Testiranje zasnovano na modelu stanja

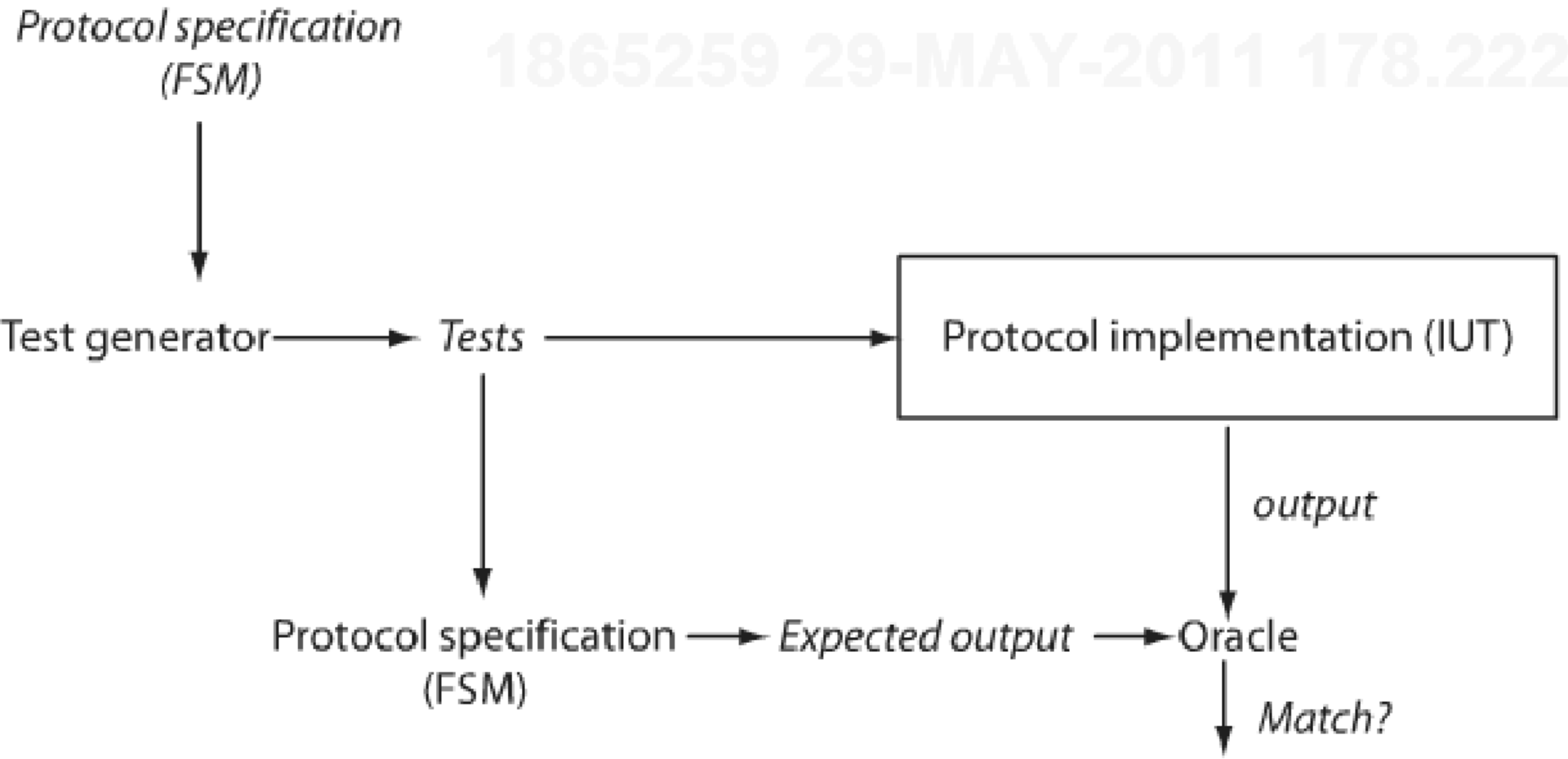
Testiranje zasnovano na modelu stanja

- Tehnika crne kutije na osnovu specifikacije ponašanja softverske komponente u vidu konačnog automata (Finite state machine, FSM)
- Prirodno, tehnika je efikasna samo za onaj softver čije se ponašanje može predstaviti konačnim automatom
- Upoznaćemo se sa tehnikom razmatrajući konkretan primer

Testiranje usaglašenosti (conformance testing)

- Ponašanje (na primer, komunikacioni protokol) se modeluje konačnim automatom (FSM)
- Model se implementira nekim softverom (implementation under test, IUT) i želimo testirati da li implementacija odgovara modelu

Testiranje usaglašenosti (conformance testing)

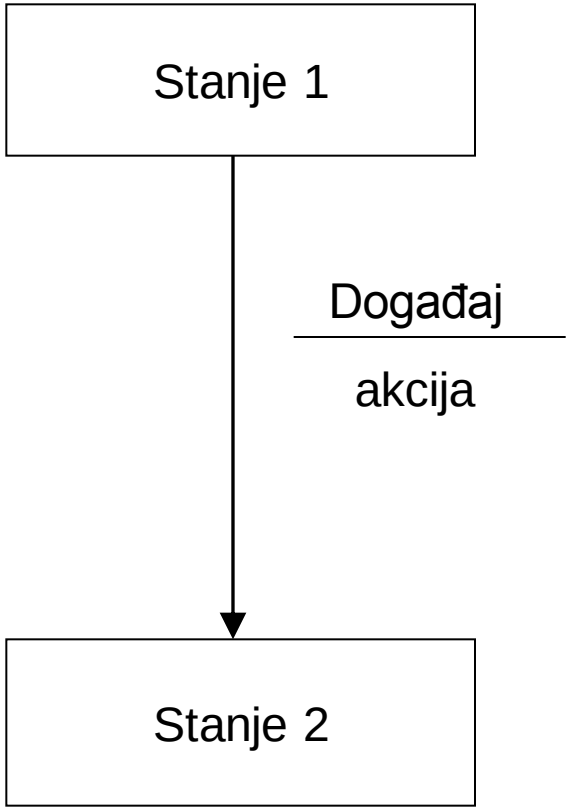


Model stanja

- Model stanja komponente određuje njena stanja, prelaze među stanjima i sa njima povezane događaje i akcije.
- Događaji su uvek izazvani nekim ulazom, dok akcije proizvode izlaz.
- Model stanja predstavlja se dijagramom promene stanja

