3) = 1 und (9) = 0?

(3) ist Eingabe → trivial

(9) = (1) ∨ (2) = 0:

erw. für ´(9)	´(1)	´(2)
0	0	0

→

zusammen:

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

Fortführung des Beispiels

erwartet für (10)	´(3)	´(9)	
1	1	1	
0	1	0	alle Wege zu „0“ plus
0	0	1	einen beliebigen zu „1“

wie erreicht man (3) = 0 und (9) = 1?

(3) ist Eingabe → trivial

(9) = (1) ∨ (2) = 1: nehme später **eine** der beiden Möglichkeiten!

erw. für ´(9)	´(1)	´(2)
1	0	1
1	1	0

→

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

Fortführung des Beispiels

(11) = ¬ (9) mit (9) = (1) ∨ (2)

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

(12) = ¬ (3)

erwartet für (12)	´(3)
1	0
0	1

6. Erstelle eine Entscheidungstabelle

	Regeln			
	1	2	3	...
Ursache 1 Ursache 2 ...				
Wirkung 1 Wirkung 2 ...				

→ es sollte keine widersprüchlichen Regeln geben
(Ursache-Teil gleich, Wirkungs-Teil verschieden)

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)				
	´(2)				
	`(3)				
W	´(10)				
	´(11)				
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1		
	´(2)	1	0		
	`(3)	1	1		
W	´(10)	1	1		
	´(11)				
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1	0	
	´(2)	1	0	0	
	`(3)	1	1	1	
W	´(10)	1	1	0	
	´(11)				
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1	0	0
	´(2)	1	0	0	1
	`(3)	1	1	1	0
W	´(10)	1	1	0	0
	´(11)				
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1	0	0
	´(2)	1	0	0	1
	`(3)	1	1	1	0
W	´(10)	1	1	0	0
	´(11)			1	
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1	0	0
	´(2)	1	0	0	1
	`(3)	1	1	1	0
W	´(10)	1	1	0	0
	´(11)	0	0	1	0
	´(12)				

Ergebnisse aufsammeln

´(9)	´(10)	´(1)	´(2)	`(3)
1	1	0	1	1
1	1	1	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
0	0	0	0	1

´(9)	´(10)	´(1)	´(2)	`(3)
1	0	0	1	0
1	0	1	0	0

erwartet für (11)	´(9)	´(1)	´(2)
1	0	0	0
0	1	0	1
0	1	1	0

erwartet für (12)	´(3)
1	0
0	1

		R1	R2	R3	R4
U	´(1)	0	1	0	0
	´(2)	1	0	0	1
	`(3)	1	1	1	0
W	´(10)	1	1	0	0
	´(11)	0	0	1	0
	´(12)	0	0	0	1

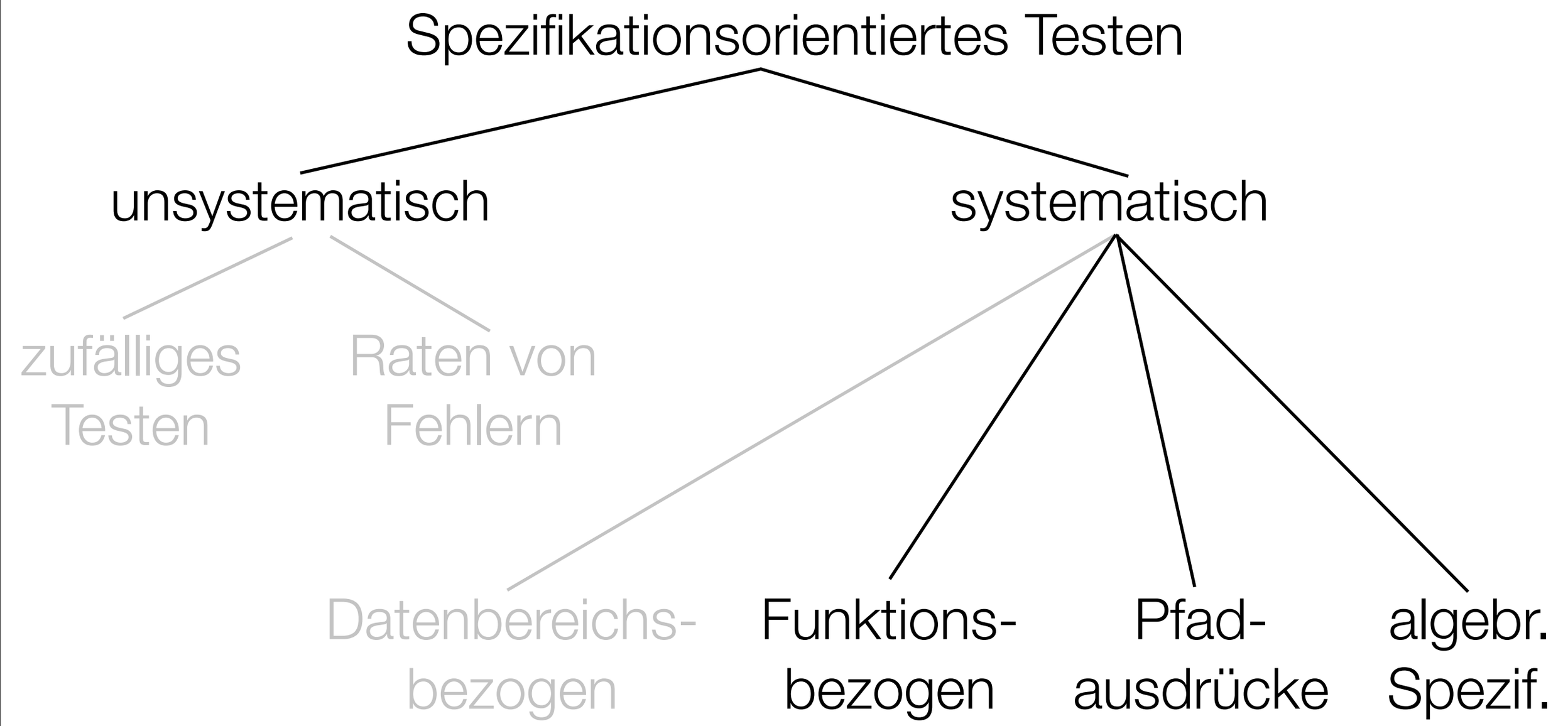
7. Ermittle zu jeder Regel einen Testfall

