

TESTIRANJE NA OSNOVU TABELA ODLUČIVANJA

Tabele odlučivanja

- U softverskom inženjerstvu, prevođenje neformalnih poslovnih zahteva (specifikacija) u izvorni kod predstavlja kritičnu tačku u kojoj najčešće nastaju greške.
- **Tabela odlučivanja** (Decision Table) je moćan formalni model za specifikaciju, analizu i testiranje kompleksne poslovne logike.
- Za razliku od tekstualnih opisa, tabele omogućavaju striktnu matematičku verifikaciju sistema pre nego što se napiše prva linija koda.

Struktura tabele odlučivanja

USLOVI (Inputs)	P1	P2	P3	P4
U1: Korisnik je registrovan	T	T	F	-
U2: Prva kupovina	T	F	-	T
U3: Korpa > 50€	T	T	F	F
U4: Nivo članstva (Extended)	VIP	Gold	Silver	Basic
AKCIJE (Outputs)				
A1: Odobri popust 20%	✓	✓	-	-
A2: Odobri besplatnu dostavu	✓	-	✓	-
A3: Pošalji poklon vaučer	-	✓	-	✓

Oznake: T = True (Tačno) F = False (Netačno) - = Nije bitno (Don't care) ✓ = Akcija se izvršava

1. GORNJI SEGMENT

USLOVI (Inputs):

Koji faktori utiču na donošenje odluke?

PRAVILA (Rules):

Kombinacije stanja uslova.

U4: Umesto T/F sadrži tekst (Extended Entry)

2. DONJI SEGMENT

AKCIJE (Outputs):

Kako sistem/softver treba da reaguje?

ISHODI (Actions Matrix):

Koje akcije se tačno okidaju za koje pravilo.

Matematički model tabele odlučivanja

- Prvo ćemo formalno definisati gornji segment tabele, to jest prostor uslova i prostor stanja.
- Neka je $C = \{c_1, c_2, \dots, c_n\}$ konačan skup od n **uslova**.
- Svaki uslov c_i ima svoj konačan domen vrednosti $D_i = \{u_{i1}, u_{i2}, \dots, u_{ik}\}$. Ako su domeni binarni ($D_i = \{T, F\}$), reč je o tabeli sa ograničenim ulazima (*Limited Entry*). Ako domeni sadrže više od dve vrednosti, tabela je sa proširenim ulazima (*Extended Entry*).
- **Prostor stanja** (Univerzum diskursa): Skup svih teorijski mogućih kombinacija uslova predstavlja Dekartov proizvod pojedinačnih domena:

$$\mathbb{U} = D_1 \times D_2 \times \dots \times D_n$$

- Svaki element $u \dots \in \mathbb{U}$ je jedna potpuno specifikovana kombinacija vrednosti uslova. Ukupan broj elemenata u univerzumu iznosi $|\mathbb{U}| = \prod_{i=1}^n |D_i|$.
- Za binarne uslove $|\mathbb{U}| = 2^n$

Formalna definicija pravila

- Tabela se sastoji od skupa pravila $R = \{R_1, R_2, \dots, R_m\}$, konačan skup svih raspoloživih akcija u sistemu. Svako pravilo R_j u tabeli je uređeni par

$$R_j = \langle K_j, A_j \rangle$$

- K_j je **konjukcija uslova** pravila R_j . Može se posmatrati kao podskup prostora stanja ($K_j \subseteq \mathbb{U}$).
Ako pravilo R_j za uslov C_i sadrži vrednost "-" ("don't care"), tada komponenta domena D_i u tom pravilu uzima celokupan skup D_i .
- A_j je skup akcija koje se izvršavaju kada je konfiguracija uslova unutar K_j .
- Iz ugla iskaznog računa, pravilo R_j je implikacija: $K_j \Rightarrow A_j$.

