SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

PROGRAMSKO SUČELJE ZA PRIRODNU DEDUKCIJU

Diplomski rad

IVICA ŠKROBO

Osijek, 2016.

Sadržaj

1. Uvod 1

1.1 Zadatak diplomskog rada 1

2. Prirodna dedukcija 2

2.1 Objašnjenje jezika i abecede propozicijske logike 2

2.1.1 Abeceda: 2

2.1.2 Jezik 2

2.2 Sustav prirodne dedukcije 4

2.3 Pravila prirodne dedukcije 5

2.3.1 Pravilo uvođenja konjukcije (Intro ∧) 5

2.3.2 Pravilo eliminiranja konjukcije (Elim ∧) : 5

2.3.3 Pravilo uvođenja disjunkcije (Intro ∨) 6

2.3.4 Pravilo eliminacije disjunkcije (Elim ∨) 6

2.3.5 Pravilo uvođenja kondicionala (Intro →) 7

2.3.6 Pravilo eliminiranja kondicionala (Elim →) 7

2.3.7 Pravilo uvođenja bikondicionala (Intro ↔) 8

2.3.8 Pravilo eliminiranja bikondicionala (Elim ↔) 9

2.3.9 Pravilo uvođenja negacije (Intro ¬) 9

2.3.10 Pravilo eliminiranja negacije (Elim ¬) 9

3. Android SUSTAV 10

3.1 Povijest Android operacijskog sustava 10

3.1.1 Verzije Android operativnog sustava 10

3.2 Android Studio 11

3.3 Java 12

3.3.1 Osnove Java jezika 12

4. ANDROID APLIKACIJA 14

4.1 Priprema razvojnog okruženja 14

4.2 Aktivnosti aplikacije 14

4.2.1 HelloActivity 15

4.2.2 SetFormulaActivity 17

4.2.3 MainActivity 25

4.2.4 EndActivity 39

5. ZAKLJUČAK 41

1. UVOD

Prirodna dedukcija je kao što samo ime govori, prirodan način za dedukcijsko zaključivanje čovjeka. Sama riječ dedukcija dolazi iz latinske riječi deductio, odnosno izvođenje. U tradicionalnoj logici, dedukcijom se nazivao samo postupak izvođenja pojedinačnoga iz općenitoga. No danas pojam dedukcije obuhvaća svaki postupak kojim se implicitni sadržaj jednih premisa koristi za oblikovanje drugih.

Prirodna dedukcija sadrži pravila "ispravnog razmišljanja". Ta pravila su formalni analog neformalnog načina dokazivanja u matematici. Odnosno omogućuju nam ispitivanje neformalnog dokaza u formalnom sustavu. Formalnost dokaza određena je njegovim načinom zapisa. Formalni dokazi zapisani su u potpunosti pomoću logičkih simbola, dok su neformalni dokazi zapisani u jednom od jezika.

1.1 Zadatak diplomskog rada

Cilj ovog diplomskog rada je napraviti sučelje u kojemu će biti moguće provjeriti unešene tvrdnje koristeći prirodnu dedukciju. Kako bi se cilj ostvario potrebno je postaviti pravila prirodne dedukcije na način da ih računalo može interpretirati. Potrebno je omogućiti korisniku programa jednostavno korištenje sučeljem, te jednostavan unos novih parametara odnosno tvrdnji dedukcije.

2. PRIRODNA DEDUKCIJA

2.1 Objašnjenje jezika i abecede propozicijske logike

Abeceda propozicijske logike sadrži skup veznika, prebrojivi skup propozicijskih varijabli i skup pomoćnih simbola.

2.1.1 Abeceda:

1. Skup veznika :

- $\wedge$ "i" , konjunkcija
- $\vee$ "ili", disjunkcija
- $\rightarrow$, kondicional
- $\leftrightarrow$, bikondicional
- $\neg$, $-$ negacija
- $\forall$, univerzalni kvantifikator
- $\exists$, egzistencijalni kvantifikator
- $=$, kvantifikator jednakosti
- $\models$, tautologija

2. Logičke konstante :

- $\top$, istina (grčko slovo Tau)
- $\bot$, neistina (non-Tau)

2. Prebrojivi skup varijabli X, Y, Z, . . .

3. Skup pomoćnih simbola "(" i ")", tj. zagrade

2.1.2 Jezik

Jezik propozicijske logike sa zadanom abecedom je skup riječi koje zovemo formule.

Najjednostavnija formula je atom ili atomarna formula. Pa je tako svaka varijabla atomarna formula. Atome ili negirane atome nazivamo literalima.

- Ako je "A" formula, tada je i "¬A" formula.
- Ako su "A" i "B" formule, tada su i "(A ∧ B)", "(A ∨ B)" i "(A → B)" također formule, te ostale kombinacije veznika s ove dvije propozicijske varijable.

Pri zapisivanju običaj je ispuštati zagrade, pa tako umjesto "((A ∨ B) ∨ C)" možemo pisati "A ∨ B ∨ C". Pri ispuštanju zagrada postavljen je prioritet veznika ovim redosljedom : ¬, ∨, ∧, →.

Postoje i potformule:

- Ako je "A" potformula od "C", tada je i "¬A" potformula od "C".
- Ako je "A ∧ B" ili "A ∨ B" ili "A→B" ili "A↔B" potformula od "F", tada su "A" i "B" potformule od F.

Primjer formule : " (A ∨ B) ∧ C"

Sve njezine potformule su : " (A ∨ B) ∧ C", "A", "B", "A ∨ B", "C".

2.2 Sustav prirodne dedukcije

Sustav prirodne dedukcije najpoznatiji je tip logike korišten u sadašnjoj filozofiji. Prirodnu dedukciju kakva danas postoji predložio je Gerhard Karl Erich Gentzen uz izjavu : " Želio sam konstruirati formalizam koji će biti što je bliže moguće stvarnom rasuđivanju. Pa je tako nastao pojam 'račun prirodne dedukcije' ". Sustav prirodne dedukcije Gentzen je razvio 1935. godine. Osnovu ovog sustava čine pravila koja se javljaju u paru, za svaki veznik po jedan par.
Jedno pravilo iz para govori kako uvesti veznik u dokazivanje, a drugo kako eliminirati veznik iz dokazivanja. Ta pravila su elementarne prirode i vezana su za smisao veznika, pa ih se ne može rastaviti na jednostavnije.

Dokazivanje je proces kojim se krenuvši od pretpostavki i koristeći niz jednostavnih pravila zaključivanja dolazi do novih tvrdnji. Pravila koja se koriste moraju biti smislena, pa tako ako pravilo primjenjeno na skup tvrdnji "P" daje tvrdnju "x", tada iz "P" slijedi "x". To jest ako iz nekog skupa pretpostavki "P" proizlazi tvrdnja "x", onda je ona logička posljedica toga skupa pretpostavki. [1]

Dokazivanje označavamo sa : $P \vdash x \rightarrow P \models x$.

Cilj sustava dokazivanja je imati dovoljno pravila da se može dokazati iz danih pretpostavki svaku tvrdnju koja iz njih logički slijedi. Jedino takav sustav dokazivanja je potpun:

Potpun sustav dokazivanja : $P \models x \rightarrow P \vdash x$.

U sustavu prirodne dedukcije svaki dokaz započinje s popisivanjem pretpostavki koje se koriste u dokazivanju. Taj početni niz pretpostavki "$a_1, a_2 \ldots a_n$" se zove bazni dokaz. Kako bi se znalo da su to pretpostavke u tekstu koristi se oznaka (P) u zagradi.
Zapis baznog dokaza : a_1 (P)

a_2 (P)

⋮

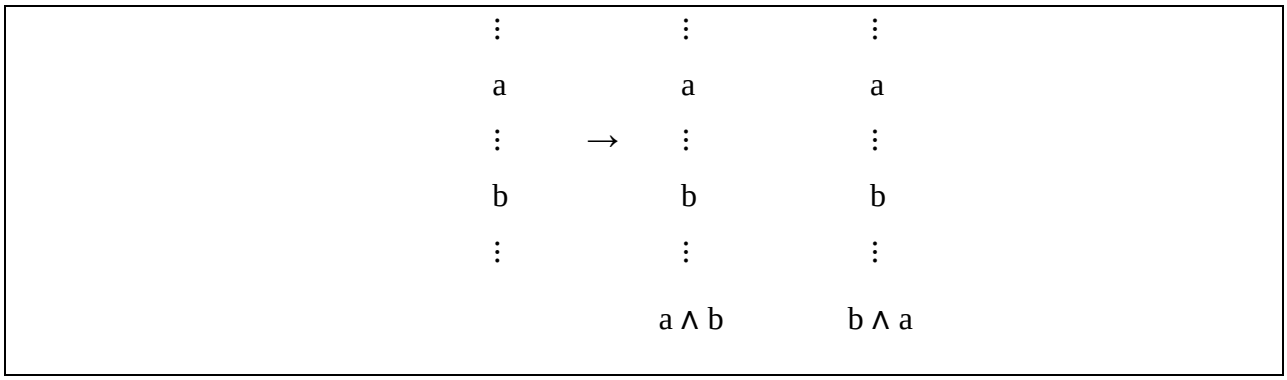
a_n (P)

Svaki dokaz u neformalnom dokazivanju je dokaz zadnje tvrdnje u nizu: $a_1, a_2 \ldots a_n \models a_n$

2.3 Pravila prirodne dedukcije

2.3.1 Pravilo uvođenja konjukcije (Intro ∧)

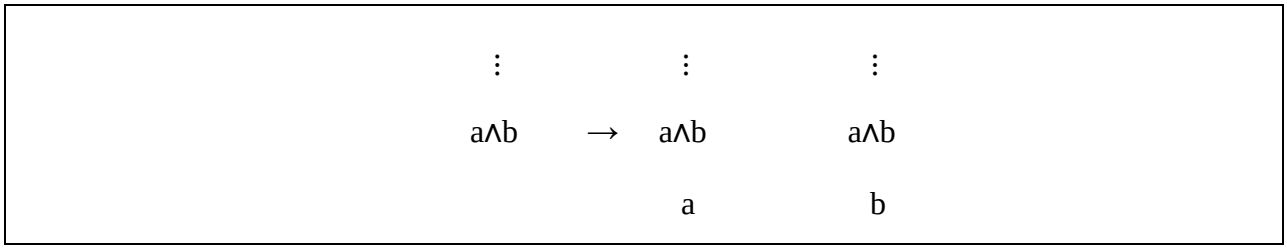
Ako je dokazana tvrdnja "a" i ako je dokazana tvrdnja "b", time je dokazana i tvrdnja "a ∧ b" i "b ∧ a".



