

# Softwarequalität

**Vorlesung 9 – Testen: Black-box-Verfahren  
(Äquivalenzklassenbildung, kombinatorisches Testen)**

*Prof. Dr. Joel Greenyer*



6. Juni 2016

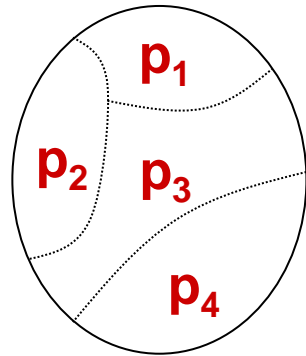
- Beispiel: Die Methode `getSeason` berechnet aus der Monatszahl die Jahreszeit
- **Annahme:** die Methode *verhält sich gleichartig* z.B. für die Eingaben 3,4,5
- Wenn ich also nicht alles testen kann (Testen kostet), dann macht es Sinn, dass ich mich darauf beschränke, einen Wert aus jeder Partition zu testen

```
/**
 * @param month number of month (must be number between 1 and 12)
 * @return "Spring" if month 3,4,5; "Summer" if month 6,7,8;
 * "Fall" if month 9,10,11; "Winter" if month 12,1,2;
 */
public static String getSeason(int month) {
    ...
}
```

- Wertebereich der Eingabe
  - $\text{month} \in \{n \mid n \in \mathbb{N} \wedge 1 \leq n \leq 12\}$ , andere int-Werte sind ungültig
- Partitionierung – hier als Menge von Teilmengen des Wertebereichs modelliert
  - $\{\{3, 4, 5\}, \{6, 7, 8\}, \{9, 10, 11\}, \{12, 1, 2\}\}$
- Tests z.B. (Eingabe  $\rightarrow$  Sollwert)
  - $3 \rightarrow \text{„Spring“}$ ;  $7 \rightarrow \text{„Summer“}$ ;  $9 \rightarrow \text{„Fall“}$ ;  $2 \rightarrow \text{„Winter“}$

```
/**
 * @param month number of month (must be number between 1 and 12)
 * @return "Spring" if month 3,4,5; "Summer" if month 6,7,8;
 * "Fall" if month 9,10,11; "Winter" if month 12,1,2;
 */
public static String getSeason(int month) {
    ...
}
```

- Bei der **Äquivalenzklassenbildung** muss eine Partitionierung  $P = \{p_1, p_2, \dots, p_n\}$  des Wertebereichs der Eingabe  $W$  gefunden werden.
- Die Idee: Wertebereich der Eingabe so aufteilen,
  - dass **zu vermuten ist**, dass sich Programm für Eingaben aus einer Teilmenge  $p_i$  **gleichartig verhält**
  - oder **vermutlich den gleichen Fehler auslöst**
- **Partitionierung** bedeutet immer
  - Die sich ergebenden Teilmengen müssen **disjunkt** sein
    - Also für alle  $p_i, p_j \in P$  gilt  $p_i \cap p_j = \emptyset$
  - Die sich ergebenden Teilmengen müssen **vollständig** sein, also zusammen den gesamten Wertebereich  $W$  ergeben
    - Also  $W = \bigcup_{i=1-n} p_i$





# Mehrdimensionale Äquivalenzklassenbildung

In der letzten Vorlesung...

- Besteht die Eingabe **aus mehreren Parametern**, oder kann ein Parameter **verschieden partitioniert** werden, ergibt sich eine **mehrdimensionale Äquivalenzklassenbildung**

- Beispiel Parkscheinautomat:

- **startDate**

1. nach 9:00 Uhr und vor 19:00 Uhr (zahlungspflichtig)
2. sonst

- **centsPaid**

1. < 50 c oder nicht teilbar durch 10 (ungültig)
2. >= 50 und <= 240 und teilbar durch 10 (gültig, „verfällt nicht“)
3. > 240 und teilbar durch 10 (gültig, „verfällt“)