Verifikacija tabele odlučivanja

- Uspešna verifikacija tabele podrazumeva ispunjenje dva ključna kriterijuma:
 - Kompletnost
 - Konzistentnost (nedvosmislenost)

A) Kompletnost

- Tabela je **kompletna** ako i samo ako je unija svih prostora uslova K_j jednaka celokupnom stanja $\mathbb{U}$. To znači da ne postoji nijedna kombinacija uslova za koju tabela nema predviđeno pravilo.

$$\bigcup_{j=1}^m K_j = \mathbb{U}$$

- **Formalna provera:** Ako uslove posmatramo kao iskazne formule, tabela je kompletna ako je disiunkcija svih uslova tautologija:

$$\models \bigvee_{j=1}^m K_j$$

- **Napomena:** Nedostajuća stanja se moraju identifikovati i dopuniti (najčešće defanzivnom akcijom "Bez akcije" ili "Greška").

A) Konzistentnost (nedvosmislenost)

- Tabela odlučivanja je konzistentna ako i samo ako za bilo koja dva različita pravila R_x i R_y , čiji se prostori uslova preklapaju, ta dva pravila impliciraju identične akcije.

$$\forall x, y \in \{1, \dots, m\} \quad (K_x \cap K_y \neq \emptyset) \implies (A_x = A_y)$$

- Ako je $K_x \cap K_y \neq \emptyset$, a $A_x \neq A_y$, tabela je **konfliktna (kontradiktorna)** jer bi za isto stanje iz preseka sistem trebalo da izvrši različite akcije.
- Ako je $K_x \cap K_y \neq \emptyset$ i $A_x = A_y$, tabela je **redundantna** (isto ponašanje je opisano na više mesta).
- Tabela ima **strogo disjunktna pravila** ako je presek prostora uslova za bilo koja dva pravila prazan skup: $\forall x \neq y, K_x \cap K_y = \emptyset$.

Algoritam kontrakcije tabele (redukcije redundantnih pravila)

- Često su specifikacije preobimne jer sadrže uslove koji u određenim situacijama nemaju uticaj na krajnji ishod (redundansa). Da bi se logika pojednostavila, primenjuje se adaptacija Quine-McCluskey algoritma.
- **Zakon redukcije:**
- Dva ili više pravila se mogu spojiti u jedno redukovano pravilo ako i samo ako:
 1. Aktiviraju potpuno identičan skup akcija ($A_x = A_y$).
 2. Svi uslovi su im identični, osim jednog uslova c_i , ali tako da ta pravila zajedno u potpunosti pokrivaju ceo domen D_i tog uslova.
- Kada se ovaj uslov ispuni, vrednosti uslova se eliminišu iz pravila i zamenjuju oznakom irelevantnosti ("-" / don't care).
- Proces se vrši iterativno sve dok je moguće pronalaziti nove parove ili klastere za spajanje. Rezultat je minimalna struktura spremna za prevođenje u programski kôd.

Primer: Automatsko odobravanje popusta i poklona

Kontekst sistema:

- Treba testirati modul za obračun benefita u korpi e-commerce sistema. Sistem na osnovu profila korisnika donosi odluku o pogodnostima koje se aktiviraju.

Ulazni parametri (Uslovi):

Status naloga: Korisnik može imati status *Zlato* ili *Standard*.

Tip korpe: Korpa može biti sa oznakom *Premium* ili *Obična*.

Primer: Automatsko odobravanje popusta i poklona

Poslovna pravila iz specifikacije:

1. Ako korisnik ima status **Zlato**, a korpa je **Premium**, sistem mu dodeljuje besplatnu dostavu i šalje poklon uzorak.
2. Ako korisnik ima status **Zlato**, a korpa je **Obična**, sistem mu takođe dodeljuje besplatnu dostavu i šalje poklon uzorak.
3. Ako korisnik ima status **Standard**, a korpa je **Premium**, sistem mu dodeljuje besplatnu dostavu (ali bez poklon uzorka).

Primer: Automatsko odobravanje popusta i poklona

Zadatak:

1. **Analiza kompletnosti:** Konstruisati inicijalnu tabelu, uočiti stanje koje nedostaje i dopuniti ga akcijom ["Bez benefita"].
2. **Konzistentnost:** Proveriti da li ima konfliktnih preklapanja.
3. **Redukcija (Kontrakcija):** Primeniti algoritam na tabli i svesti tabelu na minimalni broj pravila.

