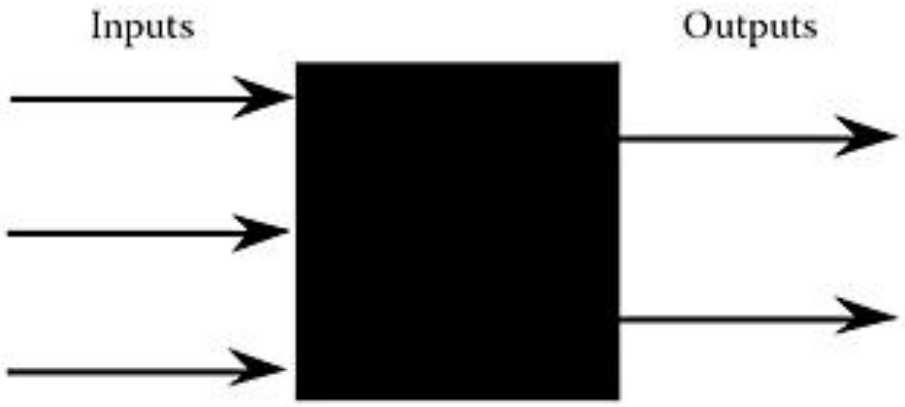
Tehnike crne kutije (funkcionalno testiranje)

Funkcionalno testiranje

- Program se posmatra kao funkcija koja mapira vrednosti iz ulaznog domena u izlazni. Implemenacija nije poznata, program predstavlja "crnu kutiju".
- Jedina informacija koja se koristi za određivanje test primera je specifikacija programa.



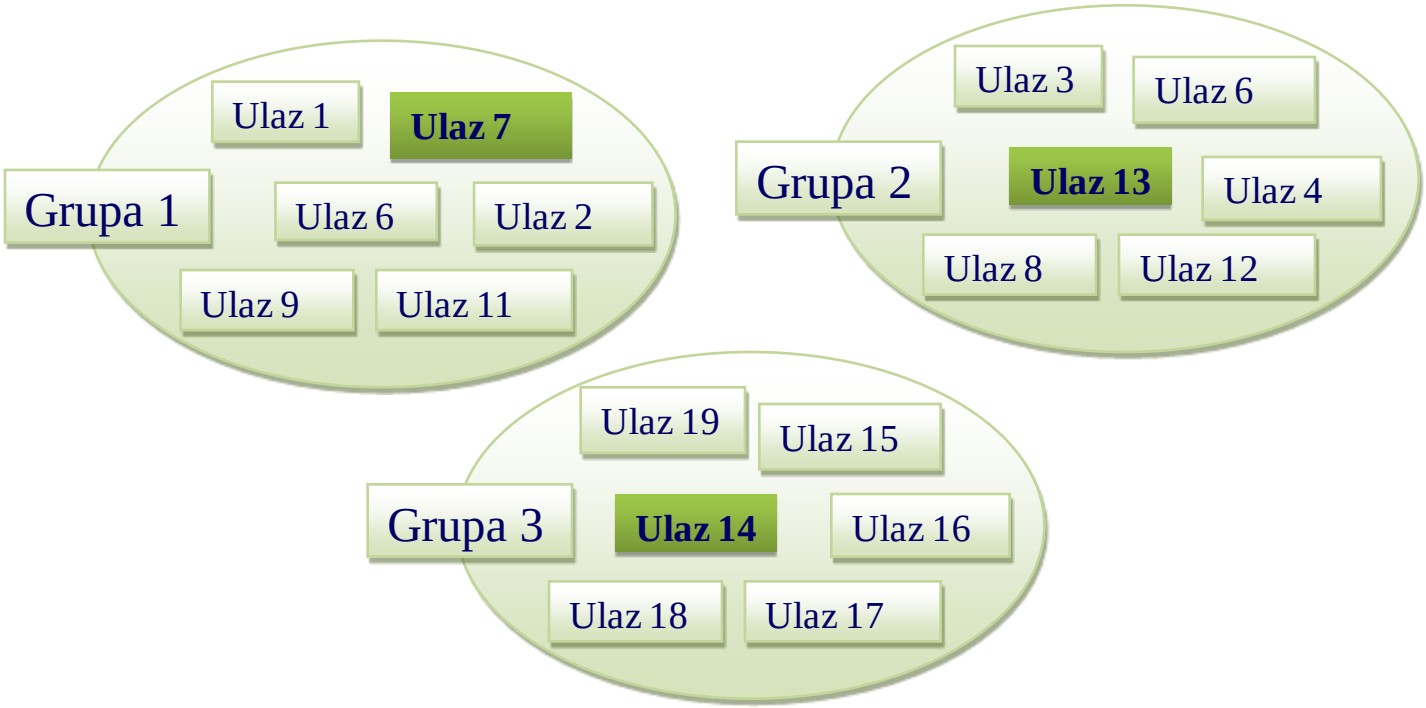
Testiranje tehnikama crne kutije

- Podela na klase ekvivalencije (klasična definicija)
- Analiza graničnih vrednosti
- Testiranje zasnovano na tabeli odlučivanja
  - Uzročno-posledični graf
- Testiranje zasnovano na modelu stanja
- Testiranje sintakse
- Kombinatorno testiranje
  - zasnovano na ortogonalnim nizovima

Podela na klase ekvivalencije

- Ulazne vrednosti programa se dele u klase ekvivalencije.
- Podela se vrši tako da važi:
  - program se ponaša na sličan način za sve ulazne vrednosti koje pripadaju istoj klasi ekvivalencije

Klase ekvivalencije



Grupe međusobno ekvivalentnih ulaznih podataka proizvode isti rezultat iz kojih se bira po jedan predstavnik grupe koji se koristi u tokom testiranja.

Zašto definišemo klase ekvivalencije?

- Da bismo testirali program sa po jednom reprezentativnom vrednošću ulaza iz svake klase ekvivalencije
  - Jednako delotvorno kao testiranje bilo kojim drugim vrednostima iz istih klasa ekvivalencije (naći će iste greške).
- U idealnom slučaju podskupovi su međusobno disjunktni i pokrivaju ceo skup ulaza (relacija "sličnosti" ulaza je relacija ekvivalencije)

Podela na klase ekvivalencije

- Kako odrediti klase ekvivalencije?
  - Posmatraju se svi uslovi vezani za ulaze i izlaze programa koji proizilaze iz specifikacije (tipovi promenljivih, eksplicitna ograničenja).
  - Za svaki uslov se posmatraju dva grupe klasa prema zadovoljenosti uslova:
    - legalne klase obuhvataju dozvoljene situacije.
    - nelegalne klase obuhvataju sve ostale situacije.

Saveti za podelu na klase ekvivalencije

- Ako je ulazni uslov programa definisan u opsegu vrednosti:
  - Na primer, broj između 1 i 5000.
  - Definišu se jedna legalna ($1 \leq broj \leq 5000$) i dve nelegalne klase ekvivalencije: (broj < 1) i (broj > 5000)

Saveti za podelu na klase ekvivalencije

- Ako ulazni uslov programa definiše neki fiksni broj:
  - Na primer, mat.br. građana ima 13 cifara.
  - Definišu se jedna legalna (mbr_c=13) i dve nelegalne klase ekvivalencije:
    (mbr_c < 13) i (mbr_c > 13)

Saveti za podelu na klase ekvivalencije

- Ako ulazni podatak uzima vrednosti iz nabrojivog skupa, pri čemu se program različito ponaša za svaku vrednost:
  - Na primer, {a,b,c}
  - Po jedna klasa za svaku dozvoljenu vrednost ulaza (tri klase)
  - Jedna klasa za ulazne vrednosti van dozvoljenog skupa (npr. d).