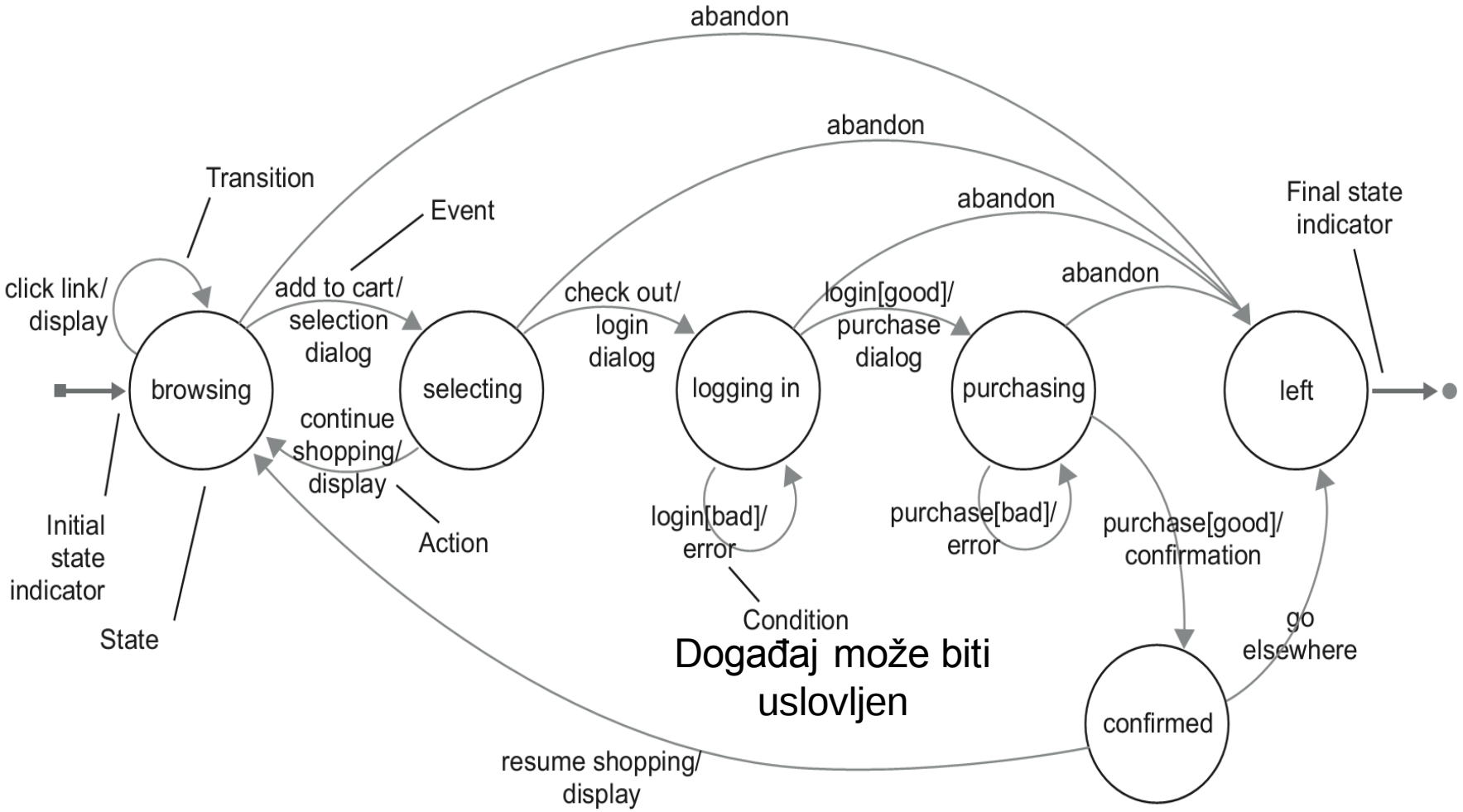
Dijagram promene stanja

- Prelaz među stanjima definisan je tekućim stanjem i ulaznim događajem i označava se parom događaj/akcija
- Stanje traje neodređen vremenski period, sve dok se nešto ne desi, spoljašnje u odnosu na sistem.
- Događaj se desi trenutno ili u ograničenom vremenskom periodu. To je nešto što se desilo, spoljašnja pojava, koja pokreće tranzicije. Događaji se mogu aktivirati na različite načine, kao što su, na primer, od strane korisnika sa tastature ili miša, spoljnog uređaja, ili čak operativnog sistema.
- Akcija je odgovor sistem tokom tranzicije između stanja. Akcija, kao i događaj, je ili trenutna ili zahteva ograničeno vreme. Često akcija može da se posmatra kao sporedni efekat događaja.

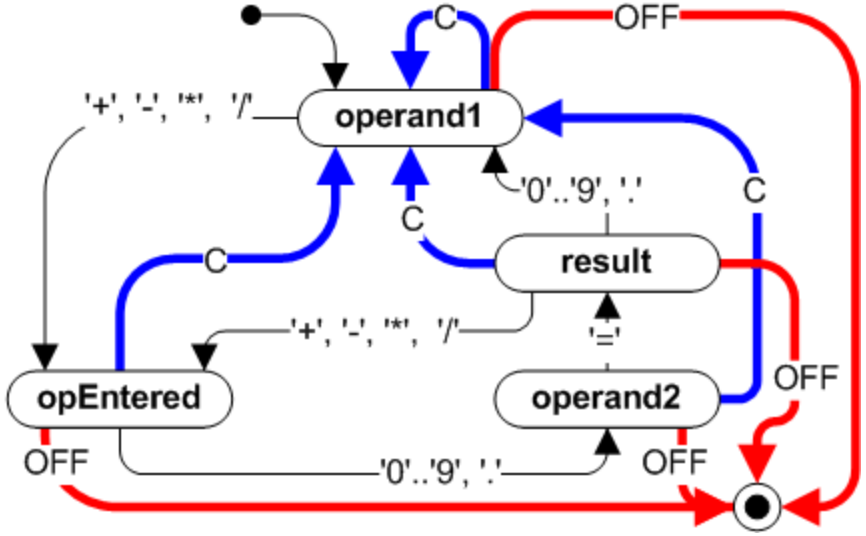


Primer: elektronska kupovina

- Iz perspektive klijenta



Primer: kalkulator

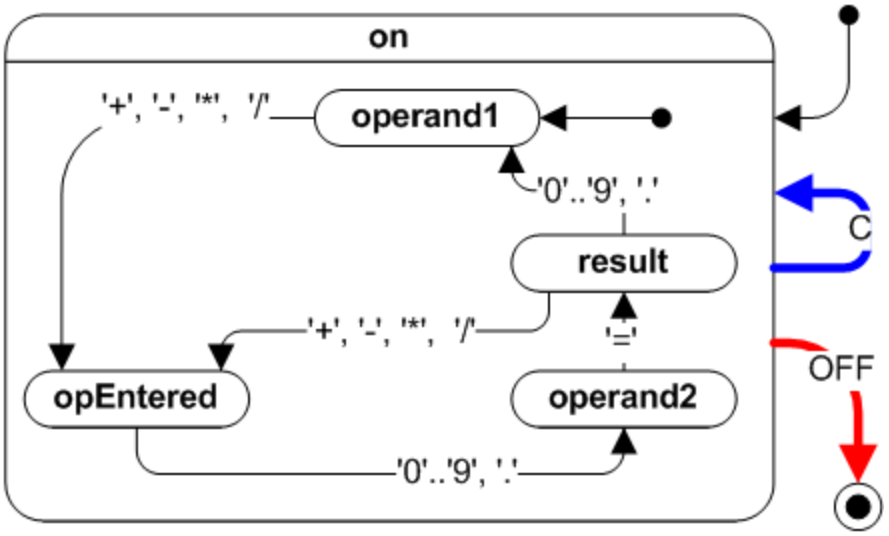


Hijerarhijski automat

- Hijerarhijski automat (hierarchical state machine, HSM) može da ima nadstanja i u njima ugneždena podstanja
- Nadstanje automatski obrađuje događaje koje podstanja ne prihvataju (definiše zajedničko ponašanje za sva svoja podstanja).
- Sledeći HSM ima isto značenje kao prethodni FSM za kalkulator.