Početne tvrdnje su "a" i "b", dok desno od strelice dodavanjem konjukcije proizlazi zaključak "a ∧ b", odnosno "b ∧ a".

2.3.2 Pravilo eliminiranja konjukcije (Elim ∧) :

Ako je dokazana tvrdnja "a ∧ b", dokazana je i tvrdnja "a" i tvrdnja "b".



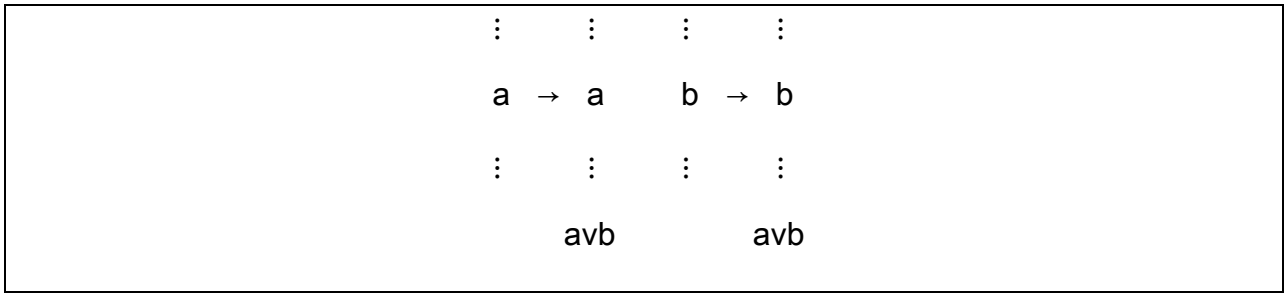
Početna tvrdnja je "a ∧ b". Eliminiranjem konjukcije početne tvrdnje dobije se zaključak "a", odnosno zaključak "b". Primjer 2.3.2. : Dokaz za (A ∧ B) ∧ C ⊢ A ∧ (B ∧ C)

1. (A ∧ B) ∧ C	(P)
2. C	(1 Elim ∧)
3. (A ∧ B)	(1 Elim ∧)
4. A	(3 Elim ∧)
5. B	(3 Elim ∧)
6. C ∧ B	(2,4 Intro ∧)
7. A ∧ (B ∧ C)	(4,6 Intro ∧)

U ovom primjeru, svakoj tvrdnji pridružen je redni broj i pravilo iz kojeg je izvedena ili u slučaju da je pretpostavka slovo P. Sam primjer pokazuje na koji način se provodi prirodna dedukcija. Pravilima eliminacije rastavljaju se tvrdnje na osnovne dijelove, pa od tih dijelova pomoću pravila uvođenja dobiva se zaključak.

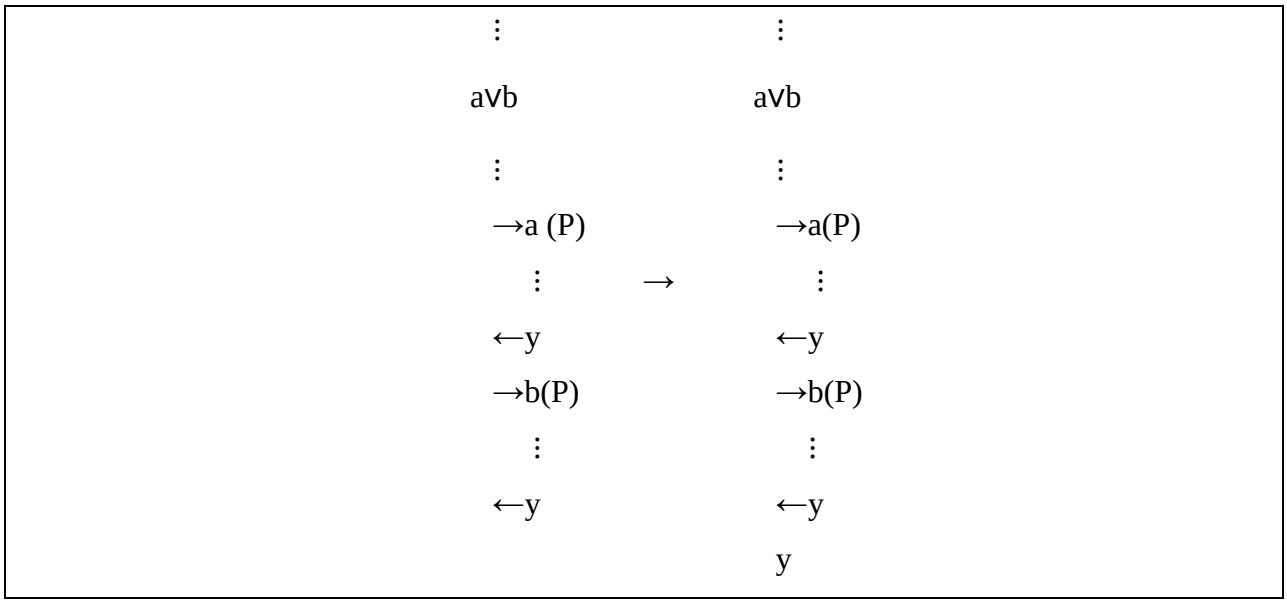
2.3.3 Pravilo uvođenja disjunkcije (Intro ∨)

Za dokaz disjunkcije "a ∨ b" potrebno je dokazati "a" ili "b".



U slučaju istinite tvrdnje "a" proizlazi "a ∨ b", te u slučaju istinite tvrdnje "b" proizlazi "a ∨ b".

2.3.4 Pravilo eliminacije disjunkcije (Elim ∨)



U slučaju istinite tvrdnje "a v b" moguća su tri scenarija. Prvi scenarij je taj da je tvrdnja "a" istinita, drugi da je tvrdnja "b". Uz pretpostavku da je a istina, dokazan je y. Isto tako uz pretpostavku da je b istina, dokazan je y. Dakle u slučaju da je "a v b" dokazano, dokazan je i y. Dio dokaza koji koristi privremenu pretpostavku zove se poddokaz. On je postavljen u desno od glavnog dokaza. Strelicom desno (→) označen je početak poddokaza, a strelicom (←) lijevo se završava i vraća na liniju dokazivanja. Takvi prijelazi na poddokaze mogu se dogoditi i unutar samog poddokaza. [2]

Primjer 2.3.4. : Dokaz za $A \wedge (B \vee C) \vdash (A \wedge B) \vee (A \wedge C)$.

1.	$A \wedge (B \vee C)$	(P)
2.	A	(1 Elim $\wedge$)
3.	$B \vee C$	(1 Elim $\wedge$)
4.	$\rightarrow B$	(P)
5.	$A \wedge B$	(2,4 Intro $\wedge$)
6.	$\leftarrow (A \wedge B) \vee (A \wedge C)$	(5 Intro $\vee$)
7.	$\rightarrow C$	(P)
8.	$A \wedge C$	(2,7 Intro $\wedge$)
9.	$\leftarrow (A \wedge B) \vee (A \wedge C)$	(8 Intro $\vee$)
10.	$(A \wedge B) \vee (A \wedge C)$	(4-6,7-9 Elim$\vee$)

2.3.5 Pravilo uvođenja kondicionala (Intro →)

⋮		⋮
→ a(P)		→ a(P)
⋮	→	⋮
←b		←b
		a→b

Za a se uzima pretpostavka, te se iz pretpostavke metodom direktnog dokaza dokazuje b.

2.3.6 Pravilo eliminiranja kondicionala (Elim →)

Pravilo modus ponens: iz a i a→a donosi se zaključak b.

⋮		⋮
a		a
⋮	→	⋮
a→b		a→b
⋮		⋮
		b

Primjer 2.3.6: Dokaz za $A \rightarrow (B \rightarrow C) \vdash (A \wedge B) \rightarrow C$

U ovom slučaju uzima se pretpostavka "A ∧ B".

1.	A→(B→C)	(P)
2.	→ A∧B	(P)
3.	A	(2 Elim ∧)
4.	B	(2 Elim ∧)
5.	B →C	(1,3 Elim →)
6.	← C	(3,5 Elim →)
7.	(A∧B) → C	(2-6 Intro →)

Prvi korak je pretpostavka za neformalni dokaz. Drugi korak postavljamo pretpostavku u poddokaz. Treći i četvrti korak je eliminacija konjukcije. dok je peti korak eliminacija kondicionala iz prve pretpostavke. Šesti korak je vraćanje zaključka poddokaza. Sedmi korak je dokaz dobiven uvođenjem kondicionala na pretpostavku iz drugog koraka i zaključak poddokaza a∧b → c.

2.3.7 Pravilo uvođenja bikondicionala (Intro ↔)

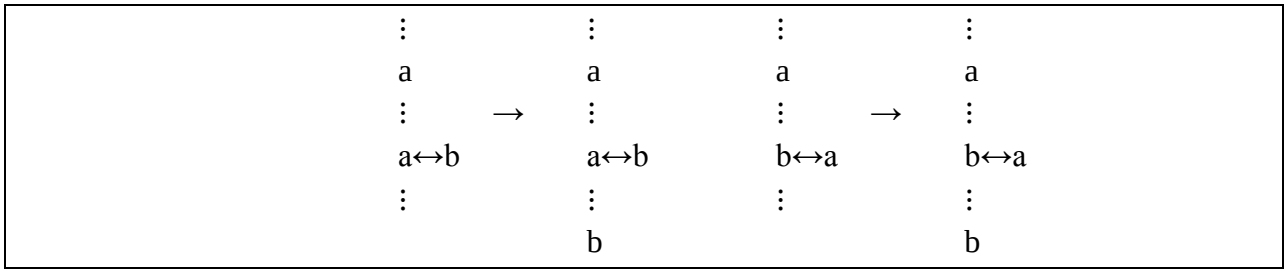
Bikondicional je definiran pomoću kondicionala i konjukcije kroz formulu :

$a \leftrightarrow b \equiv (a \rightarrow b) \wedge (b \rightarrow a)$

⋮		⋮
→a(P)		→a(P)
⋮	→	⋮
←b		←b
→b(P)		→b(P)
⋮		⋮
←a		←a
		a↔b

Odnosno iz pretpostavke A proizlazi B i iz pretpostavke B proizlazi A.

