Dynamische Testtechniken

Black Box:

- Vorgehensweise
- Äquivalenzklassen
- Grenzwertanalyse
- Cause-Effect-Graphen

Dr. Christoph Steindl

Inhalt

- Begriffsfestlegung
  - Was ist Black-Box Testen?
  - Welche Fehler können gefunden werden?
  - Funktions-, Eingabe-, Ausgabeabdeckung

- Methoden und Techniken
  - Testspezifikation
  - Testfallgenerierung
  - Äquivalenzklassen
  - Grenzwertanalyse
  - Cause-Effect-Graphen

Was ist Black-Box-Testen?

- Black-Box Testen richtet sich nur nach der Spezifikation.
  Die Implementierung ist unbekannt.
  - **Testobjekt**: Prozedur/Methode, Klasse/Modul, Programm, System, ...
  - **Testfälle**: ausschließlich aus Spezifikation bzw. Schnittstelle des Testobjekts abgeleitet

- Vorteile
  - Tester muss sich nicht in die Implementierung einarbeiten
  - Tester ist unvoreingenommen (weil er die Implementierung nicht kennt)
- Nachteile
  - kann nie vollständig sein (unendlich viele Eingabekombinationen)
  - kann nicht feststellen, ob es unnötige Programmteile gibt
  - kennt fehleranfällige Programmteile nicht

Für welche Fehlerarten?

- Black-Box Testen versucht die folgenden Arten von Fehlern zu finden:
  - falsche oder fehlende Funktionalität
  - Schnittstellenfehler
    - falsche Parameterwerte, ...
  - Fehler in Datenstrukturen
    - falsche Sortierung in Listen, Einfügen/Löschen funktioniert nicht, ...
  - Fehler bei externen Datenbankzugriffen
    - liefert falsches Datum, liest falsches Feld aus, keine Verbindung, ...
  - Performanzfehler
    - zu langsam, zu wenig Durchsatz, ...
  - Initialisierungs- bzw. Terminierungsfehler
    - falsche Anfangswerte, Ressourcen werden nicht freigegeben, ...

Abdeckung

- Abdeckung aller Funktionen
  - jede spezifizierte Funktion wird in mindestens einem Testfall ausgeführt.
- Abdeckung aller Eingaben
  - für jede Funktion wird jeder Eingabewert in mindestens einem Testfall verwendet.
- Abdeckung aller Ausgaben
  - für jede Funktion wird jeder Ausgabewert in mindestens einem Testfall erzeugt.

```
class C {
  void f1 (int x);
  int f2 (boolean x, int y);
}
=> c.f1(…), c.f2(…), …
```

```
int f2 (boolean x, int y);
// x: false | true
// y: < 0 | 0 | > 0
=> f2(false, -1), f2(true, -1),
   f2(false,  0), f2(true,  0),
   f2(false,  3), f2(true,  3)
```

```
int f2 (boolean x, int y);
// return: == 0 | > 0
=> x = f2(false, 0)
   x = f2(true,  3)
```