Primer: kalkulator

- **On** predstavlja nadstanje, a ostalo su podstanja.
- Kada treba da se testira ovakav automat, moguće je izvršiti ekspanziju nadstanja u običan automat, pa na nivou običnog definisati testove.



Kriterijumi testiranja modela stanja M

- **Pokrivanje stanja** (state cover): zahteva skup testova čije izvršavanje uzrokuje posećivanje svakog stanja u M bar jednom
- **Pokrivanje prelaza** (transition cover ili 0-switch cover): zahteva skup testova koji uzrokuju izvršavanje svakog prelaza u M bar jednom
- **Pokrivanje izmena** (switch cover ili 1-switch cover): zahteva skup testova koji uzrokuju izvršavanje svakog mogućeg para prelaza (sekvence od dva uzastopna prelaza) u M bar jednom.
- **Pokrivanje sa N izmena**: zahteva skup testova koji uzrokuju izvršavanje svake moguće sekvence od N+1 prelaza u M bar jednom.

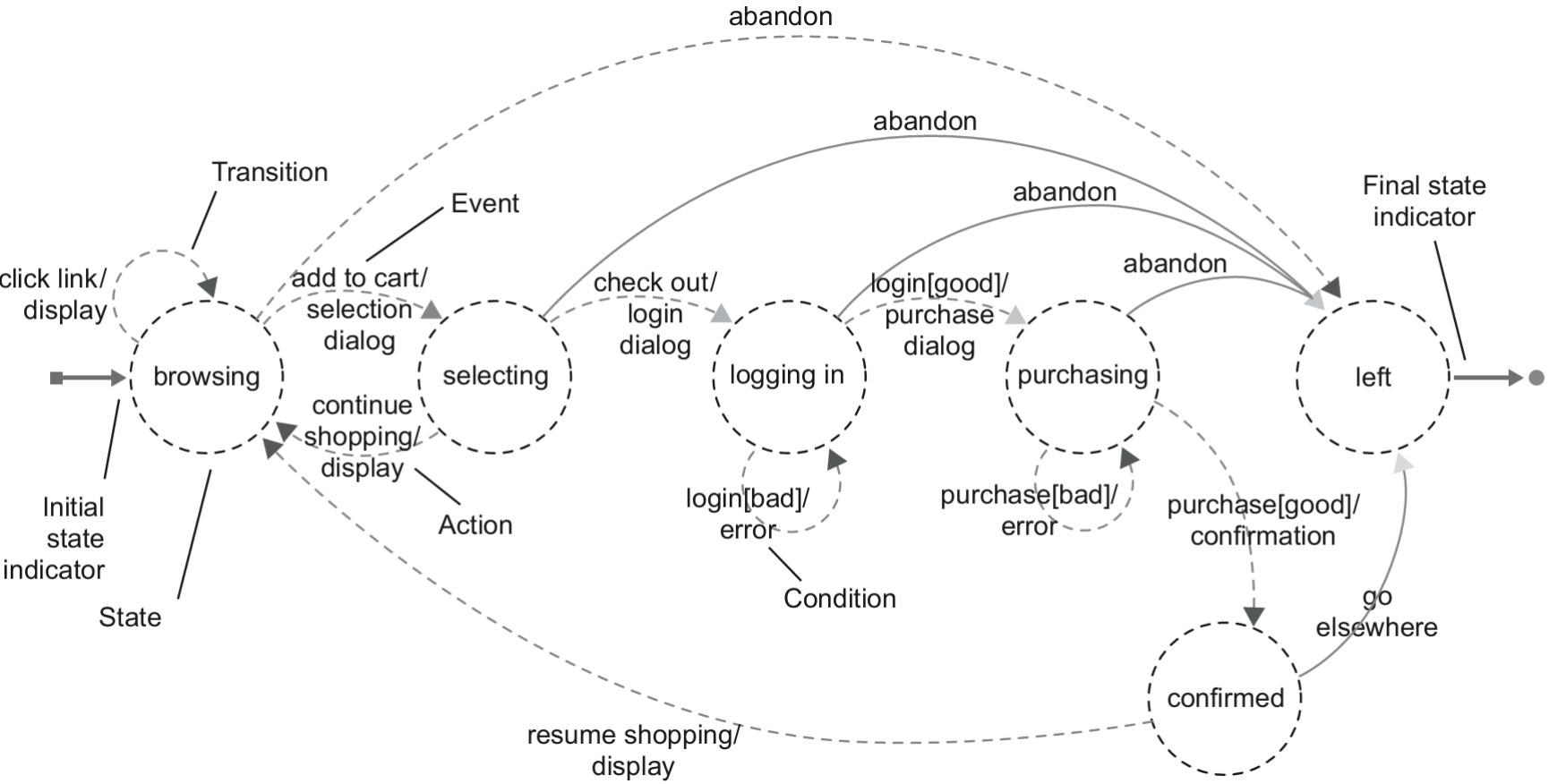
Pokrivanje stanja i prelaza

1. Usvojiti pravilo gde test mora početi i gde mora (ili može) da se završi. Na primer, test mora početi u inicijalnom stanju i može završiti samo u konačnom stanju.
2. Od dozvoljenog početnog stanja definisati sekvencu kombinacija događaj/stanje koja dovodi do dozvoljenog završanog stanja. Za svaku tranzicije koja će se desiti, zabeležiti očekivanu akciju koju bi trebalo da preuzme sistem. To je očekivani rezultat testa.
3. Označiti svako stanje i prelaz iz definisanog testa kao pokrivene (npr. na test dijagramu).
4. Ponoviti korake 2 i 3 sve dok se ne pokriju sva stanja i svi prelazi.

- Ova procedura će generisati logičke slučajeve testiranja. Da bi se napravili konkretni slučajevi testiranja, moraju se usvojiti stvarne vrednosti ulaznih i izlaznih podataka.

Primer pokrivanja stanja/prelaza

TP1. (browsing, click link, display, add to cart, selection dialog, continue shopping, display, add to cart, selection dialog, checkout, login dialog, login[bad], error, login[good], purchase dialog, purchase[bad], error, purchase[good], confirmation, resume shopping, display, abandon, left).