Rešenje 1. Formalizacija i provera kompletnosti

Definišemo domene uslova:

- D_1 (Status) = {Zlato, Standard}
- D_2 (Korpa) = {Premium, Obična}

Ukupan broj stanja u univerzumu je $|\mathbb{U}| = 2 \times 2 = 4$.

Iz tekstualne specifikacije izvađena su sledeća pravila:

- R_1 : (Zlato, Premium) $\implies$ [Dostava, Poklon]
- R_2 : (Zlato, Obična) $\implies$ [Dostava, Poklon]
- R_3 : (Standard, Premium) $\implies$ [Dostava]

Zaključak o kompletnosti: Tabela **nije kompletna**. Unija pokrivenih stanja ima 3 elementa. Nedostaje stanje (Standard, Obična).

Rešenje 2. Dopuna i konzistentnost

Autodopuna: Dodajemo odbrambeno pravilo R_4 za nedostajuću kombinaciju:

- R_4 : (Standard, Obična) $\Rightarrow$ [Bez benefita]

Sada imamo **kompletnu elementarnu tabelu**:

Pravilo	Status	Korpa	Akcije
R_1	Zlato	Premium	Dostava, Poklon
R_2	Zlato	Obična	Dostava, Poklon
R_3	Standard	Premium	Dostava
R_4	Standard	Obična	Bez benefita

Tabela je **konzistentna** jer svaka od 4 jedinstvene kombinacije ulaza vodi do tačno jednog definisanog skupa akcija (nema preklapanja).

Rešenje 3. Primena algoritma redukcije

Zakon redukcije glasi: *Ako dva pravila aktiviraju identičan skup akcija, a razlikuju se na poziciji samo jednog uslova čiji domen zajedno u potpunosti pokrivaju, taj uslov se eliminiše i menja oznakom " - ".*

1. Grupisanje po akcijama:

- **Grupa A (Akcije: [Dostava, Poklon]):** Sadrži pravila $\{R_1, R_2\}$
- **Grupa B (Akcije: [Dostava]):** Sadrži pravilo $\{R_3\}$
- **Grupa C (Akcije: [Bez benefita]):** Sadrži pravilo $\{R_4\}$

Napomena za studente: Redukcija se može vršiti **isključivo** unutar iste grupe. Pošto Grupa B i Grupa C imaju samo po jedno pravilo, ona su već u svom minimalnom obliku. Fokusiramo se na Grupu A.

Rešenje 3. Primena algoritma redukcije (nastavak)

Upoređujemo vektore uslova za pravila iz Grupe A:

- R_1 : (Zlato, Premium)
- R_2 : (Zlato, Obična)

1. **Analiza pozicije 1 (Status):** Vrednosti su identične (Zlato i Zlato). Razlika = 0.
2. **Analiza pozicije 2 (Korpa):** Vrednosti se razlikuju (Premium naspram Obična). Razlika = 1.

Ukupan broj razlika između ova dva pravila je **tačno 1**, na poziciji uslova "Korpa". Domen uslova "Korpa" je $D_2 = \{\text{Premium, Obična}\}$. Pošto pravila R_1 i R_2 u sebi sadrže oba ova elementa, oni **kompletno pokrivaju domen tog uslova**.

Rešenje 3. Primena algoritma redukcije (nastavak)

Spajamo R_1 i R_2 u jedno redukovano pravilo, gde na mestu uslova "Korpa" upisujemo crticu "-":

$$R_{1,2} : (\text{Zlato}, -) \implies [\text{Dostava}, \text{Poklon}]$$

Originalna pravila R_1 i R_2 su uspešno iskorišćena i brišu se iz finalnog skupa.