Table 2.1 Guidelines for generating equivalence classes for variables: range and strings

		Example	
Kind	**Equivalence classes**	**Constraint**	**Class representatives**[a]
Range	One class with values inside the range and two with values outside the range	$speed \in [60 \ldots 90]$	{{50}↓,{75}↑, {92}↓}
		area : float;	{{−1.0}↓,{15.52}↑}
		$area \geq 0$	
		age : int;	{{−1}↓,{56}↑, {132}↓}
		$0 \leq age \leq 120$	
		letter:char;	{{J}↑,{3}↓}

String	At least one containing all legal strings and one containing all illegal strings. Legality is determined based on constraints on the length and other semantic features of the string	*fname*: *string*;	{{ϵ}↓,{Sue}↑, {Sue2}↓,{Too Long a name}↓}
		vname: string;	{{ϵ}↓,{shape}↑, {address1}↑}, {Long variable}↓

Table 2.2 Guidelines for generating equivalence classes for variables: Enumeration and arrays

Kind	Equivalence classes	Example[a] Constraint	Example[a] Class representatives[b]
Enumeration	Each value in a separate class	*auto_color* ∈ {red, blue, green} *up*: boolean	{{red}↑, {blue}↑, {green}↑} {{true}↑, {false}↑}
Array	One class containing all legal arrays, one containing only the empty array, and one containing arrays larger than the expected size	*Java array*: int[] *aName* = new int[3];	{{[]}↓, {[−10, 20]}↑, {[−9, 0, 12, 15]}↓}

Saveti za podelu na klase ekvivalencije

- Za složene tipove podataka (zapise)
  - Odrediti kl. ekv. za svaku komponentu posebno, po potrebi razmotriti kombinacije klasa ekvivalencije

Podela na klase ekvivalencije

- Generalno, ako postoji sumnja da se program ne ponaša isto za svaki element već definisane klase ekvivalencije, treba klasu razbiti na više manjih.

Multidimenzionalno particioniranje

- Ako razmatramo svaki uslov izolovano od drugih i definišemo njegove klase ekv. Reč je o tzv. **jednodimenzionalnom particionisanju**
- Ako razmatramo kombinacije klasa ekvivalencije parova ili većeg broja uslova, radi se **multidimenzionalno particionisanje**

Multidimenzionalno particioniranje (primer)

- Razmotrimo program koji prima dva cela broja x i y, uz ograničenja:

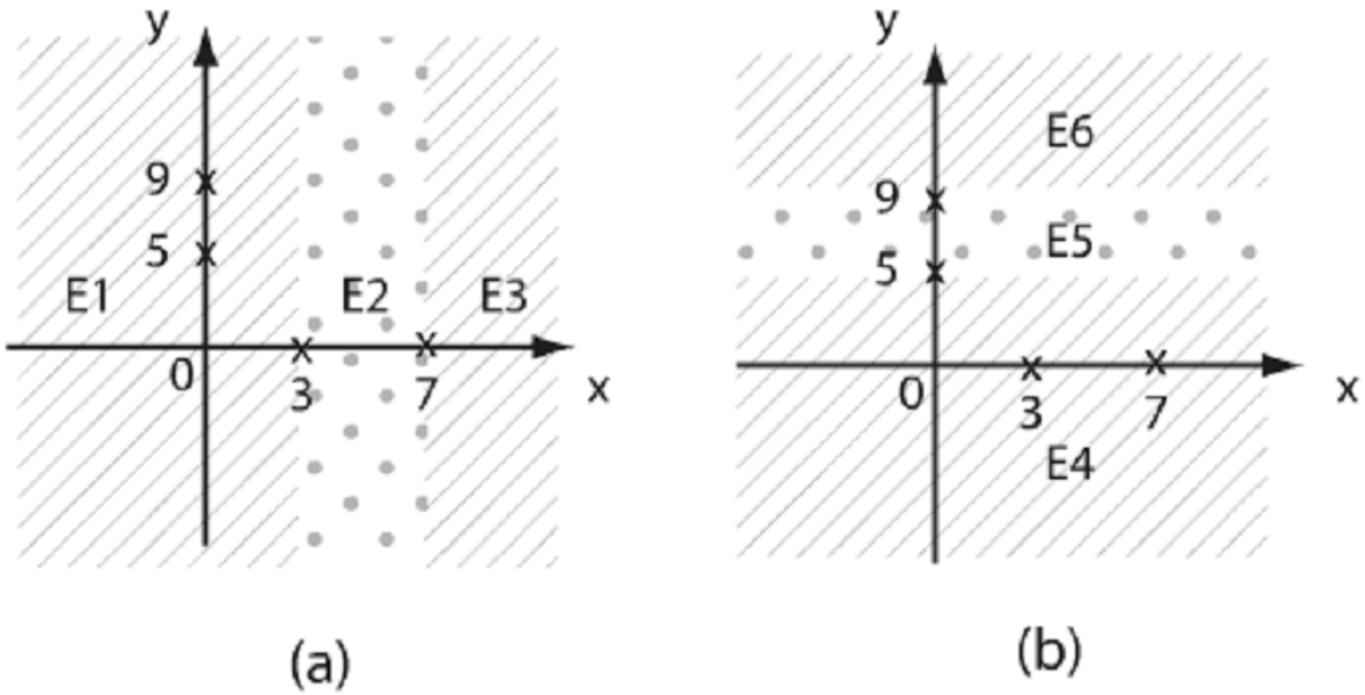
  $3 \leq x \leq 7 \qquad 5 \leq y \leq 9$

- Jednodimenz. Particionisanje (posmatramo X i Y svaki za sebe) daje sledeće klase ekv.

E1: $x < 3$	E2: $3 \leq x \leq 7$	E3: $x > 7$
E4: $y < 5$	E5: $5 \leq y \leq 9$	E6: $y > 9$

Multidimenzionalno particioniranje (primer)

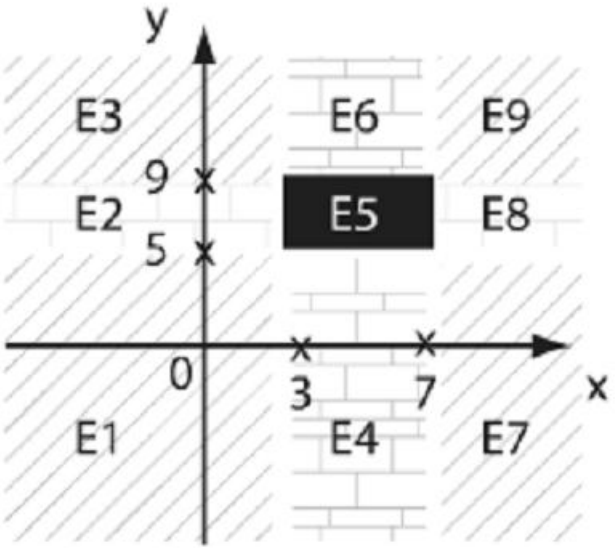
- Grafička predstava



Multidimenzionalno particioniranje (primer)

- Ako se uzmu kombinacije klasa ekvivalencija za X i Y:

E1: $x < 3, y < 5$ E2: $x < 3, 5 \leq y < 9$ E3: $x < 3, y > 9$
E4: $3 \leq x \leq 7, y < 5$ E5: $3 \leq x \leq 7, 5 \leq y \leq 9$ E6: $3 \leq x \leq 7, y > 9$
E7: $x > 7, y < 5$ E8: $x > 7, 5 \leq y \leq 9$ E9: $x > 7, y > 9$



Multidimenzionalno particioniranje

- Ako se primeni mult. p. program će biti detaljnije testiran
- Mana je što u slučaju N uslova dolazi do kombinatorne eksplozije. To se može rešavati tehnikama kombinatornog testiranja

Identifikacija test primera iz klasa

1. Dodeliti jedinstveni broj svakoj klasi.
2. Napisati test primere za sve legalne klase uključujući u jedan test što više klasa.
3. Napisati po jedan test primer za svaku nelegalnu klasu zasebno (da bi se izbeglo da jedan nelegalan ulaz maskira drugi zbog ugrađenih provera u programu).

Primer podele na klase ekvivalencije

- Dat je potprogram IME koji ispituje ispravnost imena datoteke. Sintaksa imena je data sa:

  <ime datoteke> ::= <ime> [.<tip>],

  <ime>: dužine 1-6, sadrži samo alfanumerike, počinje slovom

  <tip>: dužine 0-3, sadrži samo alfanumerike,
- Ukoliko se izostave tačka i tip, podrazumeva se .DAT.
- Potprogram postavlja kod ishoda sa značenjem: 0 - neispravno ime datoteke, 1 - ispravno ime datoteke, ima tačku ali nema tip; 2- ispravno ime datoteke, sa podrazumevanim tipom .DAT, 3 - ispravno ime sa zadatim tipom.