Primer pokrivanja stanja/prelaza

Preostali TP (iskustvena procena za ukupan broj je koliko ima ulaznih grana u finalno stanje):

2. (browsing, add to cart, selection dialog, abandon, <no action>, left)

3. (browsing, add to cart, selection dialog, checkout, login dialog, abandon, <no action>, left)

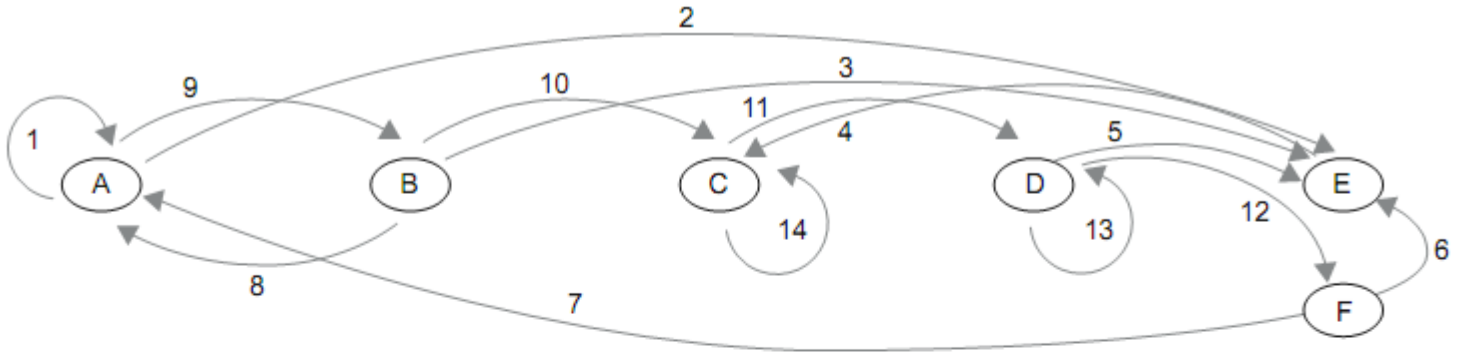
4. (browsing, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)

5. (browsing, add to cart, selection dialog, continue shopping, display, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, purchase[good], confirmation, go elsewhere, <no action>, left)

Pokrivanje izmena

- Potrebno je prvo tabelarno predstaviti sve 0-izmene, zatim 1-izmene itd. po potrebi
- Potom razviti skup testova koji pokrivaju prvo sve nula izmene, potom ih dopuniti da pokriju sve n-izmene

Primer (el. kupovina)



0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9				A9B10	A9B8	A9B3
B10	B8	B3	B10C14	B10C11	B10C4	B8A1	B8A2	B8A9			
C14	C11	C4	C14C14	C14C11	C14C4	C11D13	C11D12	C11D5			
D13	D12	D5	D13D13	D13D12	D13D5	D12F6	D12F7				
F6	F7					F7A1	F7A2	F7A9			

Stanja I prelazi su preimenovani u odnosu na raniju sliku.
Prelazi su navedeni kao oznaka stanja na koju je nadovezana oznaka prelaza

Konstrukcija tabele izmena

Deo za 0-izmena:

- Za svako stanje uvodi se po jedna vrsta
- Uvodi se onoliko kolona koliko ima maksimalno izlaznih grana iz nekog stanja
- Navedu se sve izlazne grane iz posmatranog stanja u odgovarajućoj vrsti (mogu ostati i prazne ćelije ako ima manje izlaznih grana)

Konstrukcija tabele izmena

- Deo za 1-izmenu:
- Isti broj vrsta; za svaku kolonu tabele 0-izmena uvesti N kolona gde je N maksimalan broj izlaznih grana iz nekog stanja (N=3 u primeru)
- Kolone popuniti proširenjem odgovarajućih 0-izmena za još jedan prelaz. Npr. od ćelije B10 (prelaz 10 iz B u C) dodavanjem izlaznih grana iz C dobijaju se: B10C14, B10C11, B10C4

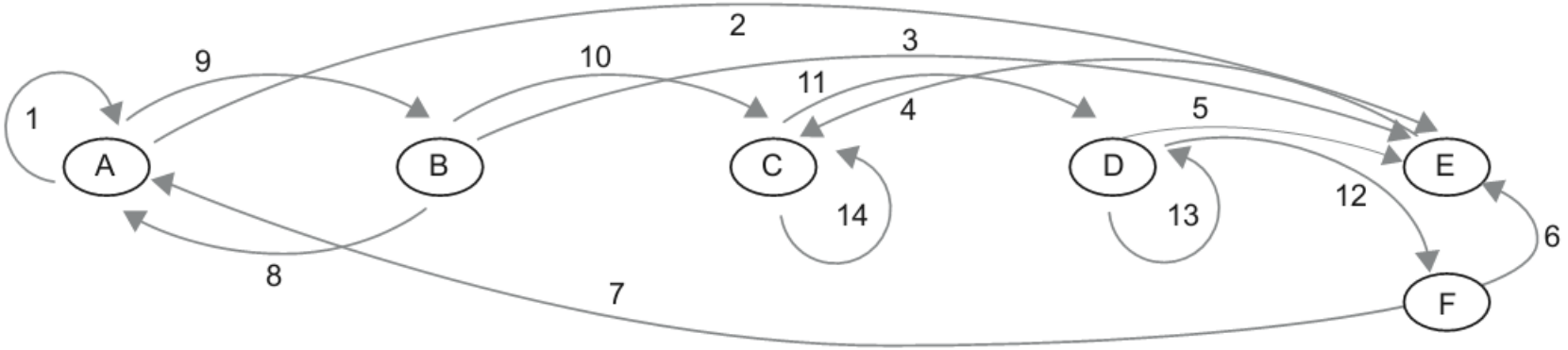
Testovi za pokrivanje N-izmena

1. Krenuti od skupa testova napravljenih po pravilima za pokrivanje stanja/prelaza.
2. Potvrditi ti testovi pokrivaju sve ćelije u kolonama 0-izmena (ako ne pokrivaju, napravljena je neka greška u prethodnim aktivnostima). Takođe naznačiti sve ulaze u kolonama N-izmena koji su pokriveni inicijalnim skupom testova.
3. Sada, izabrati neki nepokrivenu sekvencu sa N-izmena, recimo Y. Koristeći testove sa 0-izmena, konstruisati sekvencu prelaza X koja dostiže početno stanje sekvence Y.
   Nadovezati Y na X. Recimo da nova XY sekvenca završava u stanju S. Ako postoji još neka nepokrivena sekvenca (Z) sa N-izmena iz stanja S, nadovezati i tu sekvencu na XY, dobija se XYZ. Ako ne postoji, iz stanja S doći do početnog stanja neke nepokrivene sekvence W koristeći delove inicijalnog skupa testova (V). Zatim uključiti sledeću nepokrivenu sekvencu, dobija se XYVW. Sekvenca se može produžavati dok god je moguće locirati sekvence tipa Z ili W. Ako nismo mogli naći ovakve sekvence, onda postojeći test XY završiti sekvencom do nekog od prihvatljivih završnih stanja.
4. Za novonapravljeni test, označiti sekvence sa N-izmena koje on pokriva.
5. Ponavljati korake 3 i 4 dok se sve sekvence sa N-izmena ne pokriju.