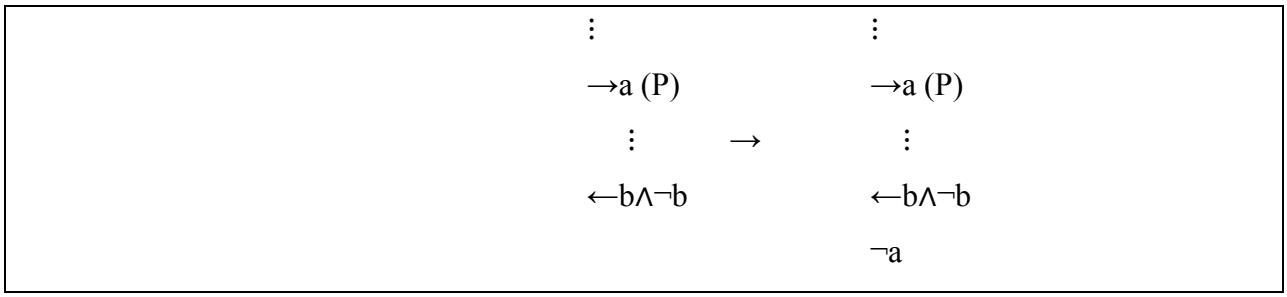
2.3.8 Pravilo eliminiranja bikondicionala (Elim ↔)



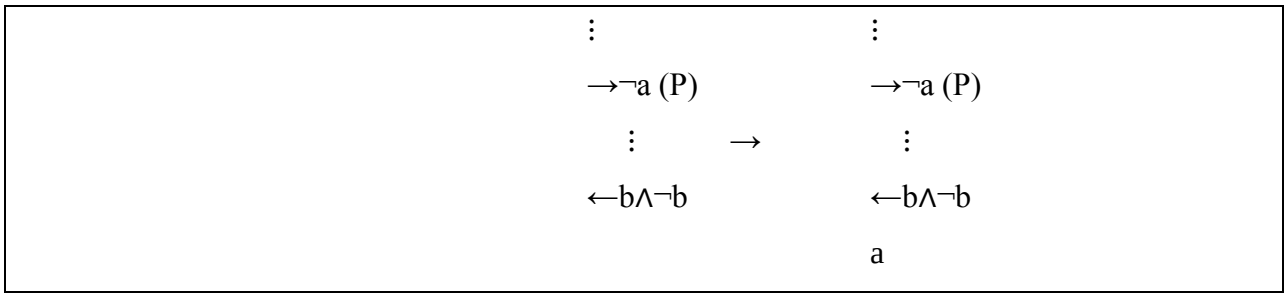
Iz tvrdnje a i tvrdnje a↔b dokazan je b. Isto tako iz tvrdnje a i b↔a, dokazan je b.

2.3.9 Pravilo uvođenja negacije (Intro ¬)

Ovo pravilo nastalo je kroz dokaz kontradikcijom. Privremenom pretpostavkom da vrijedi a, dobije se kontradikcija b∧¬b. To znači da je pretpostavka nevažeća, pa je zaključak ¬a :



2.3.10 Pravilo eliminiranja negacije (Elim ¬)



Pravilo eliminiranja negacije počinje se sa privremenom pretpostavkom da je ¬a, te se donosi poddokaz b∧¬b. Ova kontradikcija navodi na zaključak a. Kod oba pravila negacije, negacija se koristi i kao negacija koju uvodimo ili eliminiramo i kao negacija u kontradikciji poddokaza.

Primjer 2.3.10: Dokaz za A preko obaranja negacije od A.

1.	¬(¬A)	(P)
2.	→¬A	(P)
3.	← ¬A ∧ ¬(¬A)	(1,2 Intro ∧)
4.	A	(2,3 Elim ¬)

Za prepostavku se u drugom koraku uzima ¬A. Pošto je potrebno oboriti A kontradikcijom. Upravo to je treći korak, gdje se sa uvođenjem ∧ konjukcije dobiva kontradikcija prvog i drugog koraka. Što dovodi do zaključka A.

3. ANDROID SUSTAV

3.1 Povijest Android operacijskog sustava

Google Android je otvoreni operacijski sustav za mobilne uređaje temeljen na jezgri Linux operativnog sustava. Android je prilagodljiv pa se koristi i za razne uređaje poput čitača elektronskih knjiga, multimedijskih uređaja i raznoraznih prijenosnika. Tvrtku Android su osnovali Andy Rubin, Rich Miner, Nick Sears i Chris White 2003. godine. Njihovu tvrtku kupio je Google 2005 godine. No tek je 2007. godine Android bio predstavljen od strane Googla kao mobilna platforma na T-Mobile G1 uređaju. Iste godine iPhone je radio svoju vlastitu revoluciju kako bi postao najdominantniji na tržištu mobitela, pa se Google odlučio na postavljanje Android OS-a besplatno na tržište. Google je nastavio zarađivati kroz prodaju aplikacija preko Android tržišta, a pošto je Android OS sada bio besplatan tržište se naglo raširilo. Svi proizvođači tražeći za jeftinim, ali modularnim sustavom okrenuli su se prema Googleu i njihovim Android sustavom. Danas je Android najrašireniji operacijski sustav na pametnim uređajima, a u rujnu 2015. Android je imao 1.4 milijarde aktivnih korisnika.

3.1.1 Verzije Android operativnog sustava

Google je kreirao šest glavnih Android verzija sa većim brojem naknadnih ažuriranja. Android 1.X verzija puštena je 2008. godine i ova verzija je već sadržavala više mogućnosti nego ostali konkurenti na tržištu. No sadržavala je greške i nije bila toliko prilagodljiva korisniku. No sve se to popravilo u verziji 2.X. S ovom verzijom Android je postao prijateljsko sučelje korisniku, a zadržao je velik broj korisnih mogućnosti iz prethodne verzije. Tada je naišao novi izazov iz tvrtke Apple, kreirali su iPad koji je bio ispred svoje konkurencije tableta iz Googlea. Glavni razlog je bio što su aplikacije za iPad bile specifično dizajnirane za njegovu veličinu, dok su aplikacije za razne tablete u Android-u bile dizajnirane za sve uređaje bez obzira na veličinu ekrana. To je kod tableta s Android sustavom dovelo do različitih izgleda aplikacije na različitim ekranima. S verzijom 3.x Google je pokušao donijeti optimizirano sučelje za tablet. No sa novim sučeljem bilo je potrebno optimizirati i aplikacije, pa taj pokušaj nije urodio plodom. Tek verzija 4.0 sa imenom "Ice Cream Sandwich" je uspjela smanjiti razliku između tableta i mobitela. Ova verzija donijela je brzinu, izgled i mnoge mogućnosti, te je kroz daljnje 4.x verzije Google pogurao programere da se bave aplikacijama optimalnim za tablete. Verzija 5.x donijela je najveću promijenu u izgledu sučelja na uređajima, dok je 6.x verzija nastavila u istom smjeru te ubrzala rad na uređajima. Google je razvio strategiju u kojoj svaka verzija olakšava korisnicima rad na njihovim uređajima i daje im nove mogućnosti.

3.2 Android Studio

Android uređaji su relativno dostupni danas, te je programiranje u Android-u osmišljeno na relativno jednostavan način. Ono što čini Android posebnim je to što je zamišljen kao projekt otvorenog koda, te ga prema tome može koristiti tko želi i primjenjivati prema svojim željama. Android studio korišten je u izradi sučelja za prirodnu dedukciju u ovom radu i dostupan je kao i većina toga vezanog za Android besplatno. Android Studio je IDE (Integrirana razvojna okolina) za razvoj Android aplikacija.

Mogućnosti Android Studija se kreću od uređivača teksta i različitih razvojnih alata, pa do emulatora za virtualne uređaje. Najpoznatije mogućnosti su uređivač izgleda projekta sa opcijom povuci i stavi, ugrađena potpora za Google Cloud platformu i sustav Gradle,. Gradle je build sustav, odnosno sustav koji služi kako bi izvorne programske datoteke npr. ".java" datoteke pretvorio i komprimirao u ".apk" datoteku s kojom Android sustav može raditi. [3] Android projekt se sastoji od modula sa izvornim programskim datotekama i datotekama raznih resursa poput slika. Svi podaci potrebni za build aplikacije nalaze se u manifests, java i res datotekama. U manifestu se nalaze razne dozvole koje aplikacija traži kao pravo od korisnika, dok se u java datoteci nalaze sve Java izvorne programske datoteke. A u res datoteci se nalaze svi ostali resursi koji nisu programski kod.

3.3 Java

Java je objektno orjentirani programski jezik (OOP) koji je plasiran na tržište 1995. godine od tvrtke Sun Microsystems. Koristi se u raznoraznim internet preglednicima i kao dio različitih platformi, jedna od njih je Android operacijski sustav. Java se koristi unutar Android Studija, a omogućuje programeru upravljanje podacima, prihvaćanje različitih odgovora od krajnjeg korisnika, prikazivanje na ekran onoga što programer želi i mnoge druge mogućnosti. Java je pisana statički, odnosno Java zahtjeva da tip svake varijable bude deklariran. Pri pokretanju programa Java će provjeriti jesu li sve varijable deklarirane i u suprotnom će izbaciti pogrešku.