Finalna minimalna tabela odlučivanja

Sakupljanjem novonastalog pravila iz Grupe A i nepromenjenih pravila iz Grupa B i C, dobijamo optimizovanu tabelu smanjenu sa 4 na **3 pravila**:

Pravilo	Status	Korpa	Akcije	Objašnjenje pravila
$R_{1,2}$	Zlato	—	Dostava, Poklon	Ako je kupac Zlato, tip korpe je potpuno nebitan.
R_3	Standard	Premium	Dostava	Standardni kupac dobija samo dostavu ako je korpa Premium.
R_4	Standard	Obična	Bez benefita	Standardni kupac sa običnom korpom ne dobija ništa.

Automatizacija testiranja na osnovu tabela odlučivanja

- **Cucumber** (za Java, JavaScript, Ruby) i **SpecFlow** (za .NET okruženje) su vodeći alati za BDD (**Behavior-Driven Development**) koji omogućavaju da se tabele odlučivanja direktno pretoče u izvršne automatizovane testove.
- Ovi alati ne služe samo za dokumentovanje, već *direktno pokreću aplikaciju i proveravaju njen rad* na osnovu tabela koje napišete. U Cucumber/SpecFlow svetu, tabela odlučivanja se realizuje kroz sintaksu Scenario Outline i Examples tabelu.
- U nastavku je kompletan primer razvoja jednog modula: od poslovnog zahteva, preko optimizacije tabele, do gotovog koda.

Primer: Specifikacija ponašanja (Poslovni problem)

Pravimo sistem za **odobrenje kredita** u banci na osnovu tri uslova:

1. Da li klijent ima **stalan posao**? (Da/Ne)
2. Da li je **kreditno sposoban** (nema dugovanja)? (Da/Ne)
3. Da li ima **žiranta**? (Da/Ne)

Biznis pravila za odobrenje:

- Ako klijent ima stalan posao i čist kreditni biro – kredit se **odobrava** (žirant nije potreban).
- Ako klijent ima stalan posao, ali ima dugovanja – kredit se odobrava **samo ako ima žiranta**.
- Ako klijent nema stalan posao – kredit se **odbija** u svim slučajevima (žirant ne pomaže).

Primer: Formulisanje i optimizacija tabele odlučivanja

Pravilo (Rule)	Stalan posao	Kreditno sposoban	Ima žiranta	Očekivani ishod (Kredit)	Obrazloženje / Optimizacija
R1	Da	Da	–	Odobren	Ako je sve čisto, žirant je nebitan (Don't Care).
R2	Da	Ne	Da	Odobren	Ima posao, ima dug, ali žirant spašava stvar.
R3	Da	Ne	Ne	Odbijen	Ima posao, ima dug, nema žiranta.
R4	Ne	–	–	Odbijen	Nema posao – sve ostalo postaje nebitno (Don't Care). Spojena 4 pravila u jedno.

Primer: Pisanje kredit.feature fajla na jeziku Gherkin

Feature: Odobrenje bankarskog kredita

Scenario Outline: Evaluacija zahteva za kredit na osnovu tabele odluciva

Given klijent aplicira za kredit

And klijent ima stalan posao postavljen na "<stalan_posao>"

And klijent je kreditno sposoban postavljen na "<kreditno_sposoban>"

And klijent ima ziranta postavljen na "<ima_ziranta>"

When sistem procesira zahtev za kredit

Then status kredita treba da bude "<ocekivani_ishod>"

Examples:

stalan_posao	kreditno_sposoban	ima_ziranta	ocekivani_ishod
Da	Da	Ne	ODOBREN
Da	Ne	Da	ODOBREN
Da	Ne	Ne	ODBIJEN
Ne	Da	Da	ODBIJEN

Promenljive iz tabele
stavljamo u zagrade
< >

U Gherkin tabelama ne
možemo lako staviti
prazno crticu "-" za
Don't Care, pa smo za
R1 i R4 uneli
proizvoljne vrednosti
koje kod treba da
ignoriše u skladu sa
optimizacijom.