startDate \ centsPaid	1.	2.
	1.	2.
1.	T11	T21
2.	T12	T22
3.	T13	T23

- Teilmengen T11, ..., T23 bilden wieder eine Partitionierung



# Mehrdimensionale Äquivalenzklassenbildung, weiteres Beispiel

---

- Es kann sein, dass es Sinn macht, Teilmengen aus einer mehrdimensionalen Äquivalenzklassenbildung wieder zu **verschmelzen**
  - Wenn zu vermuten ist, dass sich das Programm in allen Fällen gleichartig verhält
- Beispiel:
  - Das bessere von zwei Fußballteams wird primär anhand der Punkte (gewonnenen Spiele) ermittelt
  - Von zwei Teams ist das Team mit mehr Punkten der „Champion“
  - Haben zwei Teams gleiche viele Punkte, so ist das Team mit der höheren Anzahl Tore der „Champion“ (sekundäres Kriterium)
  - Ist auch die Anzahl der Tore gleich, gibt es keinen „Champion“
  - Sonst Rückgabe „null“



# Mehrdimensionale Äquivalenzklassenbildung, weiteres Beispiel

- Wie sieht hier eine gute Äquivalenzklassenbildung aus?

- Points:

1.  $t1.point < t2.points$
2.  $t1.point = t2.points$
3.  $t1.point > t2.points$

- Goals:

1.  $t1.goals < t2.goals$
2.  $t1.goals = t2.goals$
3.  $t1.goals > t2.goals$

```
public class Team {  
  
    int points;  
    int goals;  
  
    public static Team champion(Team team1,  
                                Team team2){  
        ...  
    }  
}
```

**Hinweis:** Hier geschieht die Partitionierung nicht auf Basis von zwei Eigenschaften, die sich getrennt auf die Parameter team1 und team2 beziehen, sondern auf Basis einer Relation zwischen den Parametern (deren Attributen).



# Mehrdimensionale Äquivalenzklassenbildung, weiteres Beispiel

- Wie sieht hier eine gute Äquivalenzklassenbildung aus?

- Points:

- $t1.point < t2.points$
- $t1.point = t2.points$
- $t1.point > t2.points$

- Goals:

- $t1.goals < t2.goals$
- $t1.goals = t2.goals$
- $t1.goals > t2.goals$

Zellen können verschmolzen werden – Annahme: Hier verhält sich das Programm gleichartig

Tabelle zeigt erwartete Rückgabewerte (SOLL-Werte):

goals \ points	1.	2.	3.
1.	t2	t2	t2
2.	t2	null	t1
3.	t1	t1	t1



goals \ points	1.	2.	3.
1.	t2		
2.	t2	null	t1
3.	t1		





# Abdecken von Anforderungen vs. Äquivalenzklassenbildung

---

- **Frage:** „Auch bei der Äquivalenzklassenbildung würde man ja schauen, dass verschiedene Äquivalenzklassen verschiedene Anforderungen abdecken... wie unterscheiden sich die Ansätze denn nun genau?“
- **Antwort 1:** Bei der Äquivalenzklassenbildung würden wir ggf. nicht nur versuchen, die Anforderungen abzudecken, sondern auch **andere Mutmaßungen** anstellen, wo sich das Programm andersartig verhalten könnte – und wo Entwickler vielleicht Fehler gemacht haben könnte.
  - Z.B. bei der Methode „findShortestPath(Node s, Node t)“: Hat der Entwickler an den Fall gedacht, dass t von s gar nicht erreichbar sein könnte?
  - Das erfordert natürlich Erfahrung, Gespür und Kreativität



# Abdecken von Anforderungen vs. Äquivalenzklassenbildung

---

- **Frage:** „Auch bei der Äquivalenzklassenbildung würde man ja schauen, dass verschiedene Äquivalenzklassen verschiedene Anforderungen abdecken... wie unterscheiden sich die Ansätze denn nun genau?“
- **Antwort 2:** Ja, Anforderungen (bzw. verschiedene genannte Fälle) können zu Äquivalenzklassen führen
- Aber:
  - Bei „alle Anforderungen abdecken“ reicht es, wenn **jede Anforderung** (bzw. jeder Fall) durch **einen** Testfall abgedeckt ist
  - Bei der hier vorgestellten Technik der mehrdimensionalen Äquivalenzklassenbildung würden **alle** Kombinationen von Äquivalenzklassen getestet („alle Zellen“ der Matrix)
    - außer es gibt begründeten Anlass, dass sich einige Zellen „verschmelzen“ lassen



# Abdecken von Anforderungen vs. Äquivalenzklassenbildung – Beispiel

- Angenommen ein Versicherungsbeitrag errechnet sich aus

- **Alter:**

- „Wenn 0-17 Jahre ...“ (R01)
- „Wenn 18-31 Jahre ...“ (R02)
- „Wenn 32-65 Jahre ...“ (R03)
- „Wenn über 65 Jahre ...“ (R04)

- **Geschlecht**

- „Wenn männlich ...“ (R05)
- „Wenn weiblich ...“ (R06)

- **Berufsgruppe**

- „Wenn in Gruppe 1 ...“ (R07)
- „Wenn in Gruppe 2 ...“ (R08)
- „Wenn in Gruppe 3 ...“ (R09)

**Frage:**

**Wie viele Tests ergeben sich**

**a) bei der Methode  
„Abdecken von Anforderungen“?**

**b) bei der Methode  
„Äquivalenzklassenbildung“?**

**(und warum?)**

```
public int calculateInsuranceRate(int age, Sex sex, Employment employment){  
    ...  
}
```



# Abdecken von Anforderungen vs. Äquivalenzklassenbildung – Beispiel

- Angenommen ein Versicherungsbeitrag errechnet sich aus

## – Alter:

- „Wenn 0-17 Jahre ...“ (R01)
- „Wenn 18-31 Jahre ...“ (R02)
- „Wenn 32-65 Jahre ...“ (R03)
- „Wenn über 65 Jahre ...“ (R04)

## – Geschlecht

- „Wenn männlich ...“ (R05)
- „Wenn weiblich ...“ (R06)

## – Berufsgruppe

- „Wenn in Gruppe 1 ...“ (R07)
- „Wenn in Gruppe 2 ...“ (R08)
- „Wenn in Gruppe 3 ...“ (R09)

## Äquivalenzklassenbildung:

- Aus den Anforderungen könnten wir nach den drei Kriterien ÄK (Partitionierungen) bilden
  - Alter (4 Klassen)
  - Geschlecht (2 Klassen)
  - Berufsgruppe (3 Klassen)
- Bei der mehrdimensionalen ÄK-Bildung würde eine dreidimensionale Matrix erstellt mit  $4 \times 2 \times 3$  Zellen
- Das bedeutet: **24 Testfälle**

```
public int calculateInsuranceRate(int age, Sex sex, Employment employment){  
    ...  
}
```



# Abdecken von Anforderungen vs. Äquivalenzklassenbildung – Beispiel

- Wie viele Testfälle bei „Abdecken von Anforderungen“?
  - **Minimalforderung**: Ein Testfall pro Anforderung
  - **Effizienzprinzip**: Ein Kreuz pro Spalte reicht

	Alter				Geschlecht		Berufsgruppe		
	R01	R02	R03	R04	R05	R06	R07	R08	R09
T1									
T2									
...									
...									
...									
...									



# Abdecken von Anforderungen vs. Äquivalenzklassenbildung – Beispiel

- Bei „Abdecken von Anforderungen“ reichen hier **vier Testfälle** um alle Anforderungen abzudecken
  - **Minimalforderung**: Ein Testfall pro Anforderung
  - **Effizienzprinzip**: Ein Kreuz pro Spalte reicht

	Alter				Geschlecht		Berufsgruppe		
	R01	R02	R03	R04	R05	R06	R07	R08	R09
T1	X				X		X		
T2		X				X		X	
T3			X		X				X
T4				X		X	X		

- Frage: Wieso vier?
  - Entspricht *Anzahl der Partitionen der größten ÄK-Partitionierung*

- Bei der mehrdimensionalen Äquivalenzklassenbildung steigt die Zahl der Äquivalenzklassen **exponentiell** mit der Anzahl der Klassifizierungskriterien
- Beispiel:
  - **errechnen des Versicherungsbeitrags**
    - 4 Altersstufen
    - 2 Geschlechter
    - 6 Berufsgruppen
    - 6 Arten von Vorerkrankungen (6 x ja/nein)

**Frage:** Wie viele Kombinationen gibt es?

**Antwort:**  
 $4 \cdot 2 \cdot 6 \cdot 2^6 = 3027$   
Kombinationen

- Das Testen aller Kombinationen möglicher Fälle kann sehr aufwendig werden
- Weiteres Beispiel:
  - Die Kompatibilität einer Webshop-Anwendung soll mit folgenden Systemkonfigurationen getestet werden
    - 6 Browser (IE, Firefox, Safari, ...)
    - 5 Betriebssysteme Client (Linux, OS X, Win 7, ...)
    - 2 Betriebssysteme Server (Linux, Win, ...)
    - 4 Datenbanken (MySQL, Oracle, ...)
    - 2 Server-CPU's (Intel, AMD)

$6 \cdot 5 \cdot 2 \cdot 4 \cdot 2 = 480$   
 Kombinationen





# Combinatorial Test Design (CTD)

## Interaktion von (Test-)Parametern

---

- Ist es nötig alle verschiedenen Kombinationen zu testen?
  - **Einerseits:** Mehr testen ist natürlich immer besser!
  - **Andererseits....**
- ...angenommen wir finden einen Fehler in der Webshop-Anwendung, der immer auftritt, wenn eine MySQL-Datenbank auf einem Windows Server installiert wird
  - Dann werden alle Testfälle mit dieser Kombination **denselben Fehler erzeugen** – das sind  $6 \cdot 5 \cdot 2 = 60$  Testfälle
- ...angenommen der Fehler tritt nur auf bei der Kombination MySQL + Windows Server + Nutzung des Safari-Browsers
  - Dann erzeugen immer noch  $5 \cdot 2 = 10$  Testfälle denselben Fehler
- Dies nennen wir **Level-2- bzw. Level-3-Interaktion** von (Test-)Parametern

- Statistische Untersuchungen zeigen
  - dass die Ursache von Fehlern meistens vom Wert **einer Variable** abhängt
  - Ein **Großteil der Fehler** kann gefunden werden, wenn **alle Interaktionen** von Werten **zweier Variablen** getestet werden (Level-2-Interaktion)

**Table 1. Number of variables involved in triggering software faults**

Vars	Medical Devices	Browser	Server	NASA GSFC	Network Security
1	66	29	42	68	20
2	97	76	70	93	65
3	99	95	89	98	90
4	100	97	96	100	98
5		99	96		100
6		100	100		

<http://csrc.nist.gov/groups/SNS/acts/ftfi.html>



# Combinatorial Test Design (CTD)

## Ansatz

- Das Ziel beim Combinatorial Test Design (CTD) (dt. auch **kombinatorisches Testen** oder **paarweises Testen**)
  - Eine möglichst kleine Menge an Testfällen soll 100% aller möglichen Interaktionen eines bestimmten Levels abdecken
- **Wie viele sind das?**
  - Angenommen, wir haben 20 Parameter mit jeweils fünf verschiedenen Werten (oder ÄKs)
    - Dann gibt es  $5^{20} = 95.367.431.640.625$  mögliche Kombinationen von Parametern
    - 4.750 Tests decken Level-2 ab (~Milliardstel Prozent von  $5^{20}$ )
    - 142.500 Tests, decken Level-3 ab
    - 3.028.125 decken Level-4 ab
    - 48.450.000 decken Level-5 ab (<Zehntausendstel Prozent von  $5^{20}$ )



# Combinatorial Test Design (CTD)

## Ansatz

- Das Ziel beim Combinatorial Test Design (CTD) (dt. auch **kombinatorisches Testen** oder **paarweises Testen**)
  - Eine möglichst kleine Menge an Testfällen soll 100% aller möglichen Interaktionen eines bestimmten Levels abdecken

Test	OS (Linux or Win)	DB (MySQL, Oracle or Sybase)	CPU (Intel, AMD)
T1	Linux	MySQL	Intel
T2	Linux	MySQL	AMD
T3	Linux	Oracle	Intel
T4	Linux	Oracle	AMD
T5	Linux	Sybase	Intel
T6	Linux	Sybase	AMD
T7	Windows	MySQL	Intel
T8	Windows	MySQL	AMD
T9	Windows	Oracle	Intel
T10	Windows	Oracle	AMD
T11	Windows	Sybase	Intel
T12	Windows	Sybase	AMD

- Das Ziel beim Combinatorial Test Design (CTD) (dt. auch **kombinatorisches Testen** oder **paarweises Testen**)
  - Eine möglichst kleine Menge an Testfällen soll 100% aller möglichen Interaktionen eines bestimmten Levels abdecken

Test	OS (Linux or Win)	DB (MySQL, Oracle or Sybase)	CPU (Intel, AMD)
T1	Linux	MySQL	Intel
T2	Linux	MySQL	AMD
T3	Linux	Oracle	Intel
T4	Linux	Oracle	AMD
T5	Linux	Sybase	Intel
T6	Linux	Sybase	AMD
T7	Windows	MySQL	Intel
T8	Windows	MySQL	AMD
T9	Windows	Oracle	Intel
T10	Windows	Oracle	AMD
T11	Windows	Sybase	Intel
T12	Windows	Sybase	AMD

Alle Level-2-Interaktionen abgedeckt?

Schon für kleinen Konfigurationsraum schwierig, es durch „Rumprobieren“ richtig zu machen

- Das Ziel beim Combinatorial Test Design (CTD) (dt. auch **kombinatorisches Testen** oder **paarweises Testen**)
  - Eine möglichst kleine Menge an Testfällen soll 100% aller möglichen Interaktionen eines bestimmten Levels abdecken

Test	OS (Linux or Win)	DB (MySQL, Oracle or Sybase)	CPU (Intel, AMD)
T1	Linux	MySQL	Intel
T2	Linux	MySQL	AMD
T3	Linux	Oracle	Intel
T4	Linux	Oracle	AMD
T5	Linux	Sybase	Intel
T6	Linux	Sybase	AMD
T7	Windows	MySQL	Intel
T8	Windows	MySQL	AMD
T9	Windows	Oracle	Intel
T10	Windows	Oracle	AMD
T11	Windows	Sybase	Intel
T12	Windows	Sybase	AMD

Jetzt müsste es stimmen...



# Combinatorial Test Design (CTD)

## Ansatz

---

- Das Ziel beim Combinatorial Test Design (CTD) (dt. auch **kombinatorisches Testen** oder **paarweises Testen**)
  - Eine möglichst kleine Menge an Testfällen soll 100% aller möglichen Interaktionen eines bestimmten Levels abdecken
- **Wie erzeugen wir diese Menge von Tests?**
  - **Tests** meinen im Folgenden die **Kombinationen von Eingabeparametern** – zum eigentlichen Test fehlen dann die Sollwerte
- Es gibt verschiedene Ansätze
  - **Algebraische Methoden**: Effizient, aber können schlecht Einschränkungen auf Eingabekombinationen berücksichtigen
  - **Greedy-Algorithmen**: Basieren auf Heuristiken, nicht optimal
  - **Heuristische Verfahren**: z.B. Generische Algorithmen, „Simulated Annealing“, Konvergieren zum Optimum

- **IPOG-Algorithmus** (IPOG = **I**n-**P**arameter-**O**rders-**G**eneral)
- Sei  $t$  das abzudeckende Interaktionslevel (z.B. Level-3 :  $t=3$ )
- **Idee:**
  1. Betrachte die ersten  $t$  Parameter und erstelle eine Testmenge mit 100%iger Level- $t$ -Abdeckung
  2. Dann erweitere diese Menge sukzessive:
    - **Horizontale Ausdehnung:** Jeder existierende Test wird für den nächsten noch nicht betrachteten Parameter erweitert
    - **Vertikale Ausdehnung:** Weitere Tests werden hinzugefügt, wenn notwendig um weitere  $t$ -Tupel abzudecken

aus der Vorlesung *Software Testing*, von Itai Segall, Tel Aviv University 2013/14



- Beispiel:
  - 4 Parameter: P1, P2, P3 P4
  - Wertebereich von P1, P2, P3 ist {0,1},
  - Wertebereich von P4: {0, 1, 2}
  - $t = 3$
- Schritt 1: Forme initiale  
Eingabe-Tupel
  - Hier: Alle 3-Tupel der  
Parameter P1, P2, P3:

	P1	P2	P3
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Schritt 2: Horizontale Ausdehnung
  - Betrachte nächsten Parameter P4
  - Berechne alle 3-Tupel mit vorigen Parametern, also
    - $P1 \times P2 \times P4 \cup$
    - $P1 \times P3 \times P4 \cup$
    - $P2 \times P3 \times P4$
  - Diese Menge dieser Tupel nennen wir  $\pi$ 
    - Wie groß ist  $\pi$ ?
    - Hier  $|\pi| = (2 \times 2 \times 3) \times 3 = 36$
  - Erweitere Eingabe-Tupel „horizontal“ mit Werten aus P4
    - Sodass möglichst viele Tupel aus  $\pi$  abgedeckt sind

	P1	P2	P3	P4
0	0	0	0	?
0	0	0	1	?
0	1	0	0	?
0	1	1	1	?
1	0	0	0	?
1	0	1	1	?
1	1	0	0	?
1	1	1	1	?

- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P2	P3	P4
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Nehmen wir hier z.B. „0“, dann haben wir drei Tupel abgedeckt (mehr geht nicht)

Tupel aus  $\pi$  in Matrixform:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2


P1	P2	P3	P4
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 ?
0 1 0 ?
0 1 1 ?
1 0 0 ?
1 0 1 ?
1 1 0 ?
1 1 1 ?



- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

	P1	P2	P3	P4
0	0	0	0	0
0	0	1	1	1
0	1	0	?	?
0	1	1	?	?
1	0	0	?	?
1	0	1	?	?
1	1	0	?	?
1	1	1	?	?




- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 1
0 1 0 2
0 1 1 ?
1 0 0 ?
1 0 1 ?
1 1 0 ?
1 1 1 ?




- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 1
0 1 0 2
0 1 1 0
1 0 0 ?
1 0 1 ?
1 1 0 ?
1 1 1 ?






- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 1
0 1 0 2
0 1 1 0
1 0 0 1
1 0 1 ?
1 1 0 ?
1 1 1 ?




- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 1
0 1 0 2
0 1 1 0
1 0 0 1
1 0 1 2
1 1 0 ?
1 1 1 ?




- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1 P2 P3 P4
0 0 0 0
0 0 1 1
0 1 0 2
0 1 1 0
1 0 0 1
1 0 1 2
1 1 0 0
1 1 1 ?




- Wie erweitern wir horizontal und decken dabei möglichst viele Tupel in  $\pi$  ab?

Tupel aus  $\pi$  in Matrixform:

P1 P2 P4	P1 P3 P4	P2 P3 P4
0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1
0 0 2	0 0 2	0 0 2
0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1
0 1 2	0 1 2	0 1 2
1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1
1 0 2	1 0 2	1 0 2
1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1
1 1 2	1 1 2	1 1 2

P1	P2	P3	P4
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	0
1	0	0	1
1	0	1	2
1	1	0	0
1	1	1	1



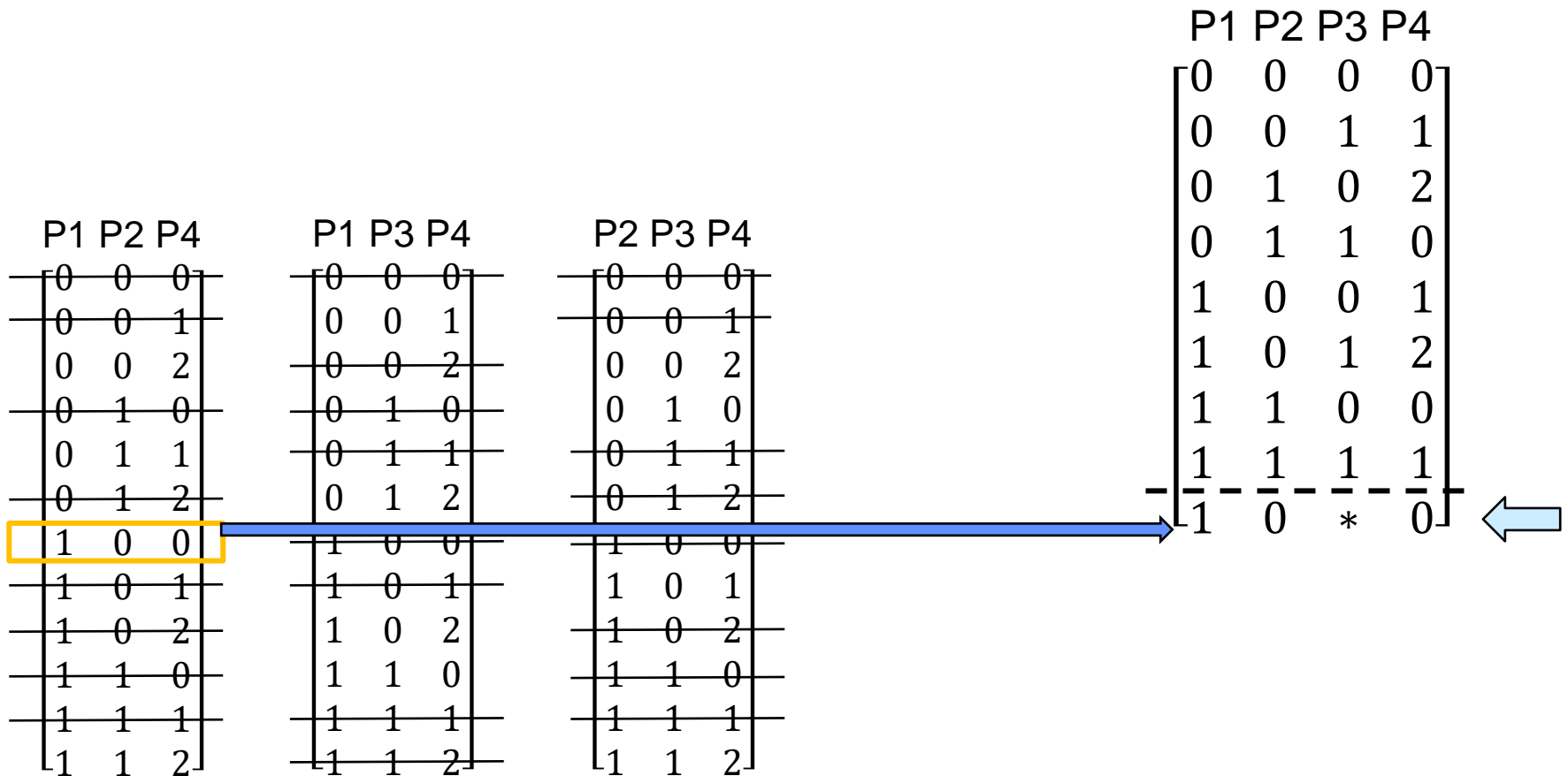
- Schritt 3: Vertikale Ausdehnung
  - Für alle t-Tupel aus  $\pi$ , die bisher nicht abgedeckt sind:
  - Füge einen Test hinzu, der ein übriges t-Tupel aus  $\pi$  abdeckt
    - Dabei haben die horizontal erweiterten Tests mehr Werte als die t-Tupel aus  $\pi$
    - An einigen Stellen können wir uns also Werte ausdenken
    - Bzw. erstmal „egal“ oder „\*“ reinschreiben

	P1	P2	P3	P4
	0	0	0	0
	0	0	1	1
	0	1	0	2
	0	1	1	0
	1	0	0	1
	1	0	1	2
	1	1	0	0
	1	1	1	1
	–	–	–	–

- Schritt 3: Vertikale Ausdehnung
  - Für alle t-Tupel aus  $\pi$ , die bisher nicht abgedeckt sind:
  - Füge einen Test hinzu, der ein übriges t-Tupel aus  $\pi$  abdeckt
    - Dabei haben die horizontal erweiterten Tests mehr Werte als die t-Tupel aus  $\pi$
    - An einigen Stellen können wir uns also Werte ausdenken
    - Bzw. erstmal „egal“ oder „\*“ reinschreiben
  - Manchmal es möglich, statt einen neuen Test für ein übriges t-Tupel in  $\pi$  hinzuzufügen, einen Test mit „\*“ Werten anzupassen

	P1	P2	P3	P4
	0	0	0	0
	0	0	1	1
	0	1	0	2
	0	1	1	0
	1	0	0	1
	1	0	1	2
	1	1	0	0
	1	1	1	1
←	1	0	*	0

- Schritt 3 nochmal im Detail:



- Schritt 3 nochmal im Detail:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P2	P3	P4
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	0
1	0	0	1
1	0	1	2
1	1	0	0
1	1	1	1
1	0	1	0

Hier kann bestehender Eintrag angepasst werden



- Schritt 3 nochmal im Detail:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

$$\begin{array}{c}
 \text{P1 P2 P3 P4} \\
 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\
 - \left[ \frac{1}{1} - \frac{1}{0} - \frac{1}{1} - \frac{1}{0} \right] \quad \leftarrow
 \end{array}$$


- Schritt 3 nochmal im Detail:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

	P1	P2	P3	P4
	0	0	0	0
	0	0	1	1
	0	1	0	2
	0	1	1	0
	1	0	0	1
	1	0	1	2
	1	1	0	0
	1	1	1	1
	1	1	1	1
-	1	0	1	0
-	0	1	*	1



- Schritt 3 nochmal im Detail:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

$$\begin{array}{c}
 \text{P1 P2 P3 P4} \\
 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\
 - \left[ \frac{1}{0} - \frac{1}{1} - \frac{1}{0} - \frac{1}{1} \right] \quad \leftarrow
 \end{array}$$

- Schritt 3 nochmal im Detail:

P1	P2	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P1	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

P2	P3	P4
0	0	0
0	0	1
0	0	2
0	1	0
0	1	1
0	1	2
1	0	0
1	0	1
1	0	2
1	1	0
1	1	1
1	1	2

$$\begin{array}{c}
 \text{P1 P2 P3 P4} \\
 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\
 - \left[ \frac{1}{0} - \frac{1}{1} - \frac{1}{0} - \frac{1}{1} \right] \leftarrow
 \end{array}$$

... usw.

- **IPOG-Algorithmus** (IPOG = **I**n-**P**arameter-**O**rders-**G**eneral)
- Sein PS (Parameter Set) eine Menge von Parametern  $P_1, P_2, \dots, P_k$

Algorithmus IPOG(int  $t$ , ParameterSet  $ps$ )

1. Initialisiere die Testmenge  $ts$  als leere Menge
2. Sortiere die Parameter von  $ps$  absteigend nach der Größe ihres Wertebereichs (Parameter mit großem Wertebereich zuerst)
3. Für jede Kombination von Werten der ersten  $t$  Parameter von  $ps$  füge zu  $ts$  einen Test hinzu
4. **for**(int  $i=t+1$ ;  $i \leq k$ ;  $i++$ )
  1. **ExtendPlusOne**( $ts, I, t, ps$ )

Optimierung kann oft vertikale Ausdehnung vermeiden

i ist bei jedem Aufruf  
um eins größer

Methode **ExtendPlusOne**(TestSet **ts**, int **i**, int **t**, ParameterSet **ps**)

1. Erzeuge eine Menge  $\pi$  von allen **t**-Tupeln mit Werten von **P<sub>i</sub>** und Werten von **P<sub>1</sub>**, ..., **P<sub>i-1</sub>**.

(z.B.  $t=3, i=4: \pi = P_1 \times P_2 \times P_4 \cup P_1 \times P_3 \times P_4 \cup P_2 \times P_3 \times P_4$ )

// *Horizontale Ausdehnung*

2. Für jeden Test **T**=(**v<sub>1</sub>**, **v<sub>2</sub>**, ..., **v<sub>i-1</sub>**) in **ts**
  1. Wähle einen Wert **v<sub>i</sub>** für **P<sub>i</sub>** und ersetze **T** mit **T'**=(**v<sub>1</sub>**, **v<sub>2</sub>**, ..., **v<sub>i-1</sub>**, **v<sub>i</sub>**) sodass **T'** die meisten Kombinationen vom Werten in  $\pi$  abdeckt.
  2. Entferne aus  $\pi$  alle durch **T'** abgedeckten Tupel.

// *Vertikale Ausdehnung*

3. Für jedes **t**-Tupel **c** in  $\pi$ 
  1. Gibt es in **ts** ein Tupel mit „\*-Werten, das angepasst werden kann um **c** abzudecken:
    1. Passe diesen Eintrag an
  2. Sonst:
    1. Erzeuge einen neuen Eintrag, um **c** abzudecken (Wo Werte nicht durch **c** definiert, „\*“ eintragen)

- Wie kann der Algorithmus Randbedingungen berücksichtigen?
  - Randbedingungen: Bestimmte Kombinationen von Eingaben sind unzulässig
- Den Algorithmus an zwei Stellen erweitern
  - Bei Schritt 3 von IPOG (der Initialen Befüllung von  $ts$ ) nur zulässige Kombinationen der ersten  $t$  Parameter hinzufügen
  - Bei der Erzeugung der Menge  $\pi$  in ExtendPlusOne nur zulässige Kombinationen hinzufügen