3.3.1 Osnove Java jezika

Osnovni i najčešći tipovi podataka :

- int (integer) - cijeli broj uključuje i nulu i negativne brojeve,
- float - aproksimacija stvarnog broja, uključuje decimalne brojeve
- char - jedan znak, npr "X"
- boolean - ima vrijednost "true" (istina) ili "false" (laž), predstavljaju 1 i 0
- String - određeni broj znakova zajedno koji čine tekst, npr:

```
String x ="Ovo je tekst tipa String";
```

- String se u Javi piše velikim slovom jer je kompleksniji tip podataka i ujedno i objekt. Objekte u Javi se piše velikim početnim slovom. Objekt tipa String ima mogućnost poziva svojih metoda, npr. x.toLowerCase(); koja vraća varijablu x tipa String sa svim velikim slovima prebačenim u mala. [4]

Metoda je dio koda, kojeg se može pozvati iz drugih dijelova koda kako bi obavila svoju zadaću. Koristi se kako bi kod bio organiziraniji i čitkiji. Primjer metode :

```
public int primjerMetode (int a) {
int b = a+1;
return b;
}
```

Prva riječ govori o vidljivosti metode. Najčešće je to public, private ili protected, no metoda može postojati i bez ove riječi. Public znači da je metoda vidljiva u cijelom kodu, private znači da je metoda vidljiva samo klasi kojoj pripada, dok protected znači da je vidljiva u paketu u kojem se

nalazi i pod-klasama paketa. Druga riječ je povratni tip metode, int metoda mora vratiti int vrijednost. U slučaju da nije potrebno vratiti vrijednost, tip metode se postavlja na void. Unutar oblih zagrada postavlja se lista parametara koje metoda prima, dok unutar vitičastih zagrada dolazi tijelo metode. Kako bi se vratila vrijednost metode potrebna je ključna riječ *return.* A kako bi se pozvala metoda negdje u kodu, potrebno je napisati ime metode i pridodati joj njene parametre. Primjer pozivanja metode:

```
...
int c;
c = primjerMetode (7);  // poziv metode
...
```

Ovdje se poziva metoda imena *primjerMetode* sa paramterom vrijednosti 7, ona vraća vrijednost 8 i tu vrijednost poprima varijabla c.

Komentari se u Javi pišu na dva načina, jedan od načina napisan je u primjeru pozivanja metode. Primjer pisanja komentara u Javi:

```
String x = "Ovo je tekst tipa String";  // Komentar od jedne linije
/* komentar u više linija
   dok se ne zatvori */
```

Sam komentar nema nikakvog učinka na kako će program raditi. Komentari se koriste kako bi olakšali razumijevanje koda.

4. ANDROID APLIKACIJA

Aplikacija je zamišljenja kao sučelje koje omogućava kreiranje dokaza kroz pravila prirodne dedukcije. Korisnik nakon pokretanja aplikacije ima na uvid polje za upis formule koju želi testirati. Nakon što korisnik dodirne polje za upis teksta formule izlazi mu prilagođena "tipkovnica" sa simbolima iz logike. Korisnik kako bi nastavio sa kreiranjem dokaza mora upisati pravilno formulu, te kliknuti na dugme "DONE". Ukoliko korisnik nije napisao ispravno formulu, na ekran mu se ispisuje prigodna poruka sa naznakom što je pogrešno u formuli. Nakon ispravnog unosa formule, korisniku se otvara prozor za kreiranje dokaza. U ovom prozoru nalaze se dvije liste. Prva lista je lista dokaza i u njoj se nalaze korisnikove pretpostavke, kao i drugi dijelovi dokaza koje korisnik kreira. Druga lista je lista prijedloga za dokaz. Pa tako druga lista sadrži na prvom mjestu stavku za kreiranje nove pretpostavke, dok se ostale stavke pune u odnosu na to što je odabrano u prvoj listi. Svakom odabranom stavkom u drugoj listi, korisnik puni prvu listu sa dijelovima dokaza i/ili pretpostavkom. Nakon svakog novog dijela dokaza, program uspoređuje zadnju stavku sa zaključkom iz prvotne formule. U slučaju da program ustvrdi da je dokaz izvršen, otvara se prozor u kojem se ispisuje cijeli put do dokaza. Nakon toga korisnik klikom na dugme "NOVI DOKAZ" može nastaviti sa kreiranjem dokaza.

4.1 Priprema razvojnog okruženja

Za razvoj ove aplikacije instaliran je AnroidStudio. Uz Android studio, instalirani su i Android SDK alati. Za testiranje aplikacije korišten je Genymotion program za simulaciju mobilnih uređaja, te usb spajanje sa fizičkim mobilnim uređajem.

4.2 Aktivnosti aplikacije

- Početna aktivnost aplikacije nazvana je HelloActivity i postavljena je na pogled activity_hello.xml. Ovaj xml pogled sadrži gumb za pokretanje sljedeće aktivnosti i jednu liniju teksta.
- Aktivnost SetFormulaActivity sadrži polje za unos teksta formule i "slidingPanel" odnosno klizeći panel na kojem je smještena improvizirana tipkovnica.
- Aktivnost MainActivity sastoji se od linije teksta za prijepis postavljene formule od korisnika, dvije liste i teksta iznad druge liste koji pojašnjava drugu listu.

- Aktivnost EndActivity je završna aktivnost za dokaz, sadrži jednu listu koju je nasljedila iz MainActivity aktivnosti. Ova lista sadrži cijeli put do dokaza. Osim liste ova aktivnost sadrži i gumb za kreiranje novog dokaza.

4.2.1 HelloActivity

Kroz metodu onCreate aktivnost se inicijalizira, dok se naredbom setContentView() postavlja pogled na xml datoteku activity_hello.xml. Metoda startNext(View v) je metoda za pokretanje iduće aktivnosti, a pokreće se na klik gumba iz activity_hello.xml datoteke.

```
public class HelloActivity extends AppCompatActivity {


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_hello);
  }
  public void startNext(View v ) {

    Intent i = new Intent(this, SetFormulaActivity.class);


    startActivity(i);
  }
}
```

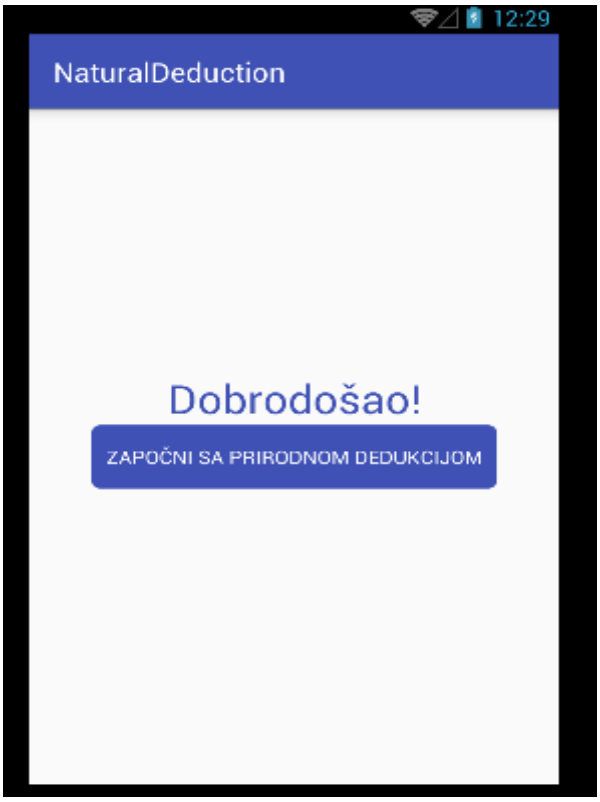
Kako bi se dizajniralo korisničko sučelje najčešće se koristi XML kod, iako je moguće i kroz pisanje Java koda. U xml datoteci moguće je definirati različite vrste rasporeda od kojih su najpoznatiji relacijski raspored (RelativeLayout) i linearni raspored (LinearLayout). Relacijski raspored omogućuje nam postavljanje jednog elementa u odnosu na poziciju drugog elementa. Dok linearni raspored zahtjeva raspored elementa kroz jedan red ili jedan stupac.

XML datoteka koja definira grafičko sučelje HelloActivity aktivnosti koristi RelativeLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_hello"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="com.one.ico.prirodnadedukcija.HelloActivity">

  <TextView
      android:id="@+id/Hello"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:textColor="@color/colorPrimary"
      android:text="@string/welcome"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="183dp"/>

  <Button android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/zapo_ni_sa_prirodnom_dedukcijom"
      android:layout_below="@id/Hello"
      android:layout_centerHorizontal="true"
      android:background="@drawable/button_selector"
      android:textColor="@drawable/button_text_color"
      android:onClick="startNext"/>
</RelativeLayout>
```

Kod android:layout_width="" i android:layout_height="" definiraju širinu i visinu pojedinačnog elementa, kao i cijelog pogleda. TextView je pogled koji sadrži tekst, u ovom slučaju to je pogled koji sadrži tekst id-a Hello. Osim teksta ovaj pogled sadrži i gumb koji kroz kod android:onClick="startNext" govori kako se na dodir ovog dugma pokreće metoda startNext(View view). Ovdje je "view" pogled na dugme koje je stisnuto.



Slika 1 Početni zaslon, HelloActivity

4.2.2 SetFormulaActivity

Ova aktivnost sadrži kod za provjeru korisničkog unosa, te intent za pokretanje nove aplikacije. XML postavljen na ovu aktivnost je set_formula.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <com.sothree.slidinguppanel.SlidingUpPanelLayout
        xmlns:sothree="http://schemas.android.com/apk/res-auto"
    android:id="@+id/sliding_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="bottom"
    sothree:umanoDragView="@+id/dragView"
    sothree:umanoInitialState="hidden">
    <!-- glavni dio layouta-->
    <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:weightSum="1">
```