Primer: Java Kod Aplikacije (KreditniSistem.java)

Ovo je stvarni produkcionni kod (ili mock servis) koji simulira biznis logiku koju testiramo na osnovu optimizovane tabele odlučivanja.

```
package com.banka;

public class KreditniSistem {

    public String procesirajKredit(boolean imaPosao, boolean kreditnoSposoban, boolean imaZiranta) {
        // R4: Nema stalan posao -> ODBIJEN (ostali uslovi se ignorišu)
        if (!imaPosao) {
            return "ODBIJEN";
        }
        // R1: Ima posao i čist kreditni biro -> ODOBREN (žirant se ignoriše)
        if (kreditnoSposoban) {
            return "ODOBREN";
        }
        // R2 & R3: Ima posao, ali ima dugovanja -> zavisi isključivo od žiranta
        if (imaZiranta) {
            return "ODOBREN";
        }

        return "ODBIJEN";
    }
}
```

Primer: Definicija koraka (KreditSteps.java)

```
import com.banka.KreditniSistem;  
import io.cucumber.java.en.Given;  
import io.cucumber.java.en.When;  
import io.cucumber.java.en.Then;  
import org.junit.jupiter.api.Assertions;
```

```
public class KreditSteps {
```

```
    private KreditniSistem sistem;  
    private boolean stalanPosao;  
    private boolean kreditnoSposoban;  
    private boolean imaZiranta;  
    private String stvarniIshod;
```

```
    @Given("klijent aplicira za kredit")  
    public void klijent_aplicira_za_kredit() {  
        sistem = new KreditniSistem();  
    }
```

```
    @Given("klijent ima stalan posao postavljen na {string}")  
    public void klijent_ima_stalan_posao_postavljen_na(String  
vrednost) {  
        this.stalanPosao = vrednost.equalsIgnoreCase("Da");  
    }
```

```
    @Given("klijent je kreditno sposoban postavljen na {string}")  
    public void klijent_je_kreditno_sposoban_postavljen_na(String  
vrednost) {  
        this.kreditnoSposoban = vrednost.equalsIgnoreCase("Da");  
    }
```

```
    @Given("klijent ima ziranta postavljen na {string}")  
    public void klijent_ima_ziranta_postavljen_na(String vrednost) {  
        this.imaZiranta = vrednost.equalsIgnoreCase("Da");  
    }
```

```
    @When("sistem procesira zahtev za kredit")  
    public void sistem_procesira_zahtev_za_kredit() {  
        // Prosledujemo parametre iz trenutnog reda tabele u sistem  
        this.stvarniIshod = sistem.procesirajKredit(stalanPosao,  
            kreditnoSposoban, imaZiranta);  
    }
```

```
    @Then("status kredita treba da bude {string}")  
    public void status_kredita_treba_da_bude(String ocekivaniIshod) {  
        // Provera poklapanja stvarnog ishoda koda i ocekivanog  
        // ishoda iz tabele odlucivanja  
        Assertions.assertEquals(ocekivaniIshod, stvarniIshod,  
            "Ocekivani ishod se ne poklapa sa rezultatom sistema!");  
    }
```

Ova klasa hvata podatke iz svakog reda tabele iz .feature fajla, mapira ih u Java tipove podataka (boolean i String) i vrši asertaciju rezultata pomoću JUnit 5 Assertions klase.

Primer: pokretač testova (RunCucumberTest.java)

- Da bi JUnit 5 prepoznao Cucumber testove pri pokretanju komande mvn test, potrebna je prazna klasa sa anotacijama za konfiguraciju.

```
package com.bank;
```



```
import org.junit.platform.suite.api.ConfigurationParameter;  
import org.junit.platform.suite.api.IncludeEngines;  
import org.junit.platform.suite.api.SelectClasspathResource;  
import org.junit.platform.suite.api.Suite;  
import static io.cucumber.core.options.Constants.PLUGIN_PROPERTY_NAME;
```



```
@Suite  
@IncludeEngines("cucumber")  
@SelectClasspathResource("features")  
// Opciono: Dodavanje plugin-a za generisanje preglednog HTML izveštaja o testiranju tabele  
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value = "pretty, html:target/cucumber-  
reports.html")  
public class RunCucumberTest {  
    // Klasa ostaje prazna, služi samo kao ulazna tačka za JUnit  
}
```