Vollständigkeit

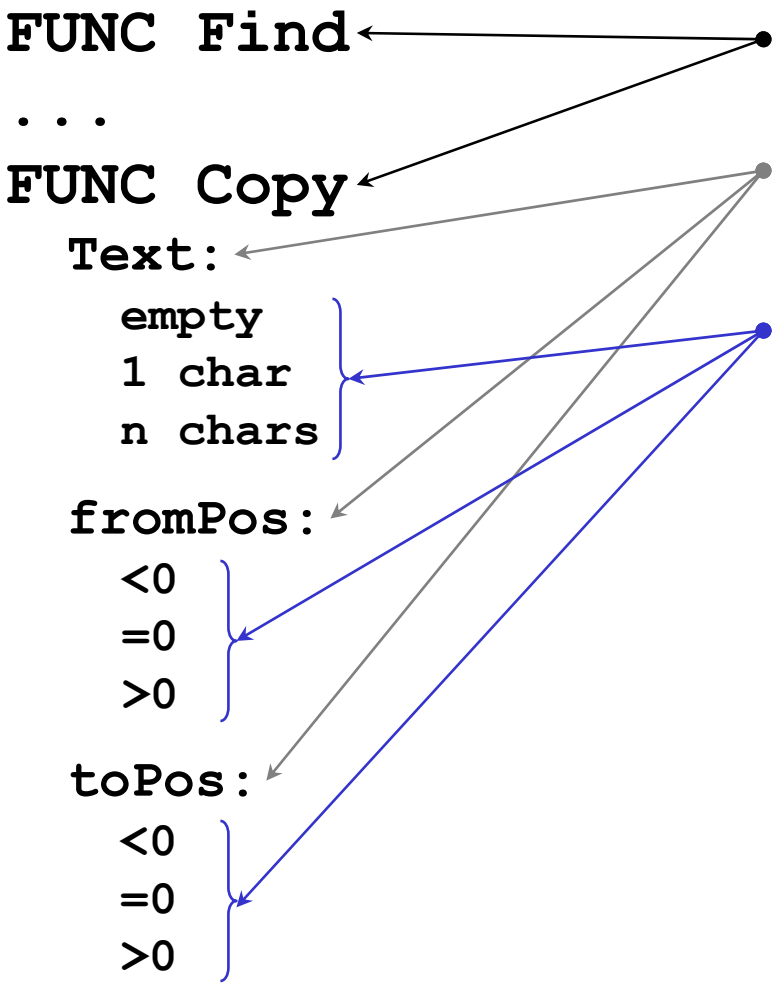
- Um **alle** Fehler zu finden,
  müsste man **alle** möglichen Eingaben testen.
- Alle möglichen Eingaben bestehen aus:
  - allen *gültigen* Eingaben (meist sehr, sehr viele)
  - allen *ungültigen* Eingaben (meist unendlich viele)
  - z.B. müsste ein Java-Compiler mit allen gültigen und ungültigen Java-Programmen getestet werden.

⇨ gezielte fehlerträchtige Testfälle auswählen!

Black-Box-Testen

- Vorgehensweise:
  - Black-Box-Testfälle ausgehend von der Spezifikation erstellen:
    - Eingabewerte auswählen
    - Erwartete Ergebnissen festlegen (!)
  - Testfälle ausführen
  - tatsächliche Ergebnisse mit den erwarteten vergleichen
- Ziele:
  - Fehlerträchtige Testfälle auswählen (die mit größter Wahrscheinlichkeit Fehler finden)
  - möglichst viel mit möglichst wenigen Testfällen testen
  - *trotzdem:* klare, verständliche und nachvollziehbare Testfälle

Vorgehensweise zum systematischen Erstellen von Testfällen



❶ finde zu testende **Funktionen**

❷ finde **Faktoren** von denen das Ergebnis der Funktion abhängt

❸ finde für jeden Faktor seine **Äquivalenzklassen**

❹ wähle Werte für Faktoren aus (ev. mit **Grenzwertanalyse**) und bilde **Testfälle** durch Kombination der Faktorwerte

	Text	fromPos	toPos
Testfall 1:	""	-1	-1
Testfall 2:	""	-1	0
Testfall 3:	""	0	-1
Testfall 4:	""	0	0
...	...	...	...
Testfall 27:	"Hello"	1	3

Systemat. Erstellen von Testfällen (1)

❶ Finde zu testende Funktionen

```
FUNC Copy , FUNC Paste, FUNC Find, ...
```

❷ Finde für jede Funktion alle Ein/Ausgabegrößen (= **Faktoren**)

- explizite: Parameter
- implizite: globale Variablen, Systemzustand, ...

```
FUNC Copy
    Factor text, Factor fromPos, Factor toPos, ...
FUNC Paste
    Factor text, Factor pos
...
```

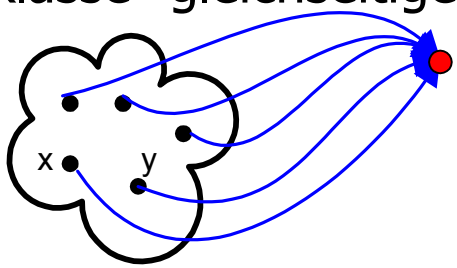
Systemat. Erstellen von Testfällen (2)

❸ Finde für jeden Faktor seine **Äquivalenzklassen**

```
Factor pos
   <0 , =0 , >0
```

Äquivalenzklassen

- sind Mengen von Werten, die das gleiche Ergebnis liefern
  - Bsp.: Dreiecks-Programm aus Einführung
    Äquivalenzklasse "gleichseitiges Dreieck"



	a	b	c
Fall 1	1	1	1
Fall 2	2	2	2
Fall 3	3	3	3
...	...	...	...

- nur **ein** Wert pro Äquivalenzklasse muss getestet werden!
  - (3,3,3) für alle gleichseitigen Dreiecke
  - (2,2,3), (2,3,2), (3,2,2) für alle gleichschenkligen Dreiecke
  - (2,4,3), ...(+Permutationen) für alle "normalen" Dreiecke
  - ...
- Achtung: eventuell behandelt das Programm doch nicht alle Werte der Äquivalenzklasse gleich!

Finden von Äquivalenzklassen

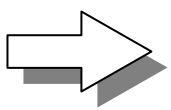
- Äquivalenzklasse hängt von der Art des Faktors (Parameters) ab:
  - *Wertebereich* (z.B. „Zähler im Bereich von 1 bis 99"):
    - 1 gültige ($1 \leq n \leq 99$)
    - 2 ungültige Klassen ($n < 1$, $n > 99$)
  - *Anzahl von Werten* (z.B. „1 bis 6 Eigentümer"):
    - 1 gültige (1 bis 6 Eigentümer)
    - 2 ungültige Klassen (keiner, mehr als 6)
  - *Menge von Werten* (z.B. „KFZ kann PKW, LKW oder Moped sein"):
    - 1 gültige Klasse je Alternative (z.B. PKW),
    - 1 ungültige Klasse für nicht genannte Möglichkeiten (z.B. Flugzeug)
  - *Muss-Bedingung* (z.B. „1. Zahl $\neq$ 0"):
    - 1 gültige (1. Zahl ist ungleich 0)
    - 1 ungültige Klasse (1. Zahl = 0)

Systemat. Erstellen von Testfällen (3)

❹ Wähle für jede Äquivalenzklasse einen Wert aus (entweder beliebig oder nach **Grenzwertanalyse**) und bilde **Testfälle** durch Kombination der Faktorwerte:

- Gültige Faktorwerte sollten mit allen anderen gültigen Faktorwerten kombiniert werden.
- Ungültige Faktorwerte (Fehlerfälle, Sonderfälle) sollten nicht mit anderen ungültigen kombiniert werden („Teste jeden Fehler für sich"). Es genügt, sie einmal zu testen.

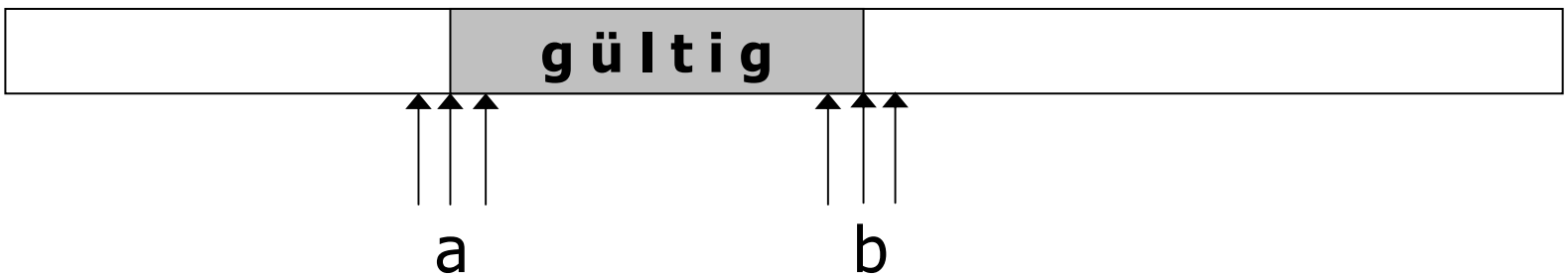
```
Factor A
   Value 1
   Value 2
Factor B
   Value 1
   Value 2
   Value 3
```



(A1,B1), (A1,B2), (A1,B3),
(A2,B1), (A2,B2), (A2,B3)

Grenzwertanalyse

- zur Erhöhung der Erfolgsquote der aus Äquivalenzklassen ausgewählten Werte
- statt irgendeines Wertes aus der Klasse nimmt man Grenzwerte, weil dort erfahrungsgemäß die größte Fehlerwahrscheinlichkeit liegt (Achtung: Heuristik!)



  - Testwerte: `a-1, a, a+1, b-1, b, b+1`

Beispiel: Find - Spezifikation

- Verbale Spezifikation einer Find-Funktion:

  Syntax:

  Find <pattern> <file>

  Funktion:

  Find sucht ein gegebenes Muster in einer Datei. Alle Zeilen, die das Muster enthalten, werden am Bildschirm ausgegeben. Wenn eine Zeile das Muster mehrfach enthält, wird sie trotzdem nur einmal ausgegeben.

  Das Muster ist eine beliebige Zeichenkette, die nicht länger als die max. Zeilenlänge der Sätze in der Datei sein darf. Wenn das Muster Leerzeichen enthält, muss es unter Hochkomma ("...") geschrieben werden. Ein Hochkomma im Muster wird durch Voranstellen eines umgekehrten Schrägstriches ("\") angegeben.

Beispiel: Find - Faktoren

❶ Funktionen: Find

❷ Faktoren:

- Eingabefaktoren:
  - Muster
    - Länge des Musters
    - Muster steht unter Hochkomma
    - Muster enthält Leerzeichen (in Hochkommas)
    - Muster enthält Hochkommas
  - Dateiname
- Ausgabefaktoren:
  - Anzahl der Zeilen mit Vorkommen
  - Anzahl der Vorkommen pro Zeile

Beispiel: Find - Äquivalenzklassen

❸ Äquivalenzklassen:

- Länge: = 0 , > 0 , > max. Zeilenlänge
- Muster in Hochkommas: ja , nein , falsche Hochkommas
- Blanks: ja , nein
- Hochkommas: ja , nein
- Dateiname: gültig , Datei nicht gefunden , fehlt

- Anzahl Vorkommen: 0 , 1 , > 1
- Vorkommen pro Zeile: 1 , > 1

❹ ⇨ ergibt $3 \cdot 3 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 2 = 648$ Kombinationen

Beispiel: Find – Testfallreduktion (1)

- aber: viele Kombinationen sind tatsächlich nicht möglich

  z.B. *Musterlänge*: 0 und *enthält Blanks*: ja

  ⇨ Einführen von

- Fehler- und Sonderfällen
  - nur **ein** Testfall pro Fehler- bzw. Sonderfall
    (darf nur mit anderen gültigen Faktorwerten kombiniert werden)

- Restriktionen durch
  - Definieren von **Eigenschaften**, die von bestimmten Wertebelegungen erfüllt werden, und
  - Aufstellen von **Bedingungen**, die erfüllt sein müssen, damit die Wertekombination für einen Testfall verwendet werden darf

Beispiel: Find – Testfallreduktion (2)

	Fehler-/ Sonderfall	Restriktionen	
		Bedingung	Eigenschaft
FUNCTION Find			
Länge:			
= 0	EINMAL	IF hochkomma	
> 0			muster = TRUE
> max	FEHLER		
Muster in HK:			
ja			hochkomma = TRUE
nein		IF muster	
falsch	FEHLER		
Blanks:			
ja	EINMAL	IF hochkomma	
nein			
HKs:			
ja	EINMAL	IF hochkomma	
nein			
...			

Beispiel: Find – Testfallreduktion (3)

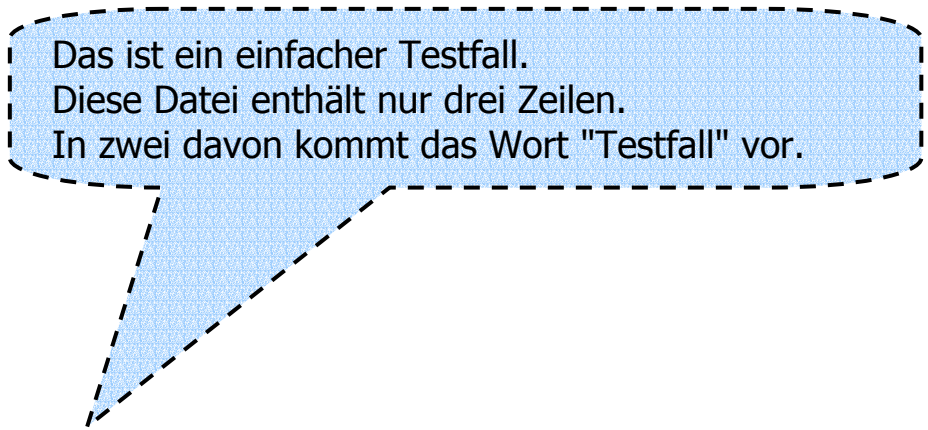
	Fehler-/ Sonderfall	Restriktionen
...		
Dateiname:		
gültig		datei = TRUE
nicht gefunden	FEHLER	
fehlt	FEHLER	
Anzahl Vorkommen:		
0	EINMAL	
1		IF muster & datei ; gefunden = TRUE
> 1	EINMAL	IF muster & datei ; gefunden = TRUE
Vorkommen pro Zeile:		
1		IF gefunden
> 1	EINMAL	IF gefunden

⇨ Reduktion auf **4+6+1·2·1·1·1·1·1 = 12** Testfälle
Fehler+Einmal+Kombinationen

Beispiel: Find - Testfälle

		Länge	*inHK*	*Blanks*	*HKs*	*Dateiname*	*Zeilen*	*inZeile*
Fehlerfälle	1	**> max**	nein	nein	nein	gültig	0	-
	2	> 0	**falsch**	nein	nein	gültig	0	-
	3	> 0	nein	nein	nein	**nicht gefunden**	0	-
	4	> 0	nein	nein	nein	**fehlt**	0	-
Sonderfälle	5	**0**	ja	nein	nein	gültig	0	-
	6	> 0	ja	**ja**	nein	gültig	1	1
	7	> 0	ja	nein	**ja**	gültig	1	1
	8	> 0	nein	nein	nein	gültig	**0**	-
	9	> 0	nein	nein	nein	gültig	**> 1**	1
	10	> 0	nein	nein	nein	gültig	1	**> 1**
	~~11~~	~~> 0~~	~~ja~~	~~nein~~	~~nein~~	~~gültig~~	~~1~~	~~1~~
	~~12~~	~~> 0~~	~~nein~~	~~nein~~	~~nein~~	~~gültig~~	~~1~~	~~1~~

Beispiel: Find - Testfall 9



- Kommando:

  Find Testfall Fall_09.txt

- Erwartetes Ergebnis:

  Das ist ein einfacher Testfall.

  In zwei davon kommt das Wort "Testfall" vor.

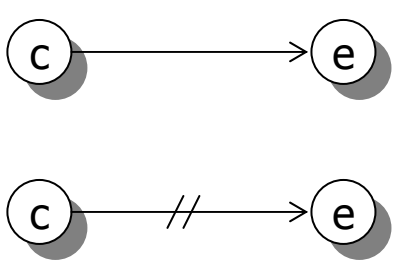
Cause-Effect-Graphen

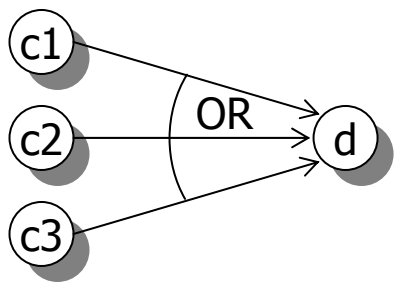
- Stellen Beziehungen zwischen Ursachen (causes) und Wirkungen (effects) dar

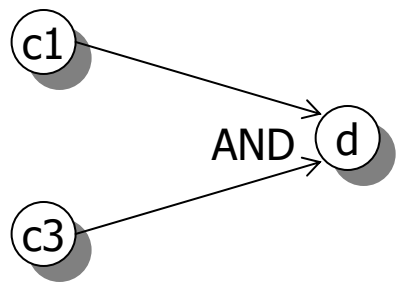
  cause = (Klasse von) Eingabebedingungen, Systemzustand

  effect = (Klasse von) Ausgabebedingungen,
  Aktionen (Veränderung des Systemzustandes)

- Elementare (logische) Verknüpfungen:





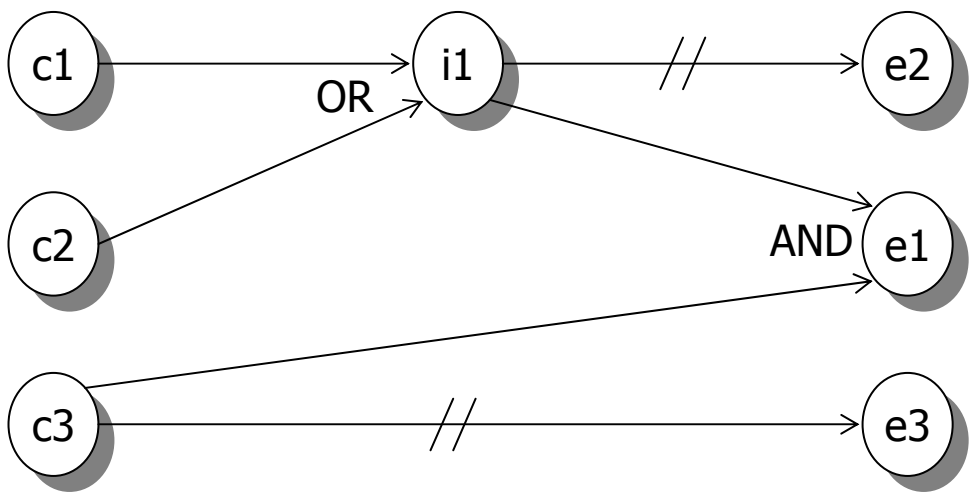


Beispiel-Cause-Effect-Graph

Spezifikation:

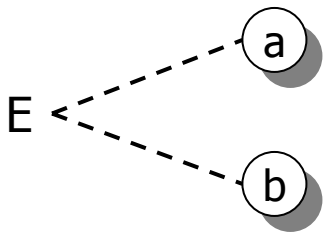
Das erste Zeichen muss ein "A" oder ein "B" sein. Das zweite Zeichen muss eine Ziffer sein. In dieser Situation wird die Datei aktualisiert. Wenn das erste Zeichen falsch ist, wird die Meldung *X1* ausgegeben. Wenn das zweite Zeichen falsch ist, die Meldung *X2*.

- Causes:
  - (c1) 1.Zeichen ist "A"
  - (c2) 1.Zeichen ist "B"
  - (c3) 2.Zeichen ist Ziffer
- Effects:
  - (e1) Datei aktualisiert
  - (e2) Fehlermeldung X1
  - (e3) Fehlermeldung X2

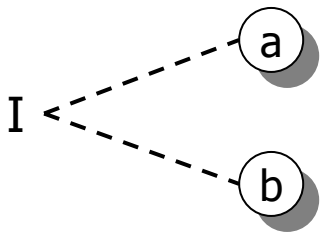


Einschränkungen (Constraints)

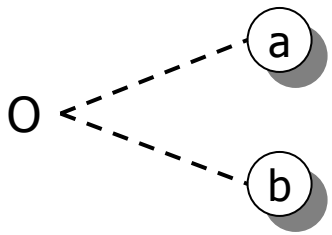
Exclusive:
max. 1 wahr



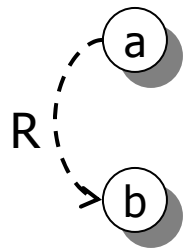
Inclusive:
min. 1 wahr



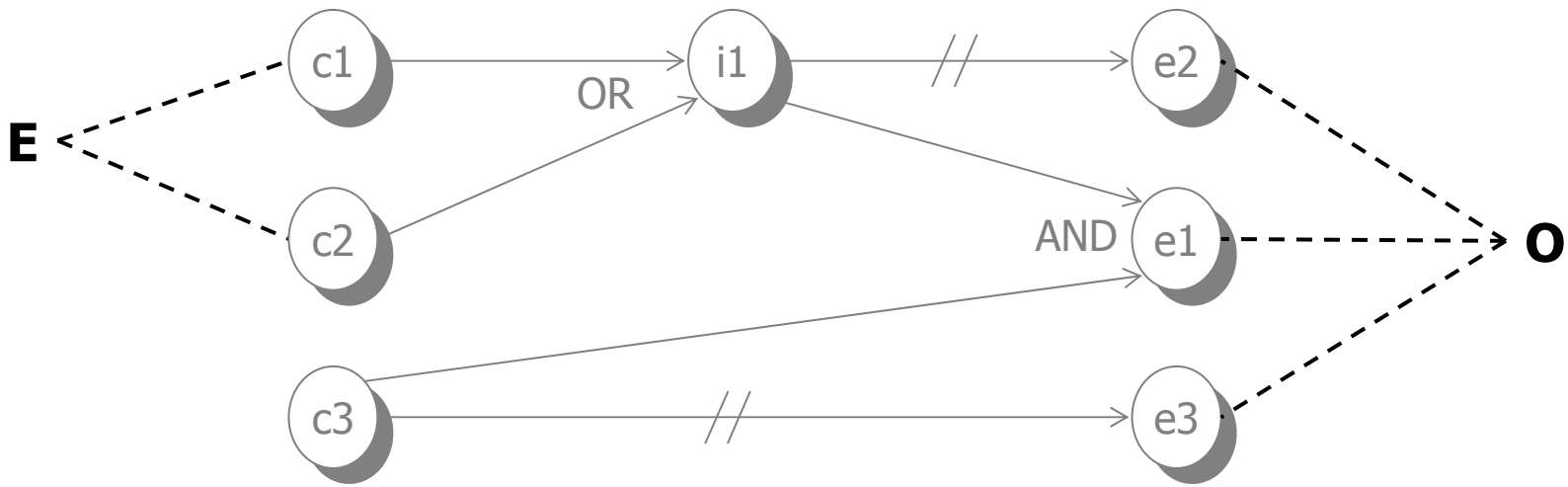
One and only one:
genau 1 wahr



Requires:
a nur wenn b



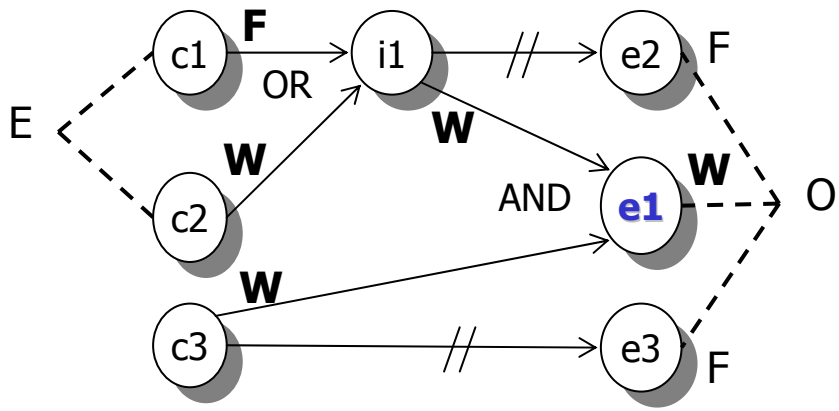
z.B.: das 1. Zeichen kann entweder ein "A" oder ein "B" sein, aber nicht beides, und sobald ein Fehler gefunden wird, wird abgebrochen.

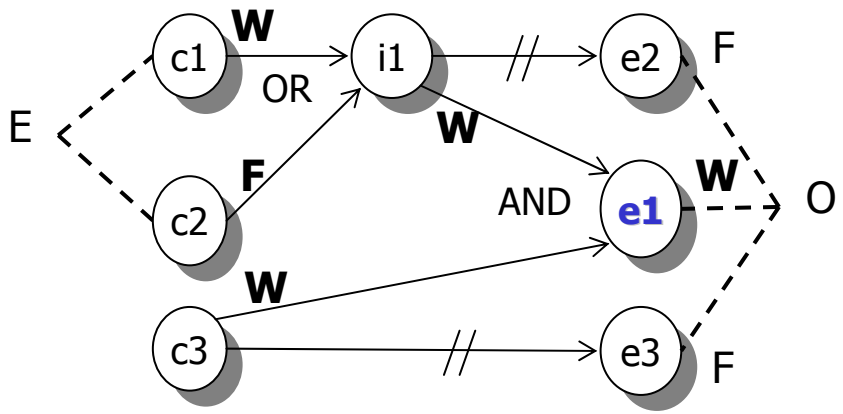


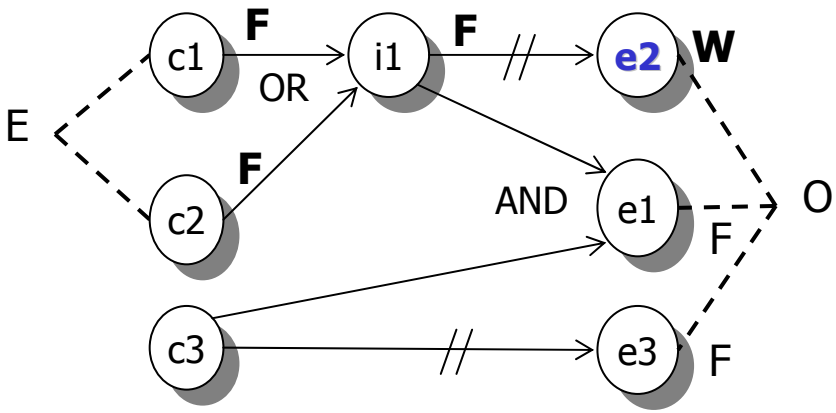
Testfälle mit C-E-Graphen finden (1)

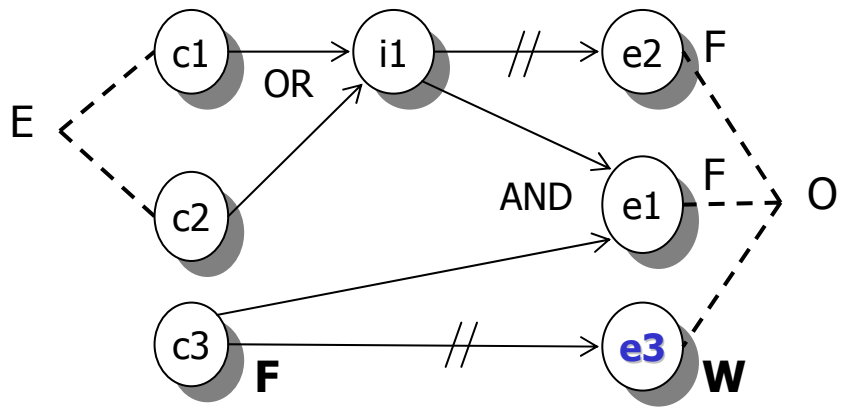
1. Unterteile die Spezifikation in "bearbeitbare" Einheiten,
   sonst wird der Graph zu komplex und unübersichtlich
2. Identifiziere die Ursachen und Wirkungen, und nummeriere sie durch.
3. Erstelle den Cause-Effect-Graphen mit Einschränkungen
4. Stelle eine Entscheidungstabelle auf
   cause als Bedingung, die erfüllt sein muss
   effect als Aktion, die dann ausgeführt wird
   $\Rightarrow$ im Graph vom effect aus rückwärts gehen
5. Jede Spalte der Entscheidungstabelle entspricht einem Testfall

Testfälle mit C-E-Graphen finden (2)









Testfälle mit C-E-Graphen finden (3)

		Testfall 1	Testfall 2	Testfall 3	Testfall 4
Eingabe	c1	Wahr (1.Zeichen "A")	Falsch (1.Zeichen kein "A")	Falsch (1.Zeichen kein "A")	---
	c2	Falsch (1.Zeichen kein "B")	Wahr (2. Zeichen "B")	Falsch (2.Zeichen kein "B")	---
	c3	Wahr (2.Zeichen Ziffer)	Wahr (2.Zeichen Ziffer)	---	Falsch (2.Zeichen keine Ziffer)
erwartetes Ergebnis	e1	Wahr (Datei aktualisiert)	Wahr (Datei aktualisiert)	Falsch	Falsch
	e2	Falsch	Falsch	Wahr (Meldung X1)	Falsch
	e3	Falsch	Falsch	Falsch	Wahr (Meldung X2)

Testfälle mit C-E-Graphen finden (4)