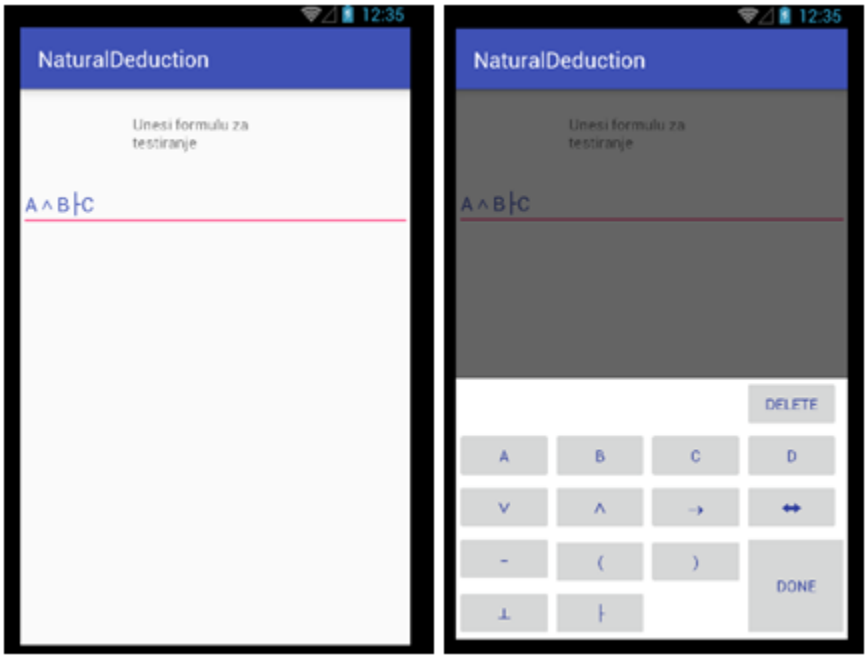
```
<View
        android:id="@+id/wholelayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        ></View>

    <EditText
        android:id="@+id/input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textViewProof"
        android:layout_alignStart="@+id/textViewProof"
        android:layout_below="@+id/textViewFormula"
        android:layout_marginTop="25dp"
        android:inputType="textNoSuggestions"
        android:maxLength="23"
        android:textColor="@color/colorPrimary"
        tools:layout_margin="25dp" />

    <TextView
        android:id="@+id/textViewFormula"
        android:layout_width="153dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="23dp"
        android:text="@string/unesi_formulu_za_testiranje"
        tools:layout_margin="25dp" />


</RelativeLayout>

<!--Sliding dio-->
<LinearLayout
    android:id="@+id/dragView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:clickable="true"
    android:focusable="false"
    android:orientation="vertical">
```

```
<GridLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:columnCount="4"
        android:orientation="horizontal">
        <Button
          android:id="@+id/delete"
          android:layout_column="3"
          android:onClick="keyOnClick"
          android:text="Delete"
          android:textColor="@color/colorPrimaryDark" />
        <Button
          android:id="@+id/A"
          android:onClick="keyOnClick"
          android:text="A"
          android:textColor="@color/colorPrimaryDark" />
      .
      .
      .
        <Button
          android:id="@+id/dokaz"
          android:layout_columnSpan="1"
          android:layout_gravity="fill"
          android:onClick="keyOnClick"
          android:text="□"
          android:textColor="@color/colorPrimaryDark" />
      </GridLayout>
    </LinearLayout>
  </com.sothree.slidinguppanel.SlidingUpPanelLayout>
</RelativeLayout>
```

Ovaj xml koristi relativni raspored elementa, te se sastoji od klizećeg panela na kojem se nalazi GridLayout odnosno mrežasti raspored i RelativeLayout. Na RelativeLayout-u se nalazi EditText id-a "input" za unos teksta i TextView sa tekstom "Unesi formulu za testiranje". Dok se GridLayout sastoji od gumbova za improviziranu tipkovnicu. Svaki gumb na ovoj tipkovnici aktivira se pozivom na funkciju keyOnClick.

U onCreate metodi aktivnosti SetFormulaActivity postavlja se pogled na EditText input i na dugme delete. Osim toga postavlja se i pogled na klizeći panel pod nazivom slidingLayout. Na pogled id-a wholeLayout postavlja se slušač, kako bi se na mjestima di se ne nalazi klizeći panel, isti na dodir spustio u slučaju da je podignut. Postavljen je i slušač na dodir za unos teksta na input.

```
public class SetFormulaActivity extends AppCompatActivity {

  private static EditText input;
  private SlidingUpPanelLayout slidingLayout;
  private TextView dokazTextV;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.set_formula);

    input = (EditText) findViewById(R.id.input);

    dokazTextV = (TextView) findViewById(R.id.textViewProof);

    Button delete = (Button) findViewById(R.id.delete);
    slidingLayout = (SlidingUpPanelLayout)   findViewById(R.id.sliding_layout);

    input.setOnTouchListener(touchListener);
    View layout = findViewById(R.id.wholelayout);
    //listener za slide "tipkovnice"
    //klikom bilo gdje na layout spusti sliding panel dole

    layout.setOnTouchListener(new View.OnTouchListener() {
      @Override
      public boolean onTouch(View view, MotionEvent motionEvent) {

        if (slidingLayout.getPanelState() != SlidingUpPanelLayout.PanelState.COLLAPSED) {
          slidingLayout.setPanelState(SlidingUpPanelLayout.PanelState.COLLAPSED);
          return true;
        }

        return false;
      }

    });
```



Slika 2 Slika SetFormulaActivity, spušten i podignut panel s tipkovnicom

Iz razloga što je regularna tipkovnica za android uređaje nepotrebna za ovu aplikaciju, sa ovim slušačem na dodir (touchListener) postavljeno je da se umjesto tipkovnice pojavi improvizirana tipkovnica na dodir EditText-a input.

```
private View.OnTouchListener touchListener = new View.OnTouchListener() {
    @Override

    public boolean onTouch(View view, MotionEvent motionEvent) {
      slidingLayout.setPanelState(SlidingUpPanelLayout.PanelState.EXPANDED);
          .
          .
          .
          .
      return true;

    }
  };
```

Za početak pisanja formuleu polje za upis, potrebno je pritisnuti gumb sa klizećeg panela. To je omogućeno sa metodom keyOnClick(view v), ovdje svaki gumb koji je pritisnut predaje svoj pogled ovoj metodi kako bi se moglo odrediti koji je gumb kliknut i odraditi ispravna akcija. Svi gumbi osim gumba done i delete pišu u EditText svoju vrijednost.

Upis tipkovnice i metoda keyOnClick(v):

```
//tipkovnica upis
  public void keyOnClick(View v) {

    switch (v.getId()) {
      case R.id.A:
        input.getText().insert(input.getSelectionStart(), "A");
        break;
      case R.id.B:
        input.getText().insert(input.getSelectionStart(), "B");
        break;
                    .
                    .
                    .
                    .
      case R.id.dokaz:
        input.getText().insert(input.getSelectionStart(), "⊢");
        break;
      case R.id.kontra:
        input.getText().insert(input.getSelectionStart(), "⊥");
        break;
      case R.id.done:
        ....
```

Gumb delete postavljen je u onCreate() metodi na listener za obični dodir i dugi dodir.

Kroz ovu metodu delete gumb omogućava brisanje prvog mjesta u lijevo od mjesta gdje je dodirnut EditText input:

```
/brisanje zadnjeg inputa
    delete.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        int length = input.getSelectionStart();
        if (length > 0) {
          input.getText().delete(length - 1, length);
        }
      }
    });
```

Kroz ovu metodu delete gumb omogućava brisanje cijelog dosadašnjeg unešenog teksta:

```
//brisanje svega na dug klik
    delete.setOnLongClickListener(new View.OnLongClickListener() {
      @Override
      public boolean onLongClick(View v) {
        input.setText("");
        return true;
      }
    });
  }
```

Pritiskom tipke "DONE" poziva se metoda inputCheck() za provjeru unosa formule. Ovoj metodi se predaje kontekst aktivnosti iz koje je pozvana kako bi se mogla pozvati i iz drugih aktivnosti, te tekst na koji se odnosi. Nakon što je provjera završila, u slučaju da je formula ispravna poziva se metoda parse() za pripremu i predaju podataka idućoj aktivnosti.

```
    case R.id.done:
          Context context = getApplicationContext();
          String text = String.valueOf(input.getText());
          if (inputCheck(context, text, 0)) {  parse();
          }
break;
. . .
```

inputCheck metoda za provjeru unosa :

```
static Boolean inputCheck(Context context, String text, int i) {
    if (text.isEmpty()) {
      Toast.makeText(context, "Molimo unesite formulu :)", Toast.LENGTH_SHORT).show();
      return false;
    }
    String asubstring = text.substring(text.length() - 1);  //zadnji char
    String startsubstring = text.substring(0, 1);  //prvi char

    int count = text.length() - text.replaceAll("⊢", "").length();
    int countZagrade1 = text.length() - text.replaceAll("\\(", "").length();
    int countZagrade2 = text.length() - text.replaceAll("\\)", "").length();

    if (i == 0 && !(count == 1)) {
      Toast.makeText(context, "U formuli može/treba biti jedan  ⊢", Toast.LENGTH_SHORT).show();
      return false;
}
```

```
if (countZagrade1 != countZagrade2) {
        Toast.makeText(context, "Zagrada nije zatvorena, molim zatvorite sve zagrade", Toast.LENGTH_SHORT).show();
        return false;

    }
        if (text.contains("AB") ... ("DD")    {
                .
                .
                .
                .
else if (text.contains("()") || text.contains("(A)") || text.contains("(B)") || text.contains("(C)") || text.contains("(D)")) {
        Toast.makeText(context, "Imate praznu zagradu",Toast.LENGTH_SHORT).show();
        return false;
    }

    if (!(asubstring.contains("A") || asubstring.contains("B") || asubstring.contains("C") || asubstring.contains("D") ||
asubstring.contains(")") || asubstring.contains("⊥"))) {
        Toast.makeText(context, "Postoji greška u formuli, vaša formula završava sa veznikom",
Toast.LENGTH_SHORT).show();

        return false;
    }

    if (!(startsubstring.contains("A") || startsubstring.contains("⊥") || startsubstring.contains("B") ||
startsubstring.contains("C") || startsubstring.contains("D") || startsubstring.contains("(") || startsubstring.contains("-
"))) {
        Toast.makeText(context, "Postoji greška u formuli, vaša formula počinje sa veznikom",
Toast.LENGTH_SHORT).show();
        return false;
    }
    return true;
  }
```

Ova metoda provjerava sve mogućnosti krivog unosa formule, kako bi kasnije bilo moguće kreirati ispravne dokaze.