- möglichst Grenzfälle verwenden!

		R1	R2	R3	R4
	´(1)	0	1	0	0
U	´(2)	1	0	0	1
	`(3)	1	1	1	0
	´(10)	1	1	0	0
W	´(11)	0	0	1	0
	´(12)	0	0	0	1

R1: B9
R2: A0
R3: X0
R4: B$

Spezifikationsorientiertes Testen



Paradigmenwechsel

Ansatz zur Generierung von Testfällen:

- bisher: von der Spezifikation der Parameter ausgehend
- jetzt: von der Spezifikation der Funktionalität ausgehen

Testen von Reihenfolgebedingungen

Arten von Reihenfolgebedingungen

a) Gesamtfunktion $f = f_1,\ldots,f_n$

- Sequenzen als Spezialfall (siehe funktionsbezogenes Testen)
- Allgemeiner Fall:
  - Alternative Reihenfolgen können möglich sein

  → Spezifikation muß dann ein Reihenfolgeorakel enthalten

Testziele:

- sind alle Reihenfolgen erzielbar?
- sind keine unerlaubten Reihenfolgen möglich?

Arten von Reihenfolgebedingungen

b) Reihenfolgebedingungen auf Zugriffsfunktionen

- typisches Beispiel: Datenbanken, Gerätetreiber

Testziele:

- alle erlaubten Reihenfolgen fehlerfrei durchführbar?
- korrekte Fehlermeldungen bei unerlaubten Reihenfolgen?

Drei Ebenen der Reihenfolgebedingungen

1. spezifizierte Reihenfolgebedingungen
2. entworfene Reihenfolgebedingungen
3. implementierte Reihenfolgebedingungen

Ziel der nachfolgenden Technik:

➡ Übereinstimmung zwischen zweien dieser Reihenfolgebed. feststellen

(z.B. zwischen Entwurf und Programm)

Hilfsmittel: Pfadausdrücke

a) ein leerer Ausdruck und ein Funktionsname $f \in F$ sind Pfadausdrücke

b) Wenn P,Q Pfadausdrücke sind dann auch:

i) die Sequenz (P ; Q)

ii) die Alternative (P | Q)

iii) die Wiederholung P^+

iv) die Optionale Ausführung [P]

c) nichts anderes ist ein Pfadausdruck

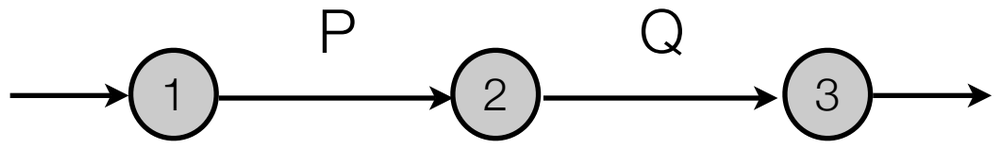
d) ; bindet stärker als | (Klammern einsparen)

Pfadausdrücke und endliche Automaten

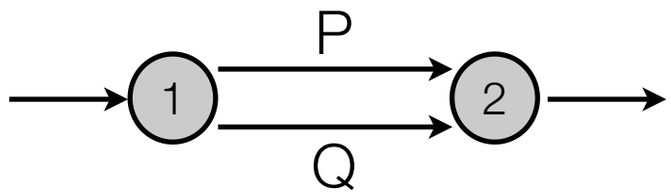
a) ein leerer Ausdruck und ein Funktionsname $f \in F$ sind Pfadausdrücke

b) Wenn P,Q Pfadausdrücke sind dann auch:

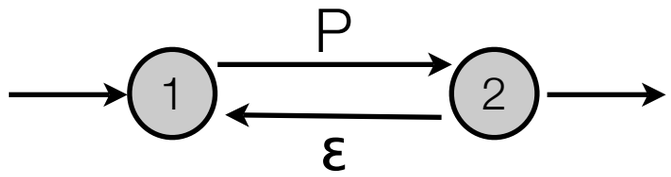
i) die Sequenz (P ; Q)



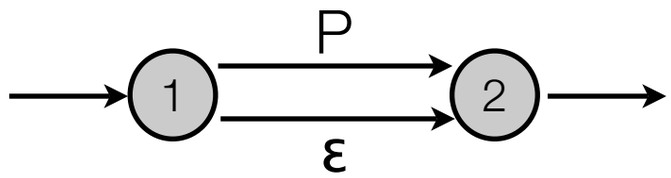
ii) die Alternative (P | Q)



iii) die Wiederholung P^+



iv) die Optionale Ausführung [P]



Konstruktion des endlichen Automaten

1. Nichtdeterministischen Automaten (NFA) mit ε-Sprüngen
   nach der Definition des Pfadausdrucks P konstruieren

2. ε-Sprünge eliminieren, z.B.
   - Ausgangsknoten von ε-Sprüngen in akz. Zustände selbst zu
     akz. Zuständen machen
   - ε-Sprünge als Schleifen am Zielknoten durch explizite Kanten auffalten

3. Automaten zum endlichen Automaten (DFA) A(P) reduzieren
   - überflüssige und äquivalente Knoten und Kanten eliminieren

4. DFA A(P) durch einen Fehlerzustand F vervollständigen

Beispiel: $P = (b \mid c)^+ ; [d ; e \mid f]$ (an der Tafel)

Testziel: Zeige Äquivalenz der Automaten

1. direkter Äquivalenzbeweis: möglich, aber aufwendig
2. „Alle Transitionen“-Kriterium
   - P sei Pfadausdruck über Menge F von Funktionssymbolen
   - A(P) der zugehörige endliche Automat
   - $T \subseteq F^*$ eine endliche Menge von endlichen Sequenzen

   T erfüllt das Kriterium „alle Transitionen“ ⇔

   für jede Kante k von A(P) gibt es in T eine Sequenz,
   in der k ausgehend vom Startzustand ausgeführt wird.

Bem.: später (Kontrollflußgraphen): stärkeres „alle Zweige“-Kriterium

Vergleich durchführen

- Vergleich zweier Automaten untereinander: einfach
- Vergleich Automat gegen Implementierung:
  - alle Anfangsstücke testen, die in akz. Zustand enden, z.B.:

    t_1: b, bb, bbc, bbcde
    t_2: c, cf

  - alle Wege in die Fehlerzustände testen

Einschränkungsmöglichkeiten

Falls erwiesen ist daß alle Fehlerfälle über eine Kante gleich behandelt werden

→ nur eine der Kanten stellvertretend testen

→ “alle korrekten / einige nicht korrekten Transitionen“-Kriterium

Weitere Hinweise

- möglichst Treibermodul / Skript schreiben
  das die Funktionen in den zu testenden Sequenzen ausführt

- Kriterien sind nur notwendige Testkriterien

  Bsp: Mutationsfehler im Pfadausdruck selbst

  statt P_1= ([a|b])$^+$; c wurde P_2=(a | b)$^+$; c realisiert

  → „alle Transitionen“-Folge „abc“ für P_1 wird auch von P_2 akzeptiert

  → Folge „c“ würde den Fehler aufdecken, obwohl sie nicht alle Transitionen ausführt.

Ende der heutigen Vorlesung

Danke fürs Zuhören!

Bis nächste Woche :-)