Primer (nastavak)

Legalne klase ekvivalencije	**Nelegalne klase ekvivalen.**
Ime:	Ime:
1. Ima 1-6 znakova	1. Ima 0 znakova
4. Sadrži samo alfanumerike	3. Ima više od 6 znakova
7. Počinje slovom	5. Ime sadrži spec. znakove
	6. Ime sadrži blankove, tabove
	8. Ime počinje cifrom
	9. Ime počinje spec. znakom
Tačka i tip:	Tačka i tip:
10. Naveden je iza imena	11. Postoji ime i tip a nema tačke
12. Postoji tačka a nema tipa	15. Tip ima više od tri znaka
13. Ne postoji tačka ni tip	17. Tip sadrži i druge znakove
14. Tip ima od 0-3 znaka	
16. Tip je alfanumerički	

Primer (nastavak)

- Testovi za legalne klase:
  - Proba.txt pokriva 1,4,7,10,14,16
  - Proba. pokriva još 12
  - Proba pokriva još 13
- Za svaku nelegalnu klasu treba poseban test ....

1. .bla
3. Dugackoime.log
5. %#,,$^$#.grr
6. Ime i prezime.dat
8. 14343
9. ..doc
11. ime tip
15. tra.lalala
17. dokument.$$$

Analiza graničnih vrednosti

- Programeri često previđaju:
  - Posebnu obradu koja je potrebna na granicama klasa ekvivalencije
- Na primer, programer može nepravilno napisati < umesto <=
- Analiza graničnih vrednosti:
  - Izabrati test primere na granicama različitih klasa ekvivalencije (jedno ili multidimenzionalnih), ili na osnovu specificiranih veza među ulazima
  - Po jedan test primer za svaku graničnu vrednost

Majersove preporuke za granične vrednosti

1. Ako ulazni uslov precizira opseg vrednosti, napisati testove za same krajeve opsega i testove ih ulaza za situacije odmah iza legalnih krajeva. Npr. Ako je ulaz u opsegu -1.0 do 1.0 testirati treba za -1.0, 1.0, -1.0001, 1.0001 (ako se pretpostavi da je najmanja delta između dva broja koja pravi neku razliku u programu .0001).

2. Ako ulazni uslov precizira broj vrednosti, napisati testove za minimalni i maksimalni broj vrednosti i jedan ispod i iznad ovih vrednosti. Na primer, ako ulazni fajl može da sadrži 1-255 zapisa, napisati testove za 0, 1, 255, i 256 zapisa.

3. Primeniti preporuku 1. na izlazne uslove.

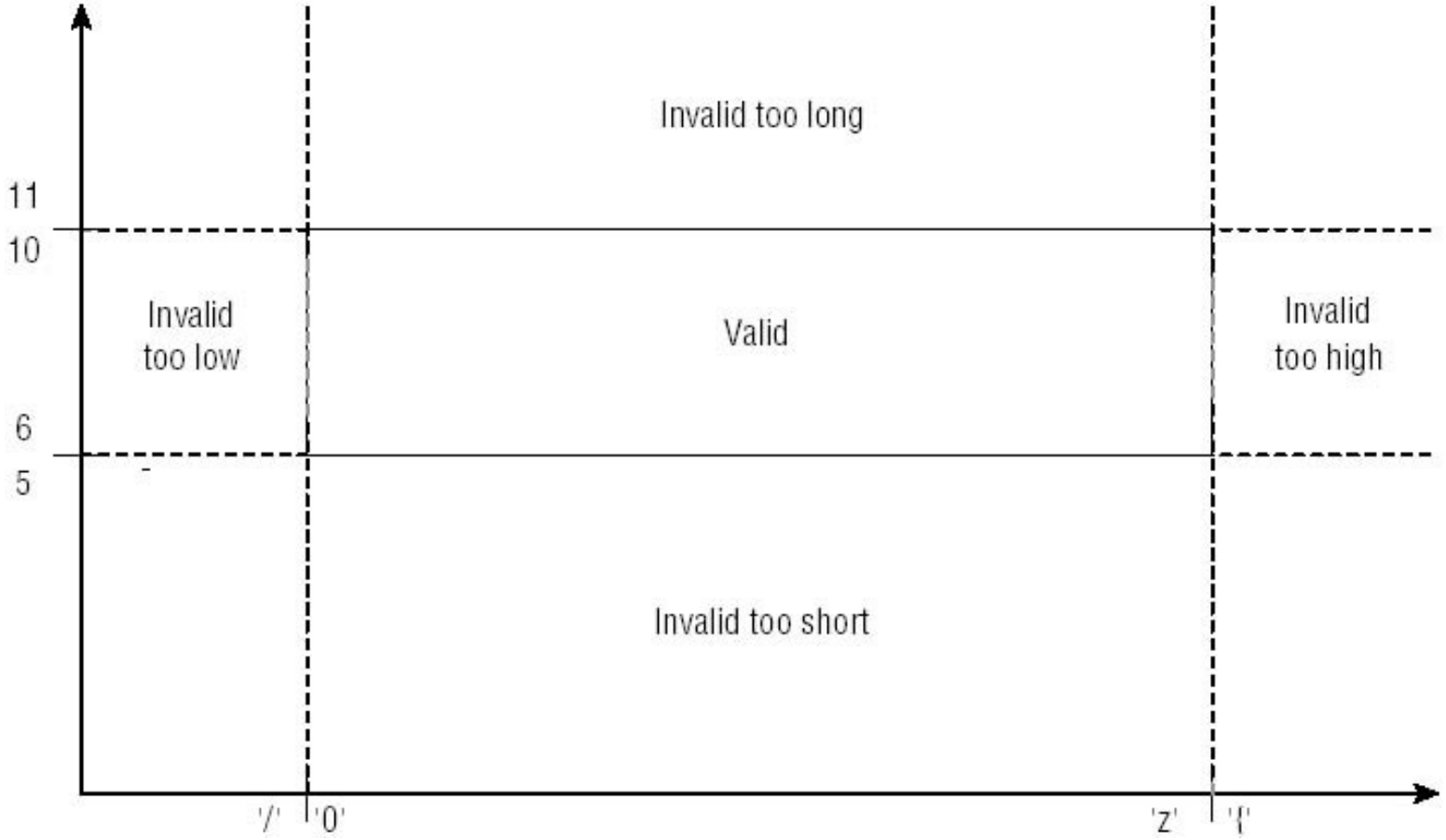
4. Primeniti preporuku 2. na izlazne uslove. Npr. ako na stranu izveštaja staje 65 redova, napisati testove za 64, 65 i 66 redova.

5. Ako je ulaz (ili izlaz) programa uređen skup (sekvencijalni fajl, na primer, ili linearna lista ili tabela), fokusirati pažnju na prvi i poslednji element skupa.

6. Upotrebiti sopstvenu pamet za nalaženje drugih graničnih uslova

Dalji primer – BVA za stringove

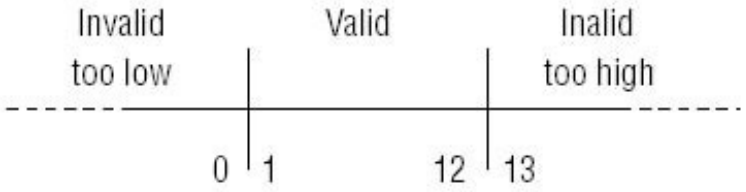
- npr. ako se traži lozinka, od 6 do 10 znakova dugačka

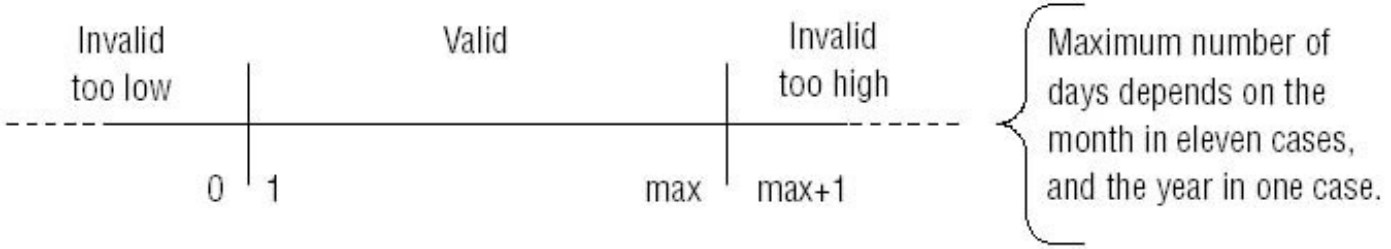


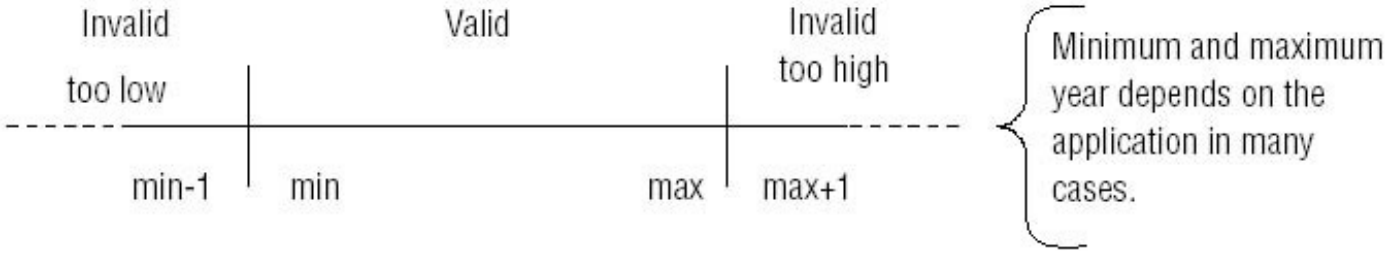
Datumi

- treba voditi računa i o formatu datuma, internoj reprezentaciji (Y2K problem, Unix brojač ide od Jan 1, 1970.

MM/DD/YY

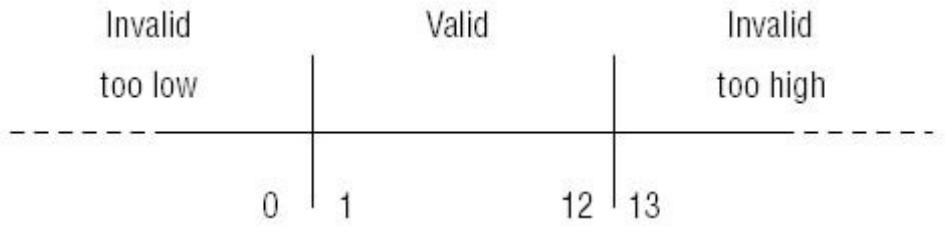


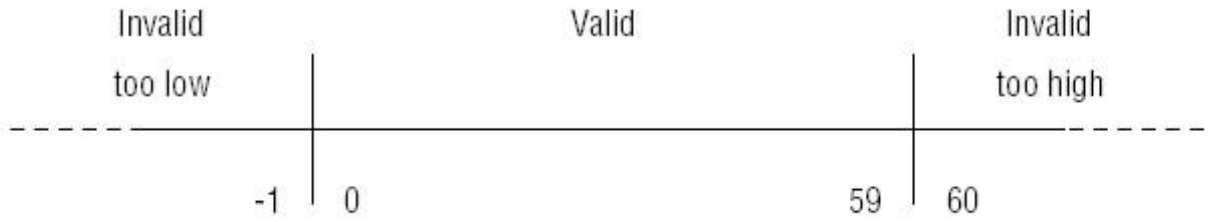


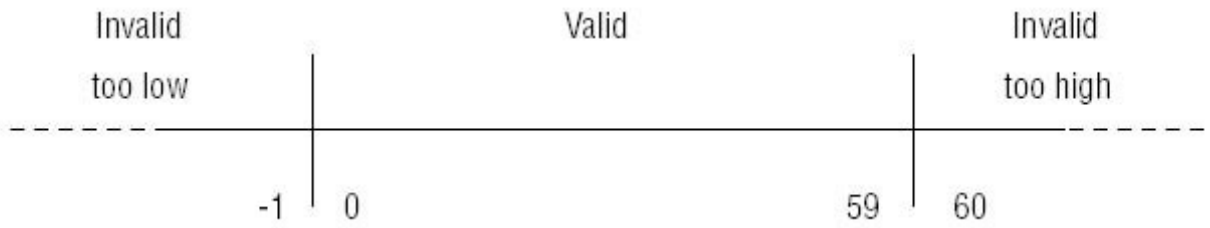


HH:MM:SS

Vreme







- Treba voditi računa i o formatima, vremenskim zonama, pomeranju vremena...

Novčane sume

- Često se reprezentuju kao decimalni brojevi sa fiksnim zarezom, znači imaju dobro definisan epsilon.
- Treba voditi računa i o valuti, formatu pisanja.

Primer analize graničnih vrednosti

Treba testirati metod *textSearch* koji vrši pretragu nepraznog stringa *s* u tekstu *txt*. Pozicije znakova u *txt* počinju od 0. Ulazni paramteri u *textSearch* su *s* i *txt*. Metod vraća ceo broj *x*. Ako je $x \geq 0$ onda *x* označava startnu poziciju stringa *s* unutar *txt*. Negativna vrednost *x* označava da *s* nije nađen unutar *txt*.

Primer...

- Identifikujemo klase ekvivalencije na osnovu ulaza
- Min. Dužina stringova je 0 i to je donja granica (gornja nije specifikacijom ograničena)

Equivalence classes for *s*: E1: empty string, E2: nonempty string.
Equivalence classes for *txt*: E3: empty string, E4: nonempty string.

Primer...

- Na osnovu izlaza programa dobijamo još dve klase ekvivalencije:
- Graničnu vrednost x=0 dobijamo kada se s nalazi u txtu i to na samom početku

$$\text{E5}: x < 0, \ \text{E6}: x \geq 0.$$

Primer...

- Sada možemo osmisliti testove za klase ekv. i do sada uočene granične vrednosti:

$T = \{t_1 : (s = \epsilon,$
$txt = \text{“Laughter is good for the heart.”}),$
$t_2 : (s = \text{“Laughter”},\ txt = \epsilon),$
$t_3 : (s = \text{“good for”},$
$txt = \text{“Laughter is good for the heart.”}),$
$t_4 : (s = \text{“Laughter”},$
$txt = \text{“Laughter is good for the heart.”})\}$

- t1 i t2 pokrivaju klase ekv. E1 do E5, t3 pokriva E6.
- t4 eksplicitno pokriva granični uslov x=0 koji smo idenfitikovali

Primer...

- Možemo kao gran. Uslov. Dodati test kada se traženi string nalazi na samom kraju:

t_5 : (*s* = "heart.",
txt = "Laughter is good for the heart.")

Sada smišljamo testove blizu granica:

- *s* starts at position 1 in *txt*. Expected output: $p = 1$.
- *s* ends at one character before the last character in *txt*. Expected output: $p = k$, where k is the position from where *s* starts in *txt*.
- All but the first character of *s* occur in *txt* starting at position 0. Expected output: $p = -1$.
- All but the last character of *s* occur in *txt* at the end. Expected output: $p = -1$.

Primer...

$t_6 : (s =$ "Laughter",
$txt =$ "Laughter is good for the heart.",)

$t_7 : (s =$ "heart",
$txt =$ "Laughter is good for the heart.",)

$t_8 : (s =$ "gLaughter",
$txt =$ "Laughter is good for the heart."),

$t_9 : (s =$ " heard.",
$txt =$ "Laughter is good for the heart.".)

Primer...

- Možemo još probati da kombinujemo klase za s i txt:

$$E_1 \times E_3,\ E_1 \times E4,\ E_2 \times E3,\ E_2 \times E4.$$

- Testovi t1, t2 i t3 pokrivaju sve klase izuzev E1 x E3. Dodatni test pokriva i ovo:

$$t_{10}: (s = \epsilon,\ txt = \epsilon)$$

Uzročno-posledični grafovi

- Alternativni naziv tehnike je modelovanje zavisnosti
- Fokusira se na modelovanje veza između ulaznih uslova programa, tzv. *uzroka* i izlaznih uslova, tzv. *posledica.*
- Zavisnost se predstavlja vizuelno u vidu uzročno-posledičnog grafa. U suštini je reč o logičkoj relaciji koja može da se predstavi bulovim izrazom.
- Graf omogućava izbor različitih kombinacija ulaznih vrednosti za testiranje. Kombinatorna eksplozija u broju testova izbegava se upotrebom heuristika tokom generisanja testova.

Osnovni koraci u primeni ove metode

1. Specifikacija programa se podeli u radne delove koji se nezavisno testiraju. To je neophodno zbog prevelike kompleksnosti grafova kojima bismo pokušali da predstavimo velike specifikacije. Pojam radnog dela nije precizno definisan, ali to je, na primer, pri testiranju skript interpretera deo specifikacije koji se odnosi na jednu njegovu naredbu; ili, pri testiranju transakcionog sistema deo specifikacije koji se odnosi na pojedinačnu transakciju itd.
2. Određuju se uzroci i posledice u specifikaciji. Uzroci su u suštini klase ekvivalencije ulaznih uslova. Posledice su efekti na ponašanje programa koje ti ulazni uslovi imaju, tj. akcija programa koju oni izazivaju. Na primeru transakcionog sistema to bi bilo izmereno stanje u bazi podataka nakon transakcije ažuriranja ili odgovarajuća poruka o grešci, ukoliko je korisnik pokušao da unese neodgovarajuće podatke. Svim uzrocima i posledicama se dodeljuju jedinstveni identifikacioni brojevi.

Osnovni koraci u primeni ove metode

3. Na osnovu semantičke analize specifikacije kreira se uzročno-posledični graf. Uzroci i posledice predstavljaju početne i završne čvorove grafa, a veze se ostvaruju preko logičkih operatora koji čine grane grafa. Po potrebi se ubacuju i međučvorovi koji predstavljaju određene logičke kombinacije ulaznih uslova.
4. Ovako kreiran graf se dopunjuje ograničenjima. Ograničenja opisuju one kombinacije među uzrocima i među posledicama, koje je nemoguće ostvariti.
5. Graf se jednim metodološkim postupkom prevodi u tabelu odlučivanja. Etaloni test primera su kolone ove tabele.
6. Test primeri se generišu iz kolona tabele odlučivanja.

Formiranje grafa

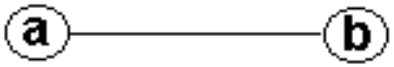
- Svaki čvor grafa predstavlja jedan ulazni uslov, kombinaciju ulaznih uslova ili posledicu i može biti u stanju 0 ili 1. Pri tome 1 označava da je uslov koji čvor predstavlja ispunjen, a 0 da nije ispunjen.
- Grane grafa označavaju logičke funkcije koji važe među čvorovima grafa, a te funkcije mogu biti:

Grane grafa

- *identity* - ako čvor a ima vrednost 1 i čvor b ima vrednost 1, a u suprotnom ima vrednost 0
- *not* - ako čvor a ima vrednost 1, čvor b ima vrednost 0, a u suprotnom ima vrednost 1
- *or* - ako bar neki od čvorova a, b ili c ima vrednost 1 i čvor d ima vrednost 1, a u suprotnom ima vrednost 0. Broj ulaza nije ograničen.
- *and* - ako i čvor a i čvor b imaju vrednost 1, onda i čvor c ima vrednost 1, a u suprotnom ima vrednost 0. Broj ulaza nije ograničen.

Grane grafa grafička predstava

IDENTITY

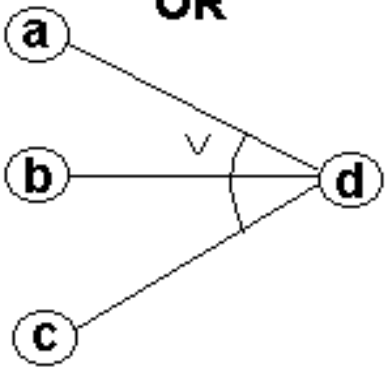


if a then b

NOT

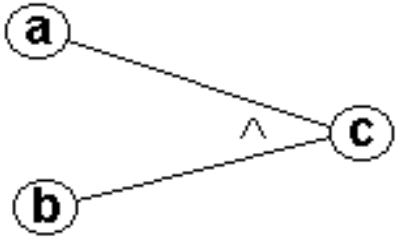
if (not a) then b

OR



if (a or b or c) then d

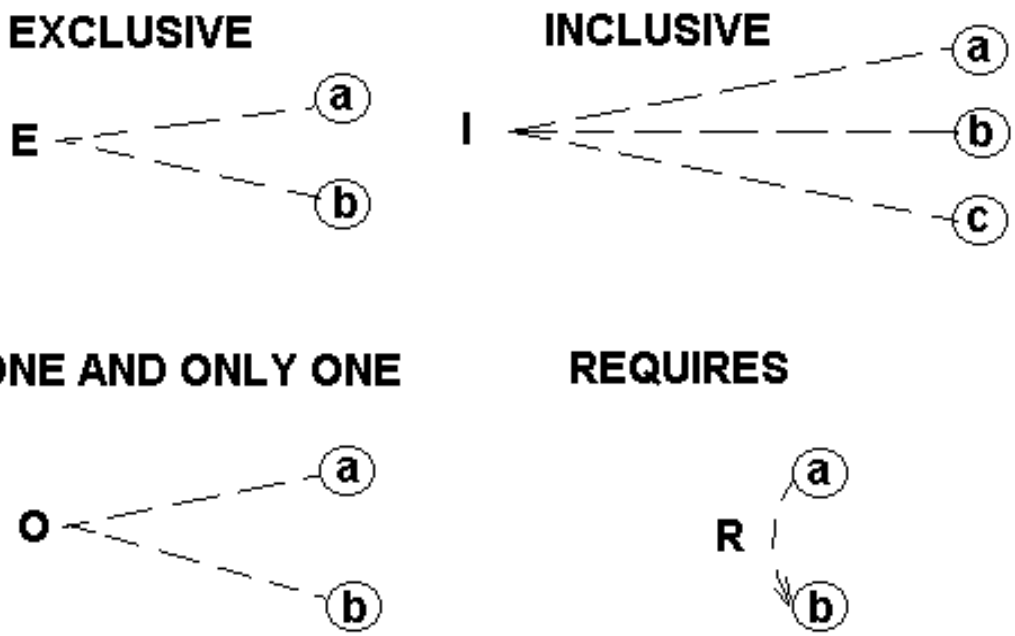
AND



if (a and b) then c

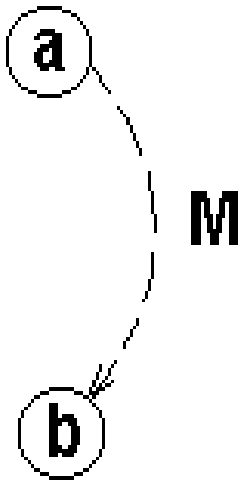
Ograničenja među uzrocima

- E - najviše jedan od čvorova a i b može imati vrednost 1
- I - bar jedan od čvorova a, b ili c mora imati vrednost 1
- O - jedan, i samo jedan, od čvorova a i b mora imati vrednost 1
- R - da bi čvor a imao vrednost 1, i čvor b mora imati vrednost 1



Ograničenje među posledicama

- M - ako je završni čvor a na vrednosti 1, završni čvor b mora imati vrednost 0



Primer

Ef_1 : Generate "Shipping invoice".

Ef_2 : Generate an "Order not shipped" regret letter.

Effect Ef_1 occurs when an order can be met from the inventory. Effect Ef_2 occurs when the order placed cannot be met from the inventory or when the ordered item has been discontinued after the order was placed. However, Ef_2 is masked by Ef_1 for the same order, that is both effects cannot occur for the same order.

Primer

- Iz britanskog standarda o testiranju softverskih komponenata http://www.testingstandards.co.uk/

Deo specifikacije zahteva

- Razmotrimo funkciju koja obradjuje podizanje novca. Ulazi funkcije su količina novca koja se podiže, tip računa i trenutno stanje računa, dok su izlazi novo stanje računa i kod akcije.
- Tip računa može biti poštanski (‘p’) ili klasični (‘c’).
- Kod akcije može biti ‘D&L’, ‘D’, ‘S&L’ ili ‘L’, tj. 'obradi podizanje novca i pošalji pismo’, ‘samo obradi podizanje novca’, 'blokiraj račun i pošalji pismo’ i 'samo pošalji pismo’, respektivno.

Nastavak...

Specifikacija funkcije data je u nastavku:

- Ukoliko ima dovoljno novčanih sredstava na računu ili bi novo stanje bilo u granicama dozvoljenog minusa, podizanje novca se obradjuje.
- Ukoliko bi podizanje novca dovelo do prekoračenja dozvoljenog minusa, podizanje novca nije moguće, dodatno ukoliko je u pitanju poštanski račun vrši se privremeno blokiranje istog.
- Pismo se šalje za sve obavljene transakcije u slučaju poštanskog računa, kao i za ne-poštanski račun ukoliko nema dovoljno novčanih sredstava (tj. račun više nije u plusu).

Uzroci i posledice:

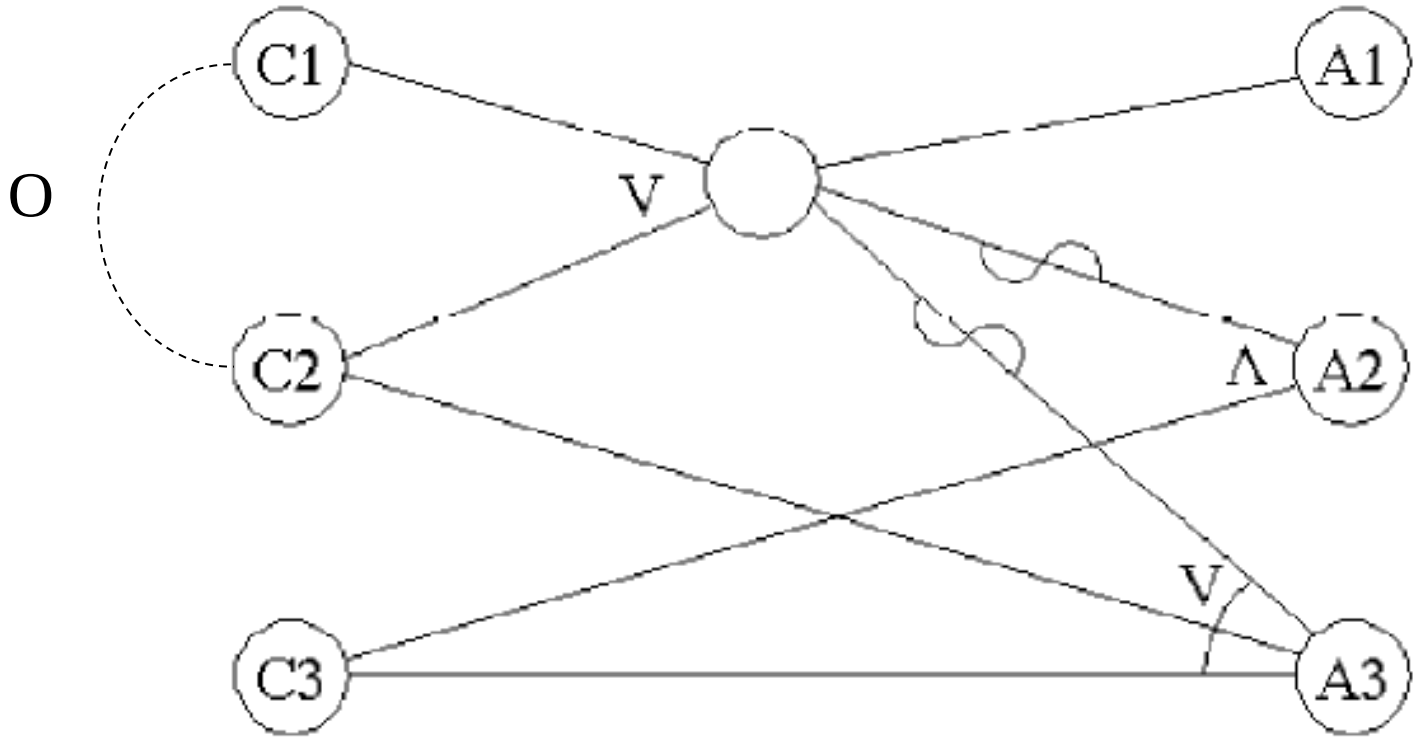
Uzroci:

- C1 Novo stanje je u plusu
- C2 Novo stanje je u dozvoljenom minusu
- C3 Račun je poštanski

Posledice:

- A1 Obrada podizanja novca
- A2 Privremeno blokiranje računa
- A3 Slanje pisma

Uzročno posledični Graf



C1 Novo stanje je u plusu

C2 Novo stanje je u dozvoljenom minusu

C3 Račun je poštanski

A1 Podizanje novca

A2 Blokiranje računa

A3 Slanje pisma

Tabela odlučivanja

- Dvodimenzionalno mapiranje uzroka na posledice
- Može se izvesti iz uzročno posledičnog grafa

Primer tabele odlučivanja

Rules:	1	2	3	4	5	6	7	8
C1: New balance in credit	F	F	F	F	T	T	T	T
C2: New balance overdraft, but within authorised limit	F	F	T	T	F	F	T	T
C3: Account is postal	F	T	F	T	F	T	F	T
A1: Process debit	F	F	T	T	T	T	*	*
A2: Suspend account	F	T	F	F	F	F	*	*
A3: Send out letter	T	T	T	T	F	T	*	*

- ima po jedan red za svaki uzrok i posledicu
- Kolone odgovaraju test primerima.

  (T-mora biti ispunjeno, F-ne sme biti ispunjeno,
  *-kombinacija uzroka nemoguća, te akcije nisu definisane)

Primer tabele odlučivanja

- Za prethodni primer tabele odlučivanja, teorijski postoji $2^3=8$ test primera (kombinacija vrednosti uzroka).
  - Upotrebom Majersove tehnike redukcije kombinacija uzroka ovaj broj se smanjuje.

  (*Glenford Myers, The Art of Software Testing, Second Edition, John Wiley&Sons, 2004)*

Transformacija grafa u tabelu odlučivanja

1. Jedan po jedan završni čvor se postavlja na vrednost 1.
2. Ide se od tog završnog čvora unazad kroz graf i pronalaze *sve* kombinacije ulaznih čvorova koje ga setuju na 1, vodeći računa o ograničenjima.
3. Od svake kombinacije ulaznih čvorova, koja setuje odabrani završni čvor na 1, formira se jedna kolona u tabeli odlučivanja. U slučaju da vrednost nekog ulaznog čvora nije bitna ili da se podrazumeva na osnovu vrednosti ostalih ulaznih čvorova i važećih ograničenja, taj ulaz u koloni tabele odlučivanja se ostavlja praznim.
4. Za svaku od gornjih kombinacija se određuje i efekat koji ima na stanje ostalih završnih čvorova. Ovo je korisno za fazu testiranja programa pomoću generisanih test problema, jer se unapred identifikuju efekti koji se očekuju pri izvršavanju programa pomoću test problema.
5. Ukoliko se pojave kolone u tabeli odlučivanja koje se preklapaju, vrši se čihovo sažimanje. Preklapanje se javlja kada su im vrednosti po svim čvorovima jednake ili kad je na određenim pozicijama vrednost u jednoj koloni nebitna, a u drugoj fiksno određena (tada se uzima ona fiksna vrednost).

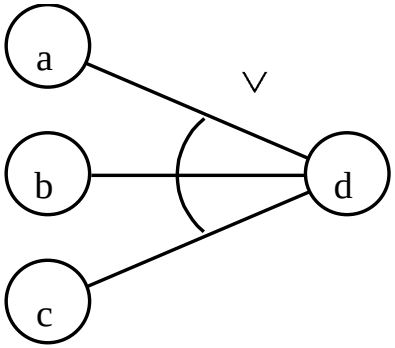
Transformacija grafa u tabelu odlučivanja

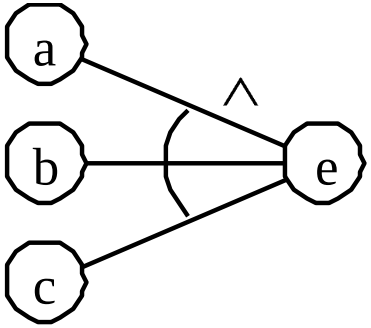
- Da bi se smanjio broj kombinacija koje se međusobno preklapaju, pri izvođenju koraka 2 se primenjuju sledeće "olakšice":

1. Kada se ide unazad kroz čvor sa operatorom OR, čiji izlaz treba da bude **1**, nikada se ne ispituje situacija u kojoj je više od jednog ulaza postavljeno na 1. Cilj ovoga je da se izbegne mogućnost da ne dođe do otkrivanja greške u slučaju da jedan ulazni uslov *maskira* drugi.
2. Kada se ide unazad kroz čvor sa operatorom AND, čiji izlaz treba da bude **0**, treba ispitati sve kombinacije ulaza koje dovode izlaz na 0. Međutim, za one kombinacije u kojima su neki od ulaza na vrednosti 1, nije potrebno ispitati sve kombinacije koje taj ulaz dovode na vrednost 1, već je dovoljna samo po jedna kombinacija.
3. Kada se ide unazad kroz čvor sa operatorom AND, čiji izlaz treba da bude **0**, za slučaj kada svi ulazi imaju vrednost 0, nije potrebno ispitati sve kombinacije koje dovode svaki od ulaza na vrednost 0, već je dovoljna samo po jedna kombinacija.

Majersove heuristike

- $\forall$ - označava da treba uzeti u obzir sve moguće kombinacije
- $\exists$ - označava da treba izabrati samo jednu jedinu kombinaciju





d / $\forall$ 0

a	b	c
$\forall$ 0	$\forall$ 0	$\forall$ 0

d / $\forall$ 1

a	b	c
$\forall$ 0	$\forall$ 0	$\forall$ 1
$\forall$ 0	$\forall$ 1	$\forall$ 0
$\forall$ 1	$\forall$ 0	$\forall$ 0

e / $\forall$1

a	b	c
$\forall$1	$\forall$1	$\forall$1

e / $\forall$0

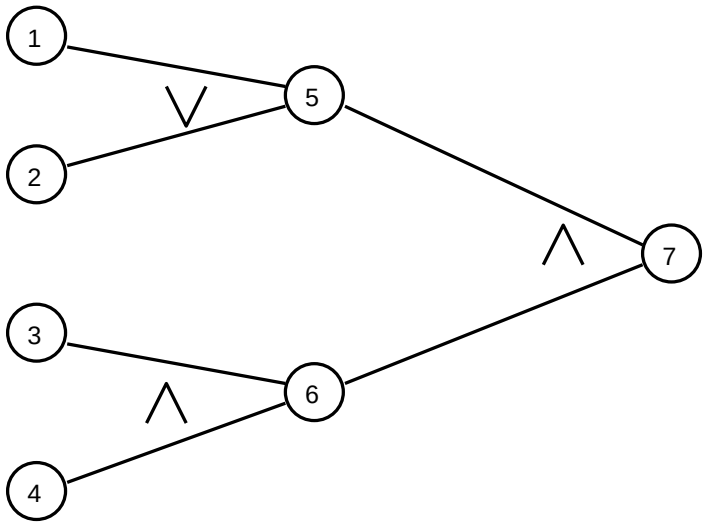
a	b	c
$\exists$0	$\exists$0	$\exists$0
$\forall$0	$\forall$0	$\exists$1
$\forall$0	$\exists$1	$\forall$0
$\forall$0	$\exists$1	$\exists$1
$\exists$1	$\forall$0	$\forall$0
$\exists$1	$\forall$0	$\exists$1
$\exists$1	$\exists$1	$\forall$0

Primer transformacije

- Pretpostavimo da želimo da odredimo sve kombinacije ulaznih uslova, koje dovode izlaz na stanje **nula**. Olakšica 3 kaže da bi trebalo da uzmemo samo jednu kombinaciju za slučaj kada su čvorovi 5 i 6 na nuli. Olakšica 2 kaže da bi u slučaju kada je čvor 5 na jedinici i čvor 6 na nuli, trebalo uzeti samo jednu kombinaciju koja dovodi čvor 5 na jedinicu, umesto da razmatramo sve moguće kombinacije koje dovode čvor 5 na jedinicu.
- Za slučaj kada je čvor 5 na nuli, a čvor 6 na jedinici, treba ispitati samo jednu kombinaciju koja dovodi čvor 6 na jedinicu. Olakšica 1 kaže da kada čvor 5 treba da ima vrednost jedan, ne treba istovremeno postavljati čvorove 1 i 2 na jedinicu. Tako dolazimo do pet kombinacija vrednosti čvorova 1 do 4, kao na primer:

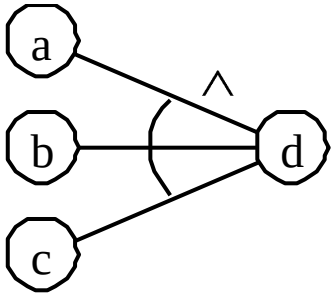
  0 0 0 0 (5=0, 6=0)
  1 0 0 0 (5=1, 6=0)
  1 0 0 1 (5=1, 6=0)
  1 0 1 0 (5=1, 6=0)
  0 0 1 1 (5=0, 6=1)

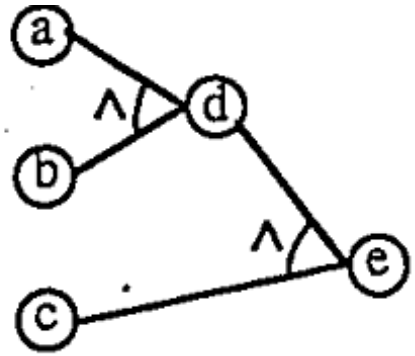
- umesto 13 mogućih kombinacija koje dovode čvor 7 na nulu.



Kritike majersovog pristupa

1. Nekonzistencija u definiciji pravila





(a) An Example CEG

$\frac{d}{\forall 0}$

a	b	c
∃0	∃0	∃0
?0	?0	∃1
?0	∃1	?0
?0	∃1	∃1
∃1	?0	?0
∃1	?0	∃1
∃1	∃1	?0

∀

		1	2	3	4	5
Cause						
	a	0	1	0	0	1
	b	0	1	0	1	0
	c	0	0	1	1	1
Effect						
	e	0	0	0	0	0

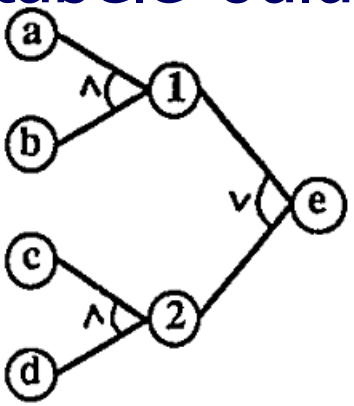
∃

		1	2	3
Cause				
	a	0	1	0
	b	0	1	0
	c	0	0	1
Effect				
	e	0	0	0

? Na jednom mestu u knjizi ∀, na drugom ∃

Kritike majersovog pristupa

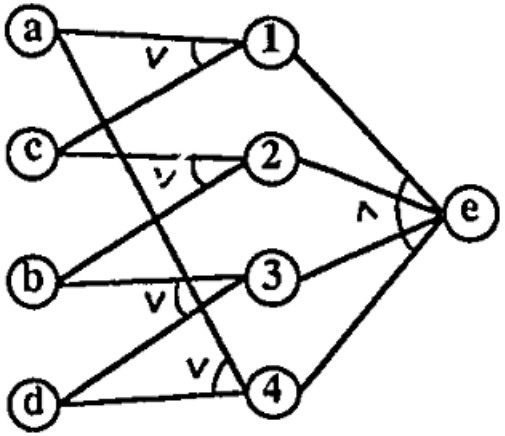
2. Različiti grafovi sa istim značenjem daju različite tabele odlučivanja



(a) CEG of e = (a and b) or (c and d)

	1	2	3	4	5	6
Causes						
a	1	1	1	0	1	0
b	1	1	1	0	0	1
c	0	1	0	1	1	1
d	0	0	1	1	1	1
Effects						
e	1	1	1	1	1	1

(c) Decision Table from CEG in (a)



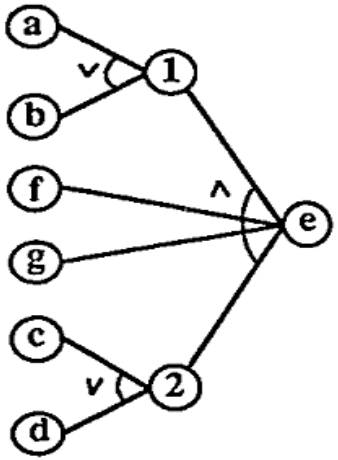
(b) CEG of e = (a or c) and (a or d) and (b or c) and (b or d)

	1	2
Causes		
a	1	0
b	1	0
c	0	1
d	0	1
Effects		
e	1	1

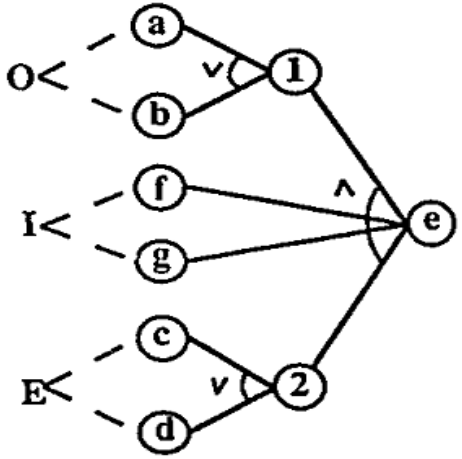
(d) Decision Table from CEG in (b)

Kritike majersovog pristupa

3. Nije lako uočiti grešku izostavljanja ograničenja uzroka



(a) Example of a CEG with missing constraints



(b) Added constraints to CEG in (a)

	1	2	3	4
Causes				
a	1	1	0	0
b	0	0	1	1
c	1	0	1	0
d	0	1	0	1
f	1	1	1	1
g	1	1	1	1
Effects				
e	1	1	1	1

(c) Same decision table for both CEGs in (a) and (b)

Kritike majersovog pristupa

4. Traže se samo kombinacije uzroka koje dovode posmatranu posledicu na 1.

- Na ovaj način, nije lako uočiti da li je posledica uvek na 1 ili uvek na 0 (nezavisno od uzroka), što bi potencijalno ukazalo na grešku u specifikaciji.

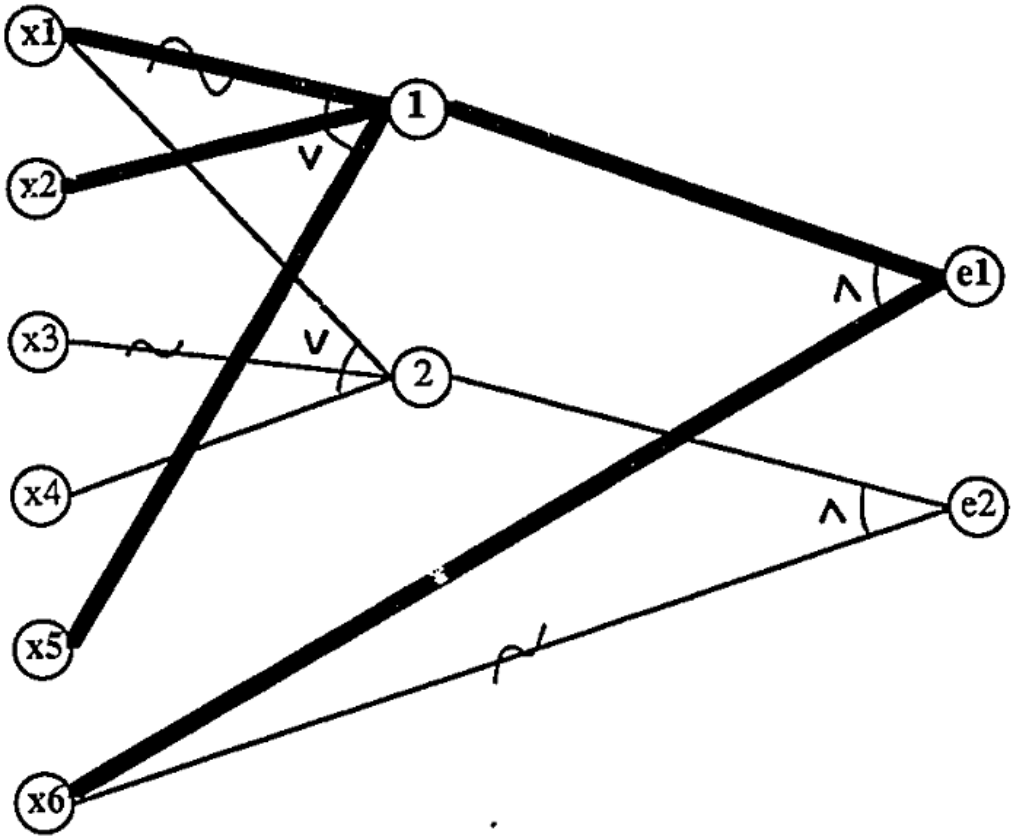
Tehnika senzitizacije putanja [nursimulu]

- Alternativna tehnika za izvođenje tabele odlučivanja iz uzročno-posledičnog grafa
- Def. 1: **Skup uzroka** posledice E je skup uzroka koji imaju uticaja na E.
- Def. 2: **Senzitizacija** posledice E na određeni uzrok C znači da se svi drugi ulazi dovode na fiksne vrednosti tako da vrednost E direktno zavisi samo od C (direktno ili invertovano).
- Def. 3: **Senzitizovana putanja** je putanja od uzroka C do posledice E na kojoj su svi međučvorovi senzitizovani na uzrok C.

Tehnika senzitizacije putanja-primer

CauseSet(e1)=
{x1,x2,x5,x6}

CauseSet(e2)=
{x1,x3,x4,x6}



- Želimo da senzitivizujemo e1 na x2:
  - Moramo prvo da senz. 1 na x1, znači x1=1 i x5=0, zbog ILI čvora
  - Sada smo senz. e1 na x2 sveli na senz. e1 na 1, tako da još treba x6=1, jer je e1 čvor I
  - Dakle postigli smo e1=x2

Tehnika senzitizacije putanja

For each effect, ej in a cause-effect graph CEG do

For each cause cj in the Cause Set C of effect ej do

->Sensitize effect ej to cause cj

->For each possible combinations create two cause combinations one with cj set to 1 and another one with cj set to 0

->Create a column in the decision table for each cause combination

->Use the values set to the causes in C and the constraints present in the CEG to determine the values of the other causes in the cause effect graph

->Use the values set to the causes in C in order to determine the values of the other effects in CEG for these two combinations

->Remove any redundant column

End for

->If possible, create two additional cause combinations where in one most of the causes in C are absent and in the other most of them are present.

->Create a column in the decision table for each cause combination

->Use the values set to the causes in C and the constraints present in the CEG to determine the values of the other causes in the CEG

->Use the values set to the causes in C in order to determine the values of he other effects in CEG for these two combinations

->Remove any redundant column

End for

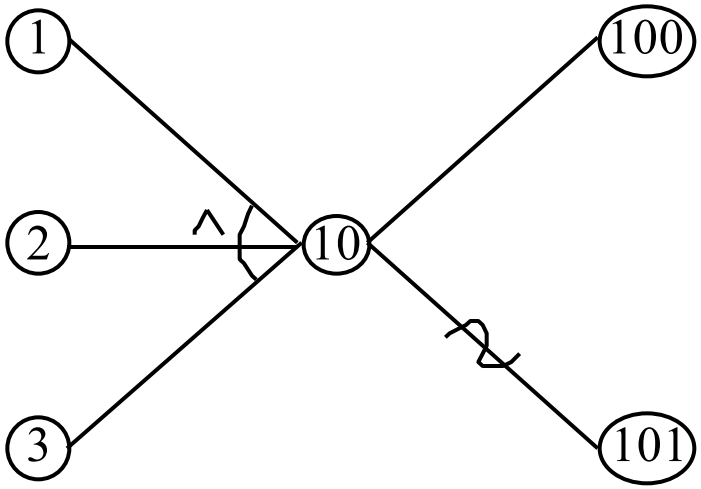
->Derive test scenarios from the decision table obtained

Primer – sendfile komanda

- *U datoj mreži, Sendfile komanda se koristi da pošalje datoteku korisniku na nekom drugom serveru datoteka. Sendfile komanda uzima tri argumenta: prvi argument treba da bude a datoteka u home direktorijumu pošiljaoca, drugi argument treba da bude ime datoteke servera primaoca, a i argument treba da bude primaočev userid. Ako su svi argumenti tačni, datoteka se uspešno šalje, u suprotnom pošiljalac dobija poruku o grešci.*

Primer – sendfile komanda

Uzroci	Posledice
. Prvi argument je ime eg fajla u home direktorijumu pošiljaoca. 2. Drugi argument je ime servera primaoca. . i argument je primaočev userid.	100. Fajl je uspešno poslat. 101. Pošiljalac dobija poruku o grešci.



Senz. putanja

	1	2	3	4	5
Causes					
1	**1**	**0**	1	1	0
2	**1**	1	**0**	1	0
3	**1**	1	1	**0**	0
Effects					
100	**1**	**0**	**0**	**0**	0
101	**0**	**1**	**1**	**1**	1

Majers

	1	2	3	4	5	6	7	8
Causes								
1	1	0	1	0	0	1	1	0
2	1	0	0	1	0	1	0	1
3	1	0	0	0	1	0	1	1
Effects								
100	1	0	0	0	0	0	0	0
101	0	1	1	1	1	1	1	1

Uzročno-posledični grafovi

- Nisu praktično primenljivi na sisteme koji:
  - Poseduju vremenska ograničenja
  - Poseduju konkurentne procese