Kako bi formula bila ispravna potrebno je :

- zatvoriti sve zagrade
- veznicima odvojiti znakove koje ćemo koristiti
- odvojiti formula na dva dijela premisu i zaključak sa " ⊢"
- završiti i početi formulu sa znakom koji nije veznik
- minuse postavljati samo ispred znakova koji to dopuštaju npr. A,B

Poziv na MainActivity kroz Intent iz SetFormulaActivity-a :

```
private void parse() {
    String formula = String.valueOf(input.getText());
    String[] parts = formula.split("⊢", 2);
    String premisa = parts[0];
    Intent i = new Intent(SetFormulaActivity.this, MainActivity.class);
    i.putExtra("FORMULA", formula);
    i.putExtra("PREMISA", premisa);
    startActivity(i);
  }
```

Prije nego što se MainActivity pozove, formula se prepolovi na dva dijela na mjestu gdje se nalazi znak "⊢". Te se tada prvi dio, odnosno premisa i formula postavljaju u Intent i poziva se metoda startActivity(intent);

4.2.3 MainActivity

MainActivity postavlja svoj pogled na xml datoteku main_activity, te pozovi metodu getIntent() kako bi prihvatila vrijednosti poslane iz SetFormulaActivity-a.

```
public class MainActivity extends AppCompatActivity {
        ....
  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
    context = this;
    Intent i = getIntent();
    String formula = i.getStringExtra("FORMULA");  String premisa = i.getStringExtra("PREMISA");  ....
```

main_activity.xml datoteka korištena za postavljanje elemenata za MainActivity:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
```

```
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="start"
      android:text=" Formula za dokazati : "
      android:textColor="@color/colorPrimary"
      android:textSize="15sp" />
    <TextView
      android:id="@+id/textFormule"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
     android:layout_gravity="center"
      android:layout_marginBottom="30dp"
      android:text=" Formula za dokazati : "
      android:textColor="@color/colorPrimary"
      android:textSize="15sp"
      />
    <ListView
      android:id="@+id/list_premisa"
      android:layout_width="match_parent"
      android:layout_height="203dp"
      android:choiceMode="multipleChoice"
      android:listSelector="@drawable/list_selector">
    </ListView>
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="start"
      android:text=" Prijedlozi : "
      android:textColor="@color/colorPrimary"
      android:textSize="15sp" />
    <ListView
      android:id="@+id/list_pretpostavke"
      android:layout_width="match_parent"
      android:layout_height="246dp"
      android:choiceMode="singleChoice">
    </ListView>
</LinearLayout>
```

Ova xml datoteka sastoji se od dva ugniježđena LinearLayout-a u glavnom LinearLayout-u. Oba unutrašnja LinearLayout-a sadrže po jedan listView, odnosno pogled na listu1 i listu2. Id prve liste je "list_dokaza", dok je id liste dva "list_pretpostavke". Prva lista postavljena je na mogućnost višestrukog izbora android:choiceMode="multipleChoice", dok je druga postavljena na

mogućnost izbora samo jednog elementa na listi odjednom singleChoice. U prvom LinearLayout-u TextView je postavljen na id "text_formule".

Kako bi se pogledi na liste iz xml-a mogli koristiti, potrebno ih je postaviti u Java kodu. Pa je listView1 postavljen na element list_dokaza, dok je listView2 postavljen na element list_pretpostavke.

```
listView1 = (ListView) findViewById(R.id.list_dokaza);
listView2 = (ListView) findViewById(R.id.list_pretpostavke);
ListObject data = new ListObject();

data.setName(premisa);
data.setRule("P");
data.setNumber(number);
details1.add(data);

textFormule.setText(formula);
final String[] f = formula.split(" ⊦", 2);

listAdapter = new ListPremiseAdapter(context, details1, R.layout.row);
listView1.setAdapter(listAdapter);
```

Instacirani objekt data je objekt tipa ListObject, a svaki objekt tipa ListObject sadrži varijablu name, rule i number. Sa details1.add(data), postavlja se objekt data na ArrayList<ListObject> , nakon što je objekt postavljen poziva se improvizirani adapter za liste. listAdapter ljepi sve podatke iz details1 na layout_row.xml datoteku. layout_row.xml predstavlja jedan element na listi, koji sadrži u sebi mjesto za name, rule i number. Nakon što su svi elementi iz details1 postavljeni na listAdapter, na listView1 se postavlja listAdapter. Isto tako, samo sa drugim podacima postavlja se listAdapter na listView2 pogled.

Slika 3 Postavljena lista1 i lista2 na MainActivity

```
        // postavljanje liste1 na slušač
listView1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                      String thingName;
                      int checked = 0;
                      int id = 0;
                      int id2 = 0;
                      int id3 = 0;
                      int id4 = 0;
                      int id5 = 0;
                      String thingRule;
                      String[] rules = new String[20];
                      String a = "";  String b = "";    String c = ""; String d = "";  String e = "";
                      SparseBooleanArray checkedItems = listView1.getCheckedItemPositions();

    // checkedItems vraća vrijednost true za sve pritisnute elemente liste

                      int count = listAdapter.getCount();
                      for (int j = 0; j < count; j++) {
                        if (checkedItems.get(j)) {
                        checked++;
```

```
// preuzimanje vrijednosti pritisnutih elemenata sa liste
                    thingName = details1.get(j).getName();
                    thingRule = details1.get(j).getRule();
                    rules[j] = thingRule;
                  .....
```

Kako bi program prepoznao koji elementi na listi su odabrani, poziva se listview1.getCheckedItemPositions(). Ova metoda postavlja vrijednost "true" na određeni element u nizu nazvanom checkedItems. To omogućuje programu da sazna vrijednosti i broj odabranih elemenata koristeći for petlju. Svaki puta kada program pronađe jedan od odabranih elementa, brojac naziva checked se poveća za jedan. Potom dodaje vrijednost imena odabranog elementa na varijablu thingName i pravilo uz element na varijablu thingRule. Sva pravila postavlja i u zaseban niz rules[j]. Ime elementa je u obliku formule npr. "A V B", dok je pravilo oblika npr. " vE".

```
// različiti slučajevi pritisnutih elemenata
switch (checked) {
        case 1:
        //svi slučajevi postavljaju ime zadnjeg odabranog elementa u zasebnu varijablu a,b,c,d,e
        a = a + thingName;

        // setUp, poziva različite provjere za format ovisno o zagradama, minusima i veličini podatka s liste
        //nakon provjere, kroz setUp se dođe do postavljanja podataka na drugu listu

        setUp(thingName, thingRule, count + 1, j + 1);
        id = j + 1;
break;

        case 2:
        // slučaj sa dva odabrana elementa sadrži veći dio koda i opisan je zasebno
        . . . .

break;
                // U slučaju 3 i 4 nijedan prijedlog se ne nalazi na drugoj listi, osim unosa nove pretpostavke
        case 3:
        // poziva metodu removefromList(details2), koja je konstruirana tako da makne sa listAdaptera sve elemente
        // osim prvog

        removefromList(details2);
        id3 = j + 1;
        c = c + thingName;
        c = checkSize(c);
```

```
break;
            case 4:
            id4 = j + 1;
            d = d + thingName;
            d = checkSize(d);
break;
            case 5:
            //  se odnosi na pravilo vE i objašnjeno je zasebno
            . . . .

break;

case default:
            //  U slučaju da je odabrano više elementa od 5
            // I u slučaju da je prethodno bio odabran samo 1 element, a sada više nije
            removefromList(details2);

break;
}
```

Kada je odabran jedan element liste1, poziva se metoda setUp(...).

Metoda setUp(...), za različite provjere veznika i zagrada kako bi se odredili prijedlozi za drugu listu:

```
private void setUp(String thingName, String thingRule, int thingNumber, int zaRule) {
    String partOne, partTwo;
    String check = thingName.replace("-", "");          //sve minuse u formuli zamjeni sa praznim mjestom
    char cijeliArray[] = check.toCharArray();           //postavi formulu bez minusa u niz simbola 'a', b'
    char prviZnak = cijeliArray[0];                     // odredi prvi  znak formule bez minusa
    if (check.length() == 1) {          //u slučaju da je formula bez minusa dužine od jednog znaka
      if (thingName.length() > 2) {

          //I u slučaju da je formula sa minusima duža od dva znaka, prvi dio formule postavi se
          //na formulu bez prva 2 znaka i pozove se setFormulaVeznik(...) metoda za postavljanje
          //prijedloga na drugu listu

          // u ovom slučaju (iako nije veznik) simbol minus se koristi kako bi se postavilo točno
          //pravilo na listu2, npr thingName= --A na listu2 će postaviti  pravilo –E (eliminacija negacije)

        partOne = thingName.substring(2, thingName.length());
        setFormulaVeznik(thingName, "-", partOne, "", thingNumber, thingRule, zaRule);
      }
```

```
else {
            // u slučaju da se formula sastoji od 2 znaka ili manje, postavi prijedloge na drugu listu preko metode
            //setFormulaVeznik(...),   slučaj : -A
,
            setFormulaVeznik (thingName, "\/", "", "", thingNumber, thingRule, zaRule);
        }


    }

    // u slucaju da  je prvi znak formule bez minusa jednak zagradi "("

    if (prviZnak == '(') {
      char[] charZagrada = thingName.toCharArray();   //postavi formulu na niz simbola imena charZagrada
      int c = findClosingParen(charZagrada, thingName.indexOf("("));   //metoda pronalaženja zatvorene zagrade

      partOne = thingName.substring(0, c + 1);   //postavi prvi dio formule do pronađene zagrade na varijablu partOne
            //U slučaju formule (A V B) V C postaviti će na :  (A V B)
           // isto tako -(-A V B) postaviti će na : –(-A V B)

      if (thingName.length() > partOne.length()) {

           //Slučaj  kada postoji drugi dio formule  :   (A V B) V C
           // drugi dio formule partTwo se postavlja na string koji počinje poslje drugog znaka nakon zagrade

           partTwo = thingName.substring(c + 2, thingName.length());