Primer (el.kupovina)

Inicijalni skup testova za pokrivanje stanja/prelaza (pokriva svetlosive ulaze)

- (A1A9B8A9B10C14C11D13D12F7A2)
- (A9B3)
- (A9B10C4)
- (A9B10C11D5)

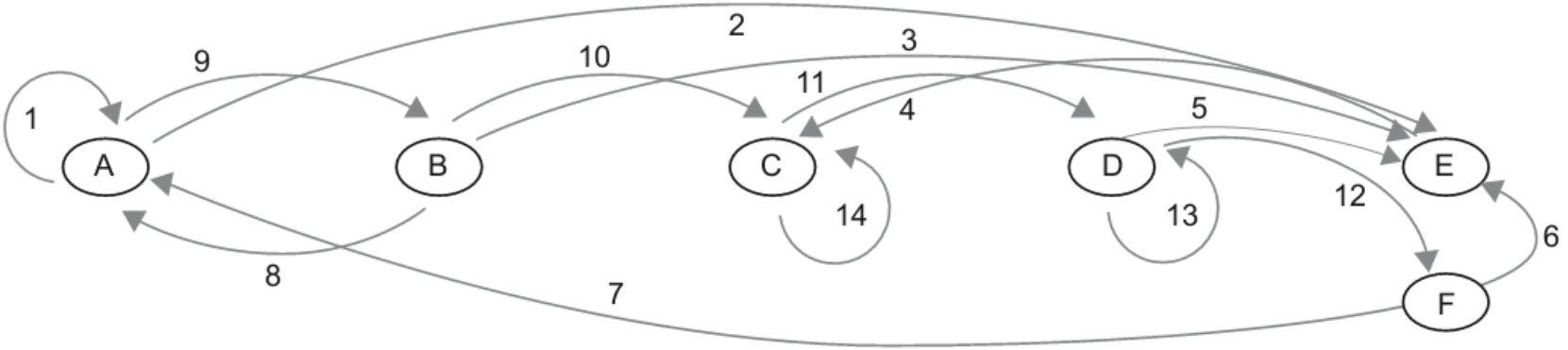


0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9				A9B10	A9B8	A9B3
B10	B8	B3	B10C14	B10C11	B10C4	B8A1	B8A2	B8A9			
C14	C11	C4	C14C14	C14C11	C14C4	C11D13	C11D12	C11D5			
D13	D12	D5	D13D13	D13D12	D13D5	D12F6	D12F7				
F6	F7					F7A1	F7A2	F7A9			

Primer (el.kupovina)

Dodatni testovi za preostale (tamnosive) ulaze:

- (A1A1A2).
- (A9B8A1A9B8A2).
- (A9B10C14C14C4).
- (A9B10C11D13D13D5).
- (A9B10C11D12F7A1A9B10C11D12F7A9).

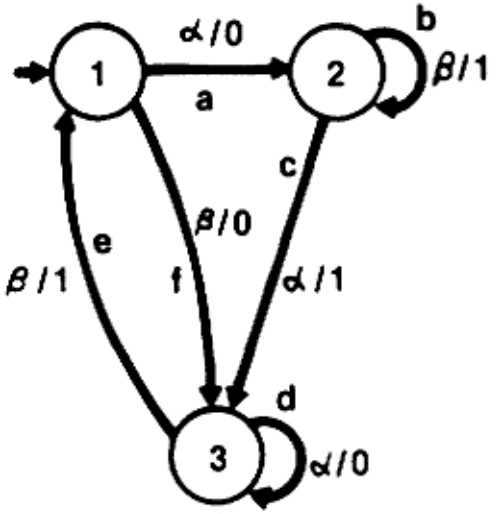


0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9				A9B10	A9B8	A9B3
B10	B8	B3	B10C14	B10C11	B10C4	B8A1	B8A2	B8A9			
C14	C11	C4	C14C14	C14C11	C14C4	C11D13	C11D12	C11D5			
D13	D12	D5	D13D13	D13D12	D13D5	D12F6	D12F7				
F6	F7					F7A1	F7A2	F7A9			

Pokrivanje skupova N-izmena (N-switch set cover)

- Definišemo pojam **skupa N-izmena** za proizvoljnu granu g kao skupa koji počinje sa g i ima još N sukcesivnih grana.
- Drugim rečima N-izmene grupišemo po početnoj grani (prelazu) i tako dobijamo skupove.
- Jedan skup N-izmena je pokriven ako imamo skup testova koji pokrivaju sve izmene u tom skupu do kojih se dolazi istovetnom putanjom iz startnog stanja.

Primer pokrivanja skupa N izmena



Machine D

Switches for Machine D:

ab, ac, bb, bc, cd, ce, dd, de, ea, ef, fd, fe

1- Switch Sets for Machine D:

{ab, ac} {bb, bc} {cd, ce}

{dd, de} {ea, ef} {fd, fe}

1- Switch Set Cover for Machine D:

β α α , β α β	fdd, fde pokriva {dd,de}	pokriva
β β α , β β β	fea, fef pokriva {ea,ef}	{fd,fe}
α α α , α α β	acd, ace pokriva {cd,ce}	pokriva
α β α , α β β	abc, abb pokriva {bb, bc}	{ab,ac}

Tabela prelaza

- Tabelarni prikazuje svih kombinacija stanja sa ulaznim događajima/uslovima
- Prikazuju se i dozvoljene i nedozvoljene kombinacije
- Na taj način analizira se šta se događa u nedozvoljenim i nedefinisanim situacijama

Konstrukcija tabele prelaza

- Prvo se izlistaju sva stanja i svi ulazi/uslovi prikazani na dijagramu stanja
- Zatim se kreira tabela koja ima red za svako stanje u kombinaciji sa svakim ulazom/uslovom:

Tekuće stanje	Ulaz /uslov	Akcija	Novo stanje

Primer (el. kupovina)

Znači mušterija može da uradi checkout samo neposredno posle dodavanja robe (ovo je nedostatak u specifikaciji)

Browsing	Continue shopping	Browsing
Selecting	Check out	Browsing
Logging	Login[bad]	Browsing
Purchasing	Login[good]	Browsing
Confirmed	Purchase[bad]	Browsing
Left	Purchase[good]	Browsing
	Abandon	Browsing
	Resume shopping	Browsing
	Go elsewhere	Selecting
		Left

× =

Current State	Event/cond	Action	New State
Browsing	Click link	Display	Browsing
Browsing	Add to cart	Selection dia	Selecting
Browsing	Continue shopping	Undefined	Undefined
Browsing	Check out	Undefined	Undefined
Browsing	Login[bad]	Undefined	Undefined
Browsing	Login[good]	Undefined	Undefined
Browsing	Purchase[bad]	Undefined	Undefined
Browsing	Purchase[good]	Undefined	Undefined
Browsing	Abandon	<no action>	Left
Browsing	Resume shopping	Undefined	Undefined
Browsing	Go elsewhere	Undefined	Undefined
Selecting	Click link	Undefined	Undefined
(Fifty-three rows, generated in the pattern shown above, not shown)			
Left	Go elsewhere	Undefined	Undefined

Pokrivanje tabele (0 izmena)

1. Odrediti inicijalni skup testova na osnovu pokrivanja dijagrama stanja za stanja/prelaze.
2. Konstruisati tabelu prelaza i verifikovati da su testovima pokriveni definisane vrste tabele.
3. Izabrati postojeći test koji posećuje neko stanje za koje u tabeli postoji jednu ili više nedefinisanih vrsta. Modifikovati test da se ubaci nedefinisani događaj/uslov za to stanje. Akcija u ovom slučaju je nedefinisana.
4. Označiti ovaj red tabele kao pokriven.
5. Ponavljati korake 3 i 4 sve dok se ne pokriju svi redovi tabele.

Primer (el. Kupovina)

- Da bismo pokrili situaciju

  Browsing, checkout, undefined, undefined

- Krenućemo od legalnog inicijalnog:

TP4. (browsing, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)

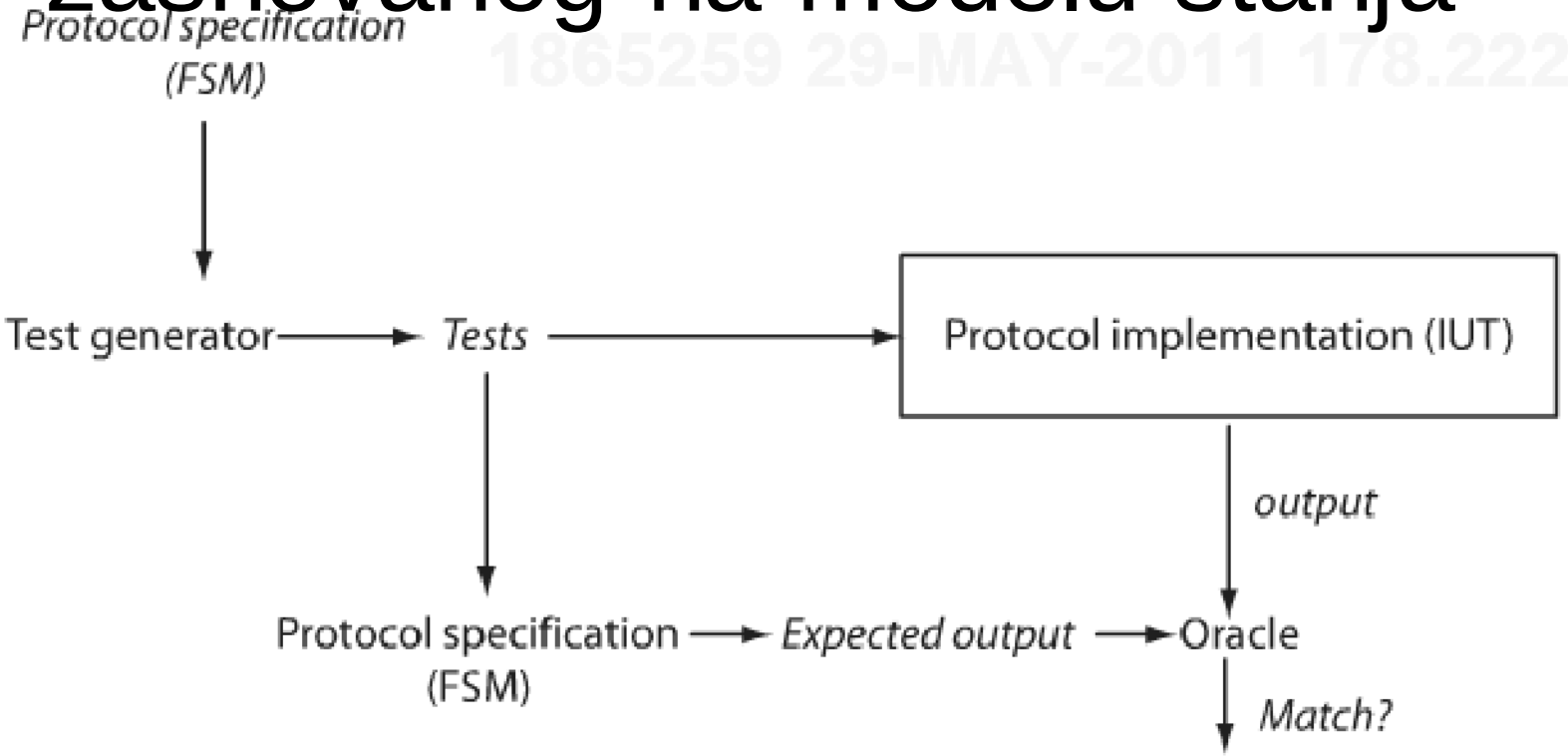
Modifikovaćemo ga na sledeći način:

- (browsing, *attempt: check out, action undefined*, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)
- Itd. Ideja je da svaki nedefinisani red tabele pokrijemo posebnim prelazom, slično kao kod klasa ekvivalencije, da ne bi jedna greška prikrila drugu (pitanje je da li bi sistem stigao do drugog prelaza)

Ponašanje sistema za nedefinisane prelaze

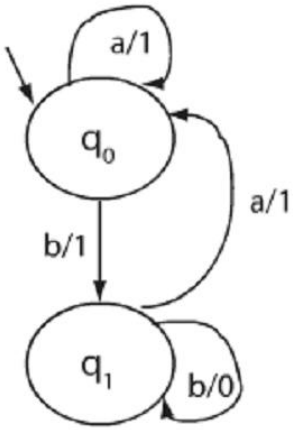
- Koje bi bilo najbolje ponašanje sistema za ovakve prelaze?
- Najbolji scenario je da je ovaj događaj/uslov nemoguće okinuti (nema menija, dugmadi, hotkey-jeva ili editovanog urla)
- Sledeći scenario je da sistem reaguje na ovakav događaj tako što ga ignoriše ili prikazuje smislenu poruku o grešci, a zatim nastavlja kao da događaja nije bilo.
- Svaki drugi ishod treba smatrati bagom.

Validnost kriterijuma testiranja zasnovanog na modelu stanja

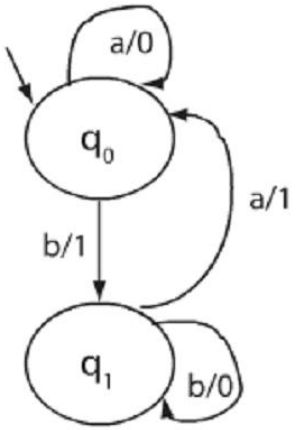


- Specifikacija (FSM) i implementacija (IUT) mogu se razlikovati
- Da li su kriterijumi validni, tj. da li garantuju da će svako odstupanje biti detektovano ?

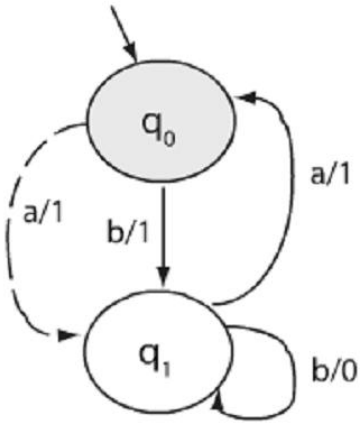
Model defekata (fault model)



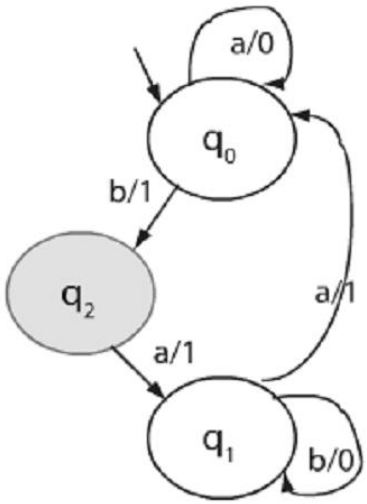
M



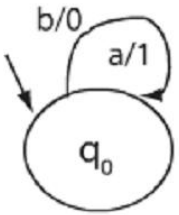
M1: greška operacije



M2: greška prelaza



M3: suvišno stanje

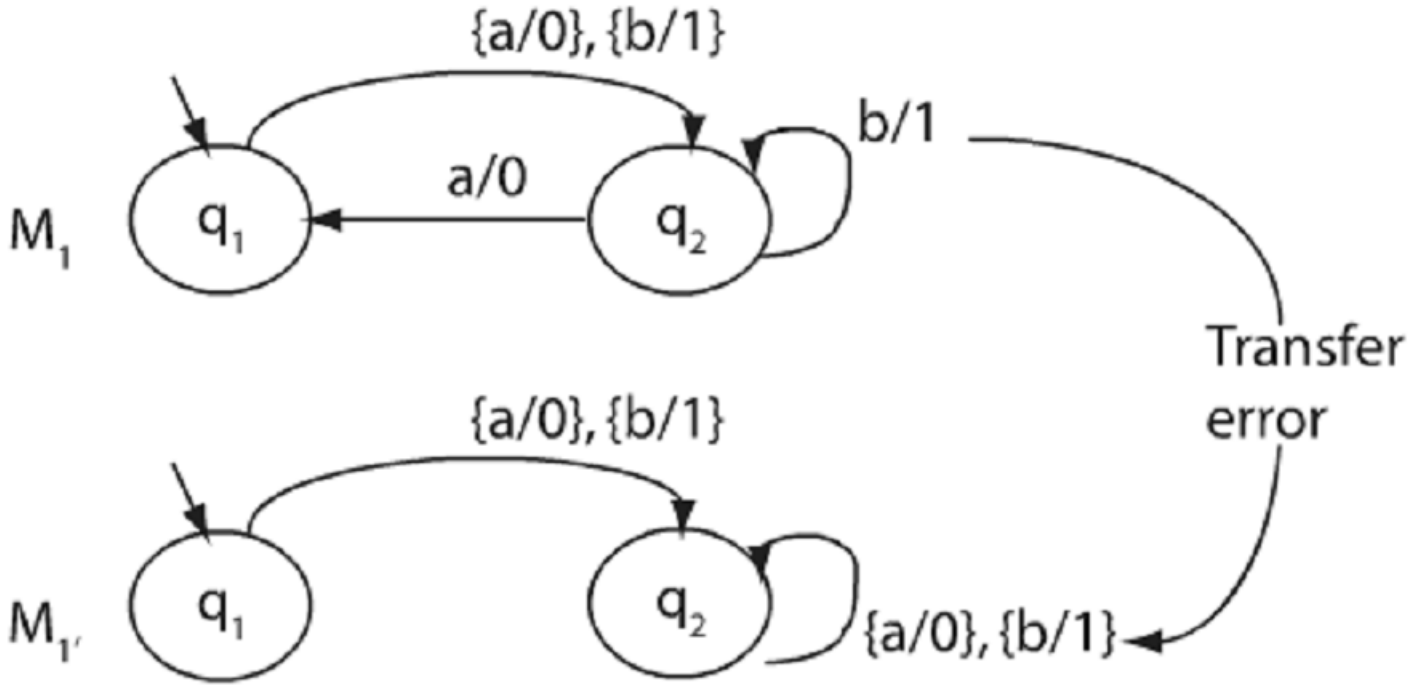


M4: stanje nedostaje

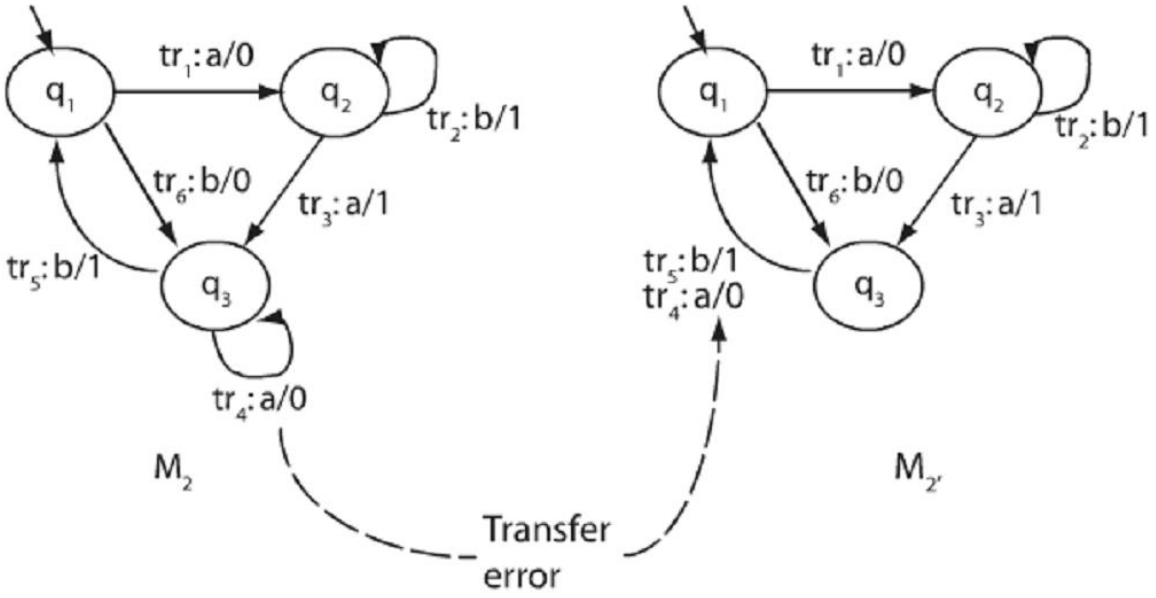
- Definiše skup pretpostavljenih defekata u implementaciji u odnosu na model
- Zajednički naziv: greške sekvencionisanja
- Koristi se za evaluaciju validnosti kriterijuma

Pokrivanje stanja/prelaza

- Ne garantuje otkrivanje svih defekata samo na osnovu izlaza:
- M1’ ima pogrešan prelaz u stanju q2
- Tp = abba, pokriva sva stanja i prelaze obe mašine daju isti izlaz 0110



Pokrivanje N-izmena



- Ne garantuje otkrivanje svih grešaka

Test sequence (t)	$O_{M2}(q_1, t)$ ($= O_{M2'}(q_1, t)$)	Switches covered
abbaaab	0111001	$(tr_1, tr_2), (tr_2, tr_2), (tr_2, tr_3), (tr_3, tr_4), (tr_4, tr_4), (tr_4, tr_5)$
aaba	0110	$(tr_1, tr_3), (tr_3, tr_5), (tr_5, tr_1)$
aabb	0110	$(tr_1, tr_3), (tr_3, tr_5), (tr_5, tr_6)$
baab	0001	$tr_6, tr_4), (tr_4, tr_4), (tr_4, tr_5)$
bb	01	(tr_6, tr_5)

Pokrivanje skupova N-izmena

- Jeste validan za model defekata koji smo usvojili, ako se svi parovi stanja automata N-razlikuju (N-distinguishable), dakle garantovano otkriva sve greške sekvenciranja.
- Par stanja S1 i S2 se N-razlikuje ako postoji sekvenca ulaza dužine N tako da automat, počev od stanja S1 sekvencu prihvata, a počev od stanja S2 sekvencu odbija.
- Dakle ako je stanja 1-razlikuju, dovoljno je uzeti pokrivanje skupova 1-izmena.
- U najgorem slučaju, minimalni automat sa N stanja ima N-1 distinguishable stanja.

Testiranje sintakse

Testiranje sintakse

- Radi se o tehnici crne kutije
- Potrebno je analizirati funkcionalnu specifikaciju softvera da bi se modelovalo njegovo ponašanje opisom sintakse ulaznih podataka
- Tehnika je onoliko efikasna koliko opis sintakse prema kojem se testira odgovara stvarnom ponašanju komponente

Primer

- Razmotrimo modul koji proverava da li vrednost ulazne promenljive **float_in** odgovara sintaksi broja u pokretnom zarezu. Izlaz (promenljiva check_res) uzima vrednosti valid ili invalid u zavisnosti od validnosti sintakse

Primer

- Opis sintakse broja u pokretnom zarezu u Backus Naur-ovoj formi (BNF):

```
float = int "e" int.
int = ["+"|"-"] nat.
nat = {dig}.
dig = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".
```

- Terminali su prikazani pod navodnicima – to su elementarni delovi sintakse ulaza, tj. znaci od kojih se sastoji ulaz.
- | je sint. operator koji razdvaja alternative
- [ ] ograđuje neobavezni element (može se izostaviti)
- { } uokviruje element sintakse koji se u ulazu može pojaviti jednom ili više puta.

Test primeri za validnu sintaksu

- Prvo ćemo izvesti opcije iz sintaksnog opisa; iza opcije stavljamo oznaku [opt_i], da bismo je kasnije mogli identifikovati.

  float nema opcija

  int ima tri opcije: nat [opt_1], “+” nat [opt_2] i “-” nat [opt_3]

  nat ima dve opc: jednocifren broj [opt_4] i višecif broj [opt_5]

  dig ima 10 opcija: po jednu za svaku cifru [opt_6-opt_15]

- Treba konstruisati po jedan test primer za svaku od 15 dobijenih opcija

Test primeri za validnu sintaksu

test case	float_in	option(s) executed	check_res
1	3e2	opt_1	'valid'
2	+2e+5	opt_2	'valid'
3	-6e-7	opt_3	'valid'
4	6e-2	opt_4	'valid'
5	1234567890e3	opt_5	'valid'
6	0e0	opt_6	'valid'
7	1e1	opt_7	'valid'
8	2e2	opt_8	'valid'
9	3e3	opt_9	'valid'
10	4e4	opt_10	'valid'
11	5e5	opt_11	'valid'
12	6e6	opt_12	'valid'
13	7e7	opt_13	'valid'
14	8e8	opt_14	'valid'
15	9e9	opt_15	'valid'

Test primeri za ispravnu sintaksu

- Testovi pokrivaju i druge opcije osim onih za koje su pravljeni.
- Ovaj skup TP nije minimalan (dovoljni bi bili npr. 2, 3 i 5 da pokriju sve opcije).
- Međutim, ako se ovako napravi za svaku opciju posebno lakše je ispraviti potencijalnu grešku ako se testiranjem uoči.

Test primeri za neispravnu sintaksu

- Prvo definišemo listu generiških mutacija, na primer:
  - m1. uvodi nedozvoljenu vrednost za sint. element
  - m2. menja element drugim definisanim elementom
  - m3. izostavlja potreban element
  - m4. dodaje suvišan element

Elementi za sintaksu float-a

- U BNF reprezentaciji obeležićemo pojedine sintaksne elemente sa el_i

float = int "e" int.	el_1 = el_2 el_3 el_4.
int = ["+"|"-"] nat.	el_5 = el_6 el_7.
nat = {dig}.	el_8 = el_9.
dig = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".	el_10 = el_11.

- Napomena: ["+"|"-"] tretiramo kao jedan element pošto mutacije nad pojedinačnim opcionim elementima ne prave test primere sa neispravnom sintaksom koristeći definisani skup mutacija

Test primeri za neispravnu sintaksu

- Sistematskom primenom definisanih mutacija na pojedine elemente sintakse, dobijamo skup test primera

Test primeri za neispravnu sintaksu

test case	float_in	mutation	element	check_res
1	xe0	m1	x for el_2	'invalid'
2	0x0	m1	x for el_3	'invalid'
3	0ex	m1	x for el_4	'invalid'
4	x0e0	m1	x for el_6	'invalid'
5	+xe0	m1	x for el_7	'invalid'
6	ee0	m2	el_3 for el_2	'invalid'
7	+e0	m2	el_6 for el_2	'invalid'
8	000	m2	el_2 for el_3	'invalid'
9	0+0	m2	el_6 for el_3	'invalid'
10	0ee	m2	el_3 for el_4	'invalid'
11	0e+	m2	el_6 for el_4	'invalid'
12	e0e0	m2	el_3 for el_6	'invalid'

Test primeri za neispravnu sintaksu (nastavak)

test case	float_in	mutation	element	check_res
13	+ee0	m2	el_3 for el_7	'invalid'
14	++e0	m2	el_6 for el_7	'invalid'
15	e0	m3	el_2	'invalid'
16	00	m3	el_3	'invalid'
17	0e	m3	el_4	'invalid'
18	y0e0	m4	y in el_1	'invalid'
19	0ye0	m4	y in el_1	'invalid'
20	0ey0	m4	y in el_1	'invalid'
21	0e0y	m4	y in el_1	'invalid'
22	y+0e0	m4	y in el_5	'invalid'
23	+y0e0	m4	y in el_5	'invalid'
24	+0yeo	m4	y in el_5	'invalid'

Test primeri za neispravnu sintaksu

- Napomene: neke mutacije daju sintaksno ispravne izraze i one su zanemarene.
  - Na primer, m2 (promena el_4 sa el_2) daje korektnu sintaksu jer su i el_2 i el_4 isti (int)
- Neke mutacije daju isti rezultat pa su pokrivene samo jednim test primerom
  - Na primer, primenom m1 menjamo el_4 sa “+”, dobijamo ulaz “0e+”, što je isto kao u TP11

Test primeri za neispravnu sintaksu

- Kombinovanjem mutacija, ili izborom drugih vrednosti kada se primenjuju pojedinačne mutacije, može se dobiti još veliki broj drugih test primera.