           //Odredi veznik i postavi prijedloge za formule kroz setFormulaVeznik

          String veznik = thingName.substring(c + 1, c + 2);

          setFormulaVeznik (thingName, veznik, partOne, partTwo, thingNumber, thingRule, zaRule);
}

else {  //kada ne postoji drugi dio slučaj :  –(-A V B)

          char treciZnak = cijeliArray[2]  //treći znak od formule gdje su svi minusi izbrisani, je veznik u ovom slučaju
         String veznik = String.valueOf(treciZnak);
          String[] parts = thingName.split(String.valueOf("\\("), 2);  // podjela formule na dva dijela prije i poslije (

//prvi dio formule parts[0] su minusi ispred zagrade ako postoje, a drugi dio parts[1] je ostatak formule nakon zagrade

          int a = parts[0].length() - 2;  // broj minusa bez prva dva minusa u formuli
```

```
//kada postoje barem dva minusa na početku formule, slučaj: -- ((A V B) V C)
        if (a >= 0) {

         //check varijabla postavljena je na formulu od mjesta drugog minusa do kraja formule, gube se dva minusa

          check = thingName.substring(parts[0].length() - a, thingName.length());
          check1 = parts[1].substring(0, parts[1].length() - 1);  // check1 postaje parts[1] bez posljednje zagrade

          if (a == 0) {

         //U slučaju da su točno dva minusa ispred prve zagrade, potrebno je maknuti zagrade
         //A to je prethodno odrađeno sa varijablom check1, pa je tako varijabla check postavljena na check1

            check = check1;

          }
         //postavljanje ispravnih prijedloga
          setFormulaVeznik (thingName, "-", check, check1, thingNumber, thingRule, zaRule);



        } else { //kada postoji točno 1 minus u formuli prije zagrade

          partTwo = parts[1].substring(0, parts[1].length() - 1);          //partTwo je formula bez minusa i zagrade
          setFormulaVeznik (thingName, "V", partOne, partTwo, thingNumber, thingRule, zaRule);
        }
      }
    }

    //Slučaj kada zagrada nije prvi znak :  -AVB, AVB
    else if ( thingName.length() > 1 && check.length() != 1)

    {
      String veznik = check.substring(1, 2);        Veznik je u ovom slučaju na drugom mjestu formule bez minusa.
      String[] parts = thingName.split(veznik, 2);

      provjeraVeznika(thingName, veznik, parts[0], parts[1], thingNumber, thingRule, zaRule);
    }
}
```

Slučaj kada su dva elementa odabrana na listi1 :

```
        case 2:
        //kada su dva elementa na listi 1 odabarana, potrebno je prvo obrisati prethodne elemente, kako bi se provela
        //provjera i postavili prijedlozi na listu 2 u prikladnom obliku
                                removefromList(details2);
                                b = b + thingName;
                                String aZ, bZ;
                                aZ = provjeraZagrade(b);
                                bZ = provjeraZagrade(a);

//U slučaju da je  jedan od elemenata pretpostavka npr. A (P), na listu2 treba postaviti prijedlog A(P) -> B, pravilo →I

                                // →I
                                 id2 = j + 1;
                                if (rules[id - 1] == "P") {
                                   provjeraVeznika(a, "P", bZ, aZ, count + 1, id + "," + id2, 0);


                                }
                                if (rules[id2 - 1] == "P") {
                                   provjeraVeznika(a, "P", aZ, bZ, count + 1, id + "," + id2, 0);


                                }

                             //Ako jedan element sadrži drugi i veznik mu je →, onda je potrebno izvršiti
                                    provjeru za eliminaciju → sa metodom provjeraEIndukcije(...).
                            // →E

                                String checkEI = "";
                                if (a.contains(b + "→") || a.contains("(" + b + ")" + "→")) {
                                   checkEI = provjeraEIndukcije(a, b, "→");
                                }



                                if (b.contains(a + "→") || b.contains("(" + a + ")" + "→")) {
                                   checkEI = provjeraEIndukcije(b, a, "→");
                                }
                                if (checkEI != "") {
                                   ListObject data = new ListObject();
                                   data.setName(checkEI);
                                   data.setNumber(count + 1);
                                   data.setRule(id + "," + id2 + "→E");
                                   details2.add(data);
                                   setOnList();
                                }
```

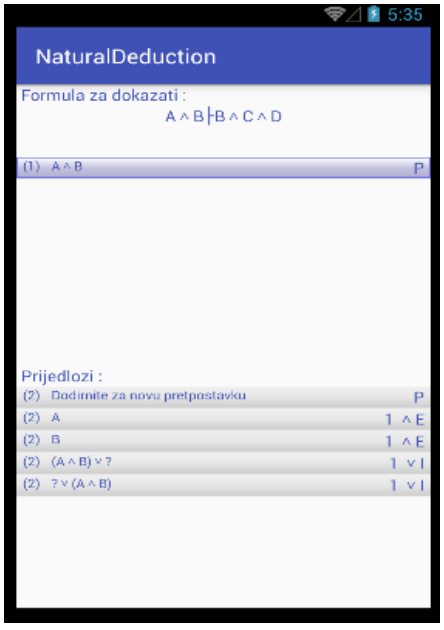
```
    //Svaki puta kada su odabrana dva elementa s liste 1, potrebno je postaviti prijedlog ∧I,  primjeri: A ∧ B i B ∧ A
                                    //∧I
                                    a = checkSize(a);
                                    b = checkSize(b);
                                //U slučaju da postoje dva minusa prije formule, postaviti prijedlog eliminacije minusa
                                    //-E
                                    if (a.equals("-" + b) || b.equals("-" + a)) {
                                      ListObject data = new ListObject();
                                      data.setName("⊥");
                                      data.setNumber(count + 1);
                                      data.setRule(id + "," + id2 + "-E");
                                      details2.add(data);
                                      setOnList();
                                    }
// U slučaju da je odabran element s vrijednošću  "⊥", potrebno je na listu dodati negirani drugi odabrani element,
//odnosno eliminaciju negacije
                                    //-I
                                    if (b.equals("⊥") || a.equals("⊥")) {
                                      if (b.equals("⊥")) {
                                        ListObject data = new ListObject();
                                        data.setName("-" + a);
                                        data.setNumber(count + 1);
                                        data.setRule(id + "," + id2 + "-E");
                                        details2.add(data);
                                        setOnList();
                                      } else  {
                                        ListObject data = new ListObject();
                                        data.setName("-" + b);
                                        data.setNumber(count + 1);
                                        data.setRule(id + "," + id2 + "-E");
                                        details2.add(data);
                                        setOnList();
                                      }

                                    }
//provjeraVeznika obavlja potrebne provjere za ostala pravila
                                    provjeraVeznika(a, "&", aZ, bZ, count + 1, id + "," + id2, 0);

}break;
```

U slučaju da je odabrano pet elementa sa liste jedan, potrebno je provjeriti da li se izvelo pravilo eliminacije disjunkcije, vE. Za pravilo eliminacije disjunkcije potrebno je odabrati element koji sadrži veznik v oko kojeg će se izvršiti eliminacije veznika. Nakon toga potrebno je unijeti pretpostavku da je lijevi dio formule do veznika v istinit i izvesti poddokaz, isto tako i za pretpostavku da je desni dio formule od veznika v istinit. Kada su oba poddokaza izvedena, potrebno ih je izabrati i zajedno s njima pretpostavke i element od kojeg je pokrenuta eliminacija veznika disjunkcije.

```
                                    case 5:

                                    int[] idLD= new int[2];
                                    int[] idSubProof = new int[2];
                                    idLD[0]=0;
                                    idLD[1]=1;
                                    idSubProof[0]=0;  //id poddokaza
                                    idSubProof[1]=0; //id poddokaza
                                    id5 = j + 1;
                                    e = e + thingName;
                                    e = checkSize(e);
                                    String[] part = new String[]{"", ""};
                                //program provjera koji element sadrži veznik v
                                    if (a.contains("v") && !rules[id - 1].equals("P")) {
                                      part = checkForV(a); //podjela formule a na dva dijela po vezniku
//provjeraLiDesno, provjerava da li postoj elementi koji je isti kao formula a do veznika v, kao i element koji je isti kao
//formula a poslije veznika v
                                      idLD = provjeraLiDesno(part, b, c, d, e, id2, id3, id4, id5);
                                      if (idLD[0] != 0 && idLD[1] != 0) {
                                //U slučaju da ti elementi postoje, određuju se elementi koji su poddokazi
                                        idSubProof = odrediPodDokaz(idLD, id2, id3, id4, id5);

                                      }
                                    } //ista provjera odvija se za sve elemente
                                   . . . .
                                    if (e.contains("v") && !rules[id5 - 1].equals("P")) {
                                      part = checkForV(b);
                                      idLD = provjeraLiDesno(part, b, c, d, a, id2, id3, id4, id);
                                      if (idLD[0] != 0 && idLD[1] != 0) {

                                        dSubProof = odrediPodDokaz(idLD, id2, id3, id4, id);
                                      }
}
                                        String a1="a1";
```

```
                    String b1="b1";
//provjera različitosti id-a dva poddokaza, u slučaju razlike, a1 i b1 se postavljaju na  ime elementa sa id-em poddokaza
                    if(idSubProof[0] != idSubProof[1]) {
                        a1 = details1.get(idSubProof[0] - 1).getName();
                       b1 = details1.get(idSubProof[1] - 1).getName();
                    }
                    // U slučaju da su a1 i b1 isti, ispisuje se formula poddokaza na listu 2 kao prijedlog
                    if(a1.equals(b1) )
                    {
                       ListObject data = new ListObject();
                       data.setName(details1.get(idSubProof[0]-1).getName());
                       data.setNumber(count + 1);
                       data.setRule(id + "," + id2 + "," + id3 + "," + id4 + "," + id5 + " ∨E");
                       details2.add(data);
                       setOnList();
                    }
                    break;
```

Nakon što su određeni prijedlozi potrebni za postaviti na listu2 poziva se setFormulaVeznik(...) metoda. Ova metoda za različitu vrijednost veznika postavlja na listu2 različite elemente. Pa se tako za veznik konjukcije ∧, na prijedlog liste2 postavljaju elementi koji slijede iz pravila eliminacije konjukcije ∧E. To su prva dva prijedloga nakon elementa za dodavanje pretpostavke.



Slika 4 Postavljanje prijedloga ∧E na listu 2

Primjer jednog slučaja setFormulaVeznik(...) metode:

```
private void setFormulaVeznik(String thingName, String veznik, String partOne, String partTwo, int thingNumber, String
rule, int number) {
    ListObject data3;
    char[] charZagrada;
    switch (veznik) {
      case "∧":
        . . . . .
        break;
      case "→":
        . . . .
        break;
      case "↔":
        . . . .
        break;
      case "∨":     //U slučaju da je kontradikcija element liste1, postavi prijedlog za unos formule sa pravilom K
        if (thingName.equals("⊥")) {
          data3 = new ListObject();
          data3.setName("?");
          data3.setRule(number + " " + "K");
          data3.setNumber(thingNumber);
          details2.add(data3);
        }
     if (!partTwo.isEmpty()) {   // U slučaju da se element sastoji od elementa sa jednim znakom, postavi znak u zagradu
          thingName = "(" + thingName + ")";
        }
         // Postavljanje pravila na details2 listu, ime se postavlja na ime odabranog elementa i kombinaciju "∨?"
        data3 = new ListObject();
        data3.setName(thingName + "∨?");
        data3.setRule(number + " " + "∨I");
        data3.setNumber(thingNumber);

        details2.add(data3);
        setOnList();
        data3 = new ListObject();

        data3.setName("?∨" + thingName);
        data3.setRule(number + " " + "∨I");
        data3.setNumber(thingNumber);

        details2.add(data3);
        setOnList();
        break;

case "-":
```

```
        ....
        break;
    case "&":
        ....
            break;

    case "P":
        ....
        break;
  }
}
```

U slučaju veznika ∨ na listu2 se postavljaju dva elementa. Prvi je formula + ?, a drugi je ? + formula. U slučaju da je odabrana formula "A v B", na listu se ispisuje element "(A v B) +?" i element "? + (A v B) ". Postoji i slučaj da je odabrani element sa liste1 ⊥, tada se na listu2 ispisuje i jedan zaseban element "?" sa pravilom K.

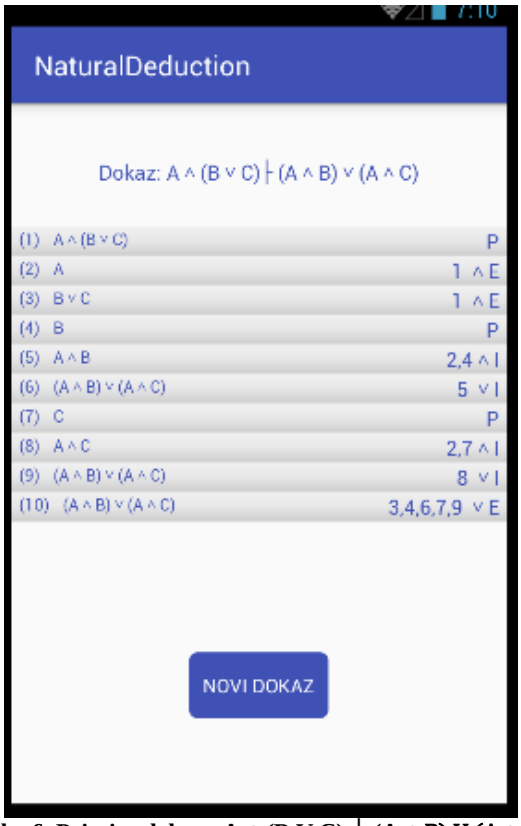
Slika 5 Primjer postavljanja prijedloga na listu2 po vezniku v

Kako bi provjerio dokaz, program za svaki unešeni posljednji element provjerava je li isti jednak desnoj strani prvotne formule napisane na vrhu aktivnosti. Osim u slučaju kada je zadnja formula ispisana pravilom indukcije vI ili je pretpostavka. U slučaju da je dokaz izvršen, poziva se Intent koji prenosi listu1 na aktivnost EndActivity.

4.2.4 EndActivity

EndActivity pogled se postavlja na xml datoteku end_activity.xml. Ova xml datoteka sastoji se od jednog listView-a, jednog gumba za pokretanje novog dokaza i teksta za dokaz :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"  >
    <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
      android:layout_marginTop="40dp"
      android:textColor="@color/colorPrimary"
      android:text=" Dokaz: "
      android:layout_centerHorizontal="true"
      android:textSize="15sp"
      android:id="@+id/textFormule"
      android:layout_marginBottom="30dp"
      />
    <ListView android:layout_width="match_parent"
      android:layout_below="@+id/textFormule"
      android:layout_height="310dp"
      android:id="@+id/list_dokaz"
      android:listSelector="@drawable/list_selector"
      android:choiceMode="none">
    </ListView>
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/list_dokaz"
  android:layout_centerHorizontal="true"
  android:text="Novi dokaz"
  android:onClick="start"
  android:background="@drawable/button_selector"
  android:textColor="@drawable/button_text_color"
  /></RelativeLayout>
```



Slika 6. Primjer dokaza A ∧ (B ∨ C) ⊢ (A ∧ B) ∨ (A ∧ C)

```
public class EndActivity extends AppCompatActivity {


  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.end_activity);  //postavljanje pogleda na end_activity.xml
    Intent intent = getIntent();
    Bundle args = intent.getBundleExtra("BUNDLE"); //dohvaćanje objekata sa aktivnosti MainActivity
    ArrayList<ListObject> object = (ArrayList<ListObject>) args.getSerializable("ARRAYLIST");
    ListView listView = (ListView) findViewById(R.id.list_dokaz);
    ListPremiseAdapter listAdapter = new ListPremiseAdapter(this,object,R.layout.row);
    listView.setAdapter(listAdapter);            //postavljanje liste s dokazom
   String premisa= intent.getStringExtra("premisa");
    String dokaz =intent.getStringExtra("dokaz");
    TextView textFormule = (TextView) findViewById(R.id.textFormule);
    textFormule.setText("Dokaz: "+ premisa +"â”ś "+ dokaz);
  }
  public void start(View v)              //gumb za pokretanje SetFormulaActivity-a, odnosno forme za upis novog dokaza
  {
    Intent i = new Intent(this,SetFormulaActivity.class);
    startActivity(i);
  }}
```

5. ZAKLJUČAK

U ovom diplomskom radu potrebno je bilo kreirati funkcionalno sučelje za prirodnu dedukciju. Za ostvarenja ovog cilja, kreirana je android aplikacija. Unutar aplikacije svako pojedinačno pravilo prirodne dedukcije, pretvoreno je u prijedloge za jednostavnije korištenje. Ovisno o tome koje je pravilo potrebno, prijedlozi se automatski generiraju na sučelju aplikacije. Ograničenje aplikacije je veličina ekrana, pa nije pogodna za dokaze sa dugačkim formulama. Aplikacija je korisna za vježbanje prirodne dedukcije, te kreiranje i provjeru dokaza.

LITERATURA

[1] M. Vukovic, Matematička logika, Zagreb: Element, 2009..

[2] N. M. Boris Čulina, Uvod u matematičku loginku i osnove matematike, Velika Gorica: Veleučilište Velika Gorica, 2012.

[3] M. Karaga, »Programiranje za suvremene procesore,« [Mrežno]. URL: https://web.math.pmf.unizg.hr/~karaga/android/android_skripta.pdf.

[4] B. Jakuben, »blog.teamtreehouse.com,« treehouse, 14 Studeni 2012. [Mrežno]. URL: http://blog.teamtreehouse.com/java-basics-for-android-development-part-1.

SAŽETAK

Tema ovog diplomskog rada je izrada programskog sučelja za prirodnu dedukciju. A svrha samog diplomskog rada je omogućiti korisniku korištenje progamskog sučelja kako bi mogao kreirati dokaze za prirodnu dedukciju kroz aplikaciju.

U prvom poglavlju objašnjen je koncept prirodne dedukcije i simbolizmi korišteni u istoj, kao i jezik prirodne dedukcije. Objašnjeno je dokazivanje i način zapisivanja istog. Nakon toga opisana su pravila prirodne dedukcije koja je potrebno postaviti za pravila programa. Drugo poglavlje se bavi samim sustavom u kojem je sučelje izrađeno, odnosno Android sustavom i Java programskim jezikom. Objašnjena je povijest sustava i zašto je sam sustav izabran za ovaj rad. Treće poglavlje odnosi se na izradu i na rad same aplikacije.

Ključne riječi: prirodna dedukcija, pravila, Android, Java

ABSTRACT

The theme of this paper is creation of an interface for natural deduction. And the purpose of the paper itself is to enable the users of the interface to create natural deduction proofs through the use of the application.

The first chapter explains the concept of natural deduction, as well as the symbols used. The same chapter explains argumentation and the way it is written. After the argumentation chapter explains the rules of the natural deduction that are necessary for the application. Second chapter is about the the system in which the interface is made, Android system and the Java programming language. It briefly explains the history of the system and why it was choosen for this assignment. Third chapter refers to the creation of the application and on how the application works.

Keywords: natural deduction, rules, Android, Java

ŽIVOTOPIS

Ivica Škrobo je rođen 03. srpnja 1991. godine u Vinkovcima. Veći dio svog dosadašnjeg života proveo je u Otoku. Godine 1998. upisuje OŠ Josipa Lovretića u Otoku. Nakon završene osnovne škole, Ivica 2006. godine upisuje matematički smjer u Gimnaziji Matije Antuna Reljkovića u Vinkovcima. Nakon uspješne završene 4. godine školovanja, upisuje Elektrotehnički fakultet u Osijeku, smjer računarstvo.

Potpis: