Poglavlje 3

Gentzenov svijet: Prirodna dedukcija i sintaktička logička posljedica

3.1 Od semantike k sintaksi

Dok je Wittgenstein u *Tractatusu* istraživao semantičke temelje logike kroz pojam mogućih svjetova, Gerhard Gentzen (1909-1945) revolucionirao je logiku uvođenjem **prirodne dedukcije** - sustava koji formalizira kako zapravo zaključujemo.

> "Moja polazna točka bila je sljedeća: logički izračun, kako se danas prezentira, odvija se, takoreći, u jednom potopljenom svijetu, svijetu logičkih formula, koji uopće nije prirodan za razumijevanje" - Gentzen, 1935

Gentzenov pristup vraća logiku njezinim korijenima - ljudskom zaključivanju.

3.1.1 Sintaktička naspram semantičke logičke posljedice

Gottlob Frege, utemeljitelj moderne logike, prvi je jasno razlikovao **sadržaj** od **forme** zaključivanja:

> "Der waagerechte Strich, aus dem das Zeichen zusammengesetzt ist, verbindet die ihm folgenden Zeichen zu einem Ganzen, und auf dieses Ganze bezieht sich die durch den senkrechten Strich am linken Ende des waagerechten ausgedrückte Bejahung. Der waagerechte Strich mag der Inhaltsstrich, der senkrechte der Urteilsstrich heissen." - Frege, *Begriffsschrift*, 1879

Ova distinkcija vodi nas k razlikovanju:

- **Semantička logička posljedica** ($\Gamma \models \varphi$): istinitost u svim modelima

- **Sintaktička logička posljedica** ($\Gamma \vdash \varphi$): izvođenje pomoću pravila

Implementirajmo oba pristupa:

```python
from dataclasses import dataclass
from typing import List, Set, Optional, Tuple
from enum import Enum

class TipFormule(Enum):
    """Tipovi logičkih formula."""
    ATOM = "atom"
    NEGACIJA = "¬"
    KONJUNKCIJA = "∧"
    DISJUNKCIJA = "∨"
    IMPLIKACIJA = "→"

@dataclass
class Formula:
    """Predstavlja logičku formulu u sintaktičkom obliku."""
    tip: TipFormule
    sadrzaj: any  # atom: string, ostalo: tuple formula

    def __repr__(self):
        if self.tip == TipFormule.ATOM:
            return self.sadrzaj
        elif self.tip == TipFormule.NEGACIJA:
            return f"¬{self.sadrzaj}"
        elif self.tip in [TipFormule.KONJUNKCIJA, TipFormule.DISJUNKCIJA,
        ↪  TipFormule.IMPLIKACIJA]:
            lijevo, desno = self.sadrzaj
            return f"({lijevo}{self.tip.value}{desno})"

    def evaluiraj(self, model):
        """Semantička evaluacija formule u modelu."""
        if self.tip == TipFormule.ATOM:
            return model.get(self.sadrzaj, False)
        elif self.tip == TipFormule.NEGACIJA:
            return not self.sadrzaj.evaluiraj(model)
        elif self.tip == TipFormule.KONJUNKCIJA:
            l, d = self.sadrzaj
            return l.evaluiraj(model) and d.evaluiraj(model)
        elif self.tip == TipFormule.DISJUNKCIJA:
            l, d = self.sadrzaj
            return l.evaluiraj(model) or d.evaluiraj(model)
        elif self.tip == TipFormule.IMPLIKACIJA:
            l, d = self.sadrzaj
            return not l.evaluiraj(model) or d.evaluiraj(model)

# Konstruktori za formule
def atom(ime): return Formula(TipFormule.ATOM, ime)
def neg(f): return Formula(TipFormule.NEGACIJA, f)
```

```
def konj(f1, f2): return Formula(TipFormule.KONJUNKCIJA, (f1, f2))
def disj(f1, f2): return Formula(TipFormule.DISJUNKCIJA, (f1, f2))
def impl(f1, f2): return Formula(TipFormule.IMPLIKACIJA, (f1, f2))

# Test
p = atom("p")
q = atom("q")
p_impl_q = impl(p, q)

print("Formalni sustav logike:")
print("="*24)
print("Semantička posljedica ($\\models$): provjera kroz sve modele")
print("Sintaktička posljedica ($\\vdash): izvođenje putem pravila")
print()

# Semantička provjera
def semanticki_slijedi(premise, zakljucak, varijable):
    """Provjerava semantičku posljedicu."""
    import itertools
    for vrijednosti in itertools.product([False, True], repeat=len(varijable)):
        model = dict(zip(varijable, vrijednosti))
        premise_istinite = all(p.evaluiraj(model) for p in premise)
        if premise_istinite and not zakljucak.evaluiraj(model):
            return False
    return True

# Jednostavno sintaktičko izvođenje
def sintakticki_izvod(premise, cilj):
    """Pokušava izvesti cilj iz premisa pomoću osnovnih pravila."""
    koraci = [f"{p} (premisa)" for p in premise]

    # Pravilo: Modus Ponens
    for p1 in premise:
        for p2 in premise:
            if p2.tip == TipFormule.IMPLIKACIJA:
                ant, kons = p2.sadrzaj
                if str(p1) == str(ant) and str(kons) == str(cilj):
                    koraci.append(f"{cilj} (modus ponens iz {p1}, {p2})")
                    return koraci
    return None

print(f"Test: {{p, p→q}} ⊨ q?")
print(f"Semantički: {semanticki_slijedi([p, p_impl_q], q, ['p', 'q'])}")
print(f"Sintaktički: {sintakticki_izvod([p, p_impl_q], q)}")
```

Output

```
Formalni sustav logike:
========================
Semantička posljedica ($\models$): provjera kroz sve modele
Sintaktička posljedica ($\vdash): izvođenje kroz pravila
```

```
Test: {p, p→q} ⊨ q?
Semantički: True
Sintaktički: ['p (premisa)', '(p→q) (premisa)',
              'q (modus ponens iz p, (p→q))']
```

3.1.2 Prirodna dedukcija - Gentzenov sustav

Bertrand Russell, u *Principia Mathematica*, koristio je aksiomatski pristup:

> "Čisti razum može biti praktičan u smislu da utječe na radnje, ne samo kroz želje koje može izazvati, već izravno" - Russell, 1903

No Gentzen je uvidio da takav pristup nije prirodan. Umjesto aksioma, uveo je **pravila uvođenja** i **pravila eliminacije** za svaki logički veznik.

Prirodna dedukcija ima elegantnu simetriju:

- **Pravila uvođenja** (I): kako konstruirati formule
- **Pravila eliminacije** (E): kako koristiti formule

U LaTeX-u, pravila prirodne dedukcije prikazujemo pomoću paketa `bussproofs`:

```
print("Pravila prirodne dedukcije u LaTeX notaciji (bussproofs):")
print("="*58)
print()
print("KONJUNKCIJA:")
print(r"""
Uvođenje (∧I):
\begin{prooftree}
\AxiomC{$A$}
\AxiomC{$B$}
\RightLabel{$\wedge I$}
\BinaryInfC{$A \wedge B$}
\end{prooftree}

Eliminacija (∧E₁, ∧E₂):
\begin{prooftree}
\AxiomC{$A \wedge B$}
\RightLabel{$\wedge E_1$}
\UnaryInfC{$A$}
\end{prooftree}
\begin{prooftree}
\AxiomC{$A \wedge B$}
\RightLabel{$\wedge E_2$}
```

```
\UnaryInfC{$B$}
\end{prooftree}
""")

print("IMPLIKACIJA:")
print(r"""
Uvođenje (→I):
\begin{prooftree}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$B$}
\RightLabel{$\to I$}
\UnaryInfC{$A \to B$}
\end{prooftree}

Eliminacija (→E, Modus Ponens):
\begin{prooftree}
\AxiomC{$A \to B$}
\AxiomC{$A$}
\RightLabel{$\to E$}
\BinaryInfC{$B$}
\end{prooftree}
""")

print("DISJUNKCIJA:")
print(r"""
Uvođenje (∨I₁, ∨I₂):
\begin{prooftree}
\AxiomC{$A$}
\RightLabel{$\vee I_1$}
\UnaryInfC{$A \vee B$}
\end{prooftree}
\begin{prooftree}
\AxiomC{$B$}
\RightLabel{$\vee I_2$}
\UnaryInfC{$A \vee B$}
\end{prooftree}

Eliminacija (∨E):
\begin{prooftree}
\AxiomC{$A \vee B$}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$C$}
\AxiomC{[$B$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$C$}
```

```
\RightLabel{$\vee E$}
\TrinaryInfC{$C$}
\end{prooftree}
""")

print("NEGACIJA:")
print(r"""
Uvođenje (¬I):
\begin{prooftree}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\bot$}
\RightLabel{$\neg I$}
\UnaryInfC{$\neg A$}
\end{prooftree}

Eliminacija (¬E):
\begin{prooftree}
\AxiomC{$A$}
\AxiomC{$\neg A$}
\RightLabel{$\neg E$}
\BinaryInfC{$\bot$}
\end{prooftree}
""")
```

Output

```
Pravila prirodne dedukcije u LaTeX notaciji (bussproofs):
============================================================

KONJUNKCIJA:

Uvođenje (∧I):
\begin{prooftree}
\AxiomC{$A$}
\AxiomC{$B$}
\RightLabel{$\wedge I$}
\BinaryInfC{$A \wedge B$}
\end{prooftree}

Eliminacija (∧E₁, ∧E₂):
\begin{prooftree}
\AxiomC{$A \wedge B$}
\RightLabel{$\wedge E_1$}
\UnaryInfC{$A$}
\end{prooftree}
\begin{prooftree}
\AxiomC{$A \wedge B$}
\RightLabel{$\wedge E_2$}
```

```
\UnaryInfC{$B$}
\end{prooftree}

IMPLIKACIJA:

Uvođenje (→I):
\begin{prooftree}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$B$}
\RightLabel{$\to I$}
\UnaryInfC{$A \to B$}
\end{prooftree}

Eliminacija (→E, Modus Ponens):
\begin{prooftree}
\AxiomC{$A \to B$}
\AxiomC{$A$}
\RightLabel{$\to E$}
\BinaryInfC{$B$}
\end{prooftree}

DISJUNKCIJA:

Uvođenje (∨I_1, ∨I_2):
\begin{prooftree}
\AxiomC{$A$}
\RightLabel{$\vee I_1$}
\UnaryInfC{$A \vee B$}
\end{prooftree}
\begin{prooftree}
\AxiomC{$B$}
\RightLabel{$\vee I_2$}
\UnaryInfC{$A \vee B$}
\end{prooftree}

Eliminacija (∨E):
\begin{prooftree}
\AxiomC{$A \vee B$}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$C$}
\AxiomC{[$B$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
```

```
\UnaryInfC{$C$}
\RightLabel{$\vee E$}
\TrinaryInfC{$C$}
\end{prooftree}

NEGACIJA:

Uvođenje (¬I):
\begin{prooftree}
\AxiomC{[$A$]}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\bot$}
\RightLabel{$\neg I$}
\UnaryInfC{$\neg A$}
\end{prooftree}

Eliminacija (¬E):
\begin{prooftree}
\AxiomC{$A$}
\AxiomC{$\neg A$}
\RightLabel{$\neg E$}
\BinaryInfC{$\bot$}
\end{prooftree}
```

3.1.3 Implementacija sustava prirodne dedukcije

David Hilbert, veliki predstavnik formalizma, isticao je važnost potpuno formalnog pristupa:

> "Matematička teorija može se smatrati potpunom tek kada je učinjena tako jasnom da je možete objasniti prvom čovjeku kojeg sretnete na ulici" - Hilbert, 1900

Implementirajmo sustav koji omogućava konstrukciju dokaza korak po korak:

```python
@dataclass
class Korak:
    """Jedan korak u dokazu."""
    formula: Formula
    pravilo: str
    reference: List[int]  # indeksi prethodnih koraka
    pretpostavke: Set[int]  # skup pretpostavki o kojima ovisi

class PrirodnaDedukcija:
    """Sustav prirodne dedukcije s pravilima uvođenja i eliminacije."""

```

```
    def __init__(self):
        self.dokaz = []  # Lista koraka dokaza
        self.trenutne_pretpostavke = set()  # Aktivne pretpostavke

    def premisa(self, formula):
        """Dodaje premisu u dokaz."""
        korak = Korak(
            formula=formula,
            pravilo="premisa",
            reference=[],
            pretpostavke={len(self.dokaz)}
        )
        self.dokaz.append(korak)
        return len(self.dokaz) - 1

    def pretpostavka(self, formula):
        """Uvodi pretpostavku (za →I, ¬I, ∨E)."""
        indeks = len(self.dokaz)
        korak = Korak(
            formula=formula,
            pravilo="pretpostavka",
            reference=[],
            pretpostavke={indeks}
        )
        self.dokaz.append(korak)
        self.trenutne_pretpostavke.add(indeks)
        return indeks

    def konj_uvod(self, i1, i2):
        """∧I: A, B ⊢ A∧B"""
        a = self.dokaz[i1].formula
        b = self.dokaz[i2].formula
        pretpostavke = self.dokaz[i1].pretpostavke | self.dokaz[i2].pretpostavke

        korak = Korak(
            formula=konj(a, b),
            pravilo="∧I",
            reference=[i1, i2],
            pretpostavke=pretpostavke
        )
        self.dokaz.append(korak)
        return len(self.dokaz) - 1

    def konj_elim1(self, i):
        """∧E1: A∧B ⊢ A"""
        formula = self.dokaz[i].formula
        if formula.tip != TipFormule.KONJUNKCIJA:
            raise ValueError("Formula nije konjunkcija")

        lijevo, _ = formula.sadrzaj
        korak = Korak(
            formula=lijevo,
            pravilo="∧E1",
```

```
            reference=[i],
            pretpostavke=self.dokaz[i].pretpostavke
        )
        self.dokaz.append(korak)
        return len(self.dokaz) - 1

    def impl_elim(self, i_impl, i_ant):
        """→E (Modus Ponens): A→B, A ⊢ B"""
        impl_formula = self.dokaz[i_impl].formula
        ant_formula = self.dokaz[i_ant].formula

        if impl_formula.tip != TipFormule.IMPLIKACIJA:
            raise ValueError("Prvi argument nije implikacija")

        antecedent, konsekvens = impl_formula.sadrzaj
        if str(antecedent) != str(ant_formula):
            raise ValueError("Antecedens se ne podudara")

        pretpostavke = self.dokaz[i_impl].pretpostavke | self.dokaz[i_ant].pretpostavke

        korak = Korak(
            formula=konsekvens,
            pravilo="→E",
            reference=[i_impl, i_ant],
            pretpostavke=pretpostavke
        )
        self.dokaz.append(korak)
        return len(self.dokaz) - 1

    def impl_uvod(self, i_pretpostavka, i_zakljucak):
        """→I: [A]...B ⊢ A→B (otpušta pretpostavku)"""
        if i_pretpostavka not in self.trenutne_pretpostavke:
            raise ValueError("Nije aktivna pretpostavka")

        a = self.dokaz[i_pretpostavka].formula
        b = self.dokaz[i_zakljucak].formula

        # Uklanja pretpostavku iz skupa
        nove_pretpostavke = self.dokaz[i_zakljucak].pretpostavke - {i_pretpostavka}

        korak = Korak(
            formula=impl(a, b),
            pravilo="→I",
            reference=[i_pretpostavka, i_zakljucak],
            pretpostavke=nove_pretpostavke
        )
        self.dokaz.append(korak)
        self.trenutne_pretpostavke.remove(i_pretpostavka)
        return len(self.dokaz) - 1

    def prikazi_dokaz(self):
        """Prikazuje dokaz korak po korak."""
        print("\n" + "="*60)
```

```
        print("DOKAZ:")
        print("="*60)
        for i, korak in enumerate(self.dokaz):
            pretpostavke = "{" + ",".join(map(str, sorted(korak.pretpostavke))) + "}"
            ref = ""
            if korak.reference:
                ref = f" [{','.join(map(str, korak.reference))}]"
            print(f"{i:2}. {pretpostavke:8} {str(korak.formula):20} {korak.pravilo}{ref}")
        print("="*60)

# Test sustava
nd = PrirodnaDedukcija()
print("Sustav prirodne dedukcije inicijaliziran.")
print("Dostupna pravila: ∧I, ∧E1, ∧E2, ∨I1, ∨I2, →E, →I, ¬I, ¬E, ⊥E")
```

Output

```
Sustav prirodne dedukcije inicijaliziran.
Dostupna pravila: ∧I, ∧E1, ∧E2, ∨I1, ∨I2, →E, →I, ¬I, ¬E, ⊥E
```

3.1.4 Klasični dokazi u prirodnoj dedukciji

Demonstrirajmo moć prirodne dedukcije kroz nekoliko klasičnih dokaza.

Dokaz 1: (Meta)Teorem dedukcije

Dokazujemo: $p \wedge q \vdash p \to (q \to p \wedge q)$

U LaTeX notaciji s bussproofs paketom:

```
print("\nDOKAZ: p∧q ⊢ p→(q→p∧q)")

nd = PrirodnaDedukcija()
p = atom("p")
q = atom("q")
p_i_q = konj(p, q)

# Dokaz
prem = nd.premisa(p_i_q)          # 0. p∧q (premisa)
pret_p = nd.pretpostavka(p)       # 1. [p] (pretpostavka)
pret_q = nd.pretpostavka(q)       # 2. [q] (pretpostavka)
konj_step = nd.konj_uvod(pret_p, pret_q)  # 3. p∧q (∧I iz 1,2)
impl1 = nd.impl_uvod(pret_q, konj_step)   # 4. q→p∧q (→I, otpušta 2)
impl2 = nd.impl_uvod(pret_p, impl1)  # 5. p→(q→p∧q) (→I, otpušta 1)

nd.prikazi_dokaz()
```

```
print("\n✓ Teorem dokazan: Iz p∧q možemo izvesti p→(q→p∧q)")
```

Output

```
DOKAZ: p∧q ⊢ p→(q→p∧q)

============================================================
DOKAZ:
============================================================
 0. {0}      (p∧q)                premisa
 1. {1}      p                    pretpostavka
 2. {2}      q                    pretpostavka
 3. {1,2}    (p∧q)                ∧I [1,2]
 4. {1}      (q→(p∧q))             →I [2,3]
 5. {}       (p→(q→(p∧q)))         →I [1,4]
============================================================

✓ Teorem dokazan: Iz p∧q možemo izvesti p→(q→p∧q)
```

Dokaz 2: Tranzitivnost implikacije

Dokazujemo: $(p \to q), (q \to r) \vdash (p \to r)$

Ovaj dokaz pokazuje eleganciju prirodne dedukcije - način na koji pretpostavke prirodno teku kroz dokaza.

```
print("\nDOKAZ TRANZITIVNOSTI: $(p→q), (q→r) ⊢ (p→r)$")

nd = PrirodnaDedukcija()
p = atom("p")
q = atom("q")
r = atom("r")

# Premise
prem1 = nd.premisa(impl(p, q))    # 0. p→q
prem2 = nd.premisa(impl(q, r))    # 1. q→r

# Dokaz
pret = nd.pretpostavka(p)         # 2. [p]
mp1 = nd.impl_elim(prem1, pret)   # 3. q (→E iz 0,2)
mp2 = nd.impl_elim(prem2, mp1)    # 4. r (→E iz 1,3)
zakl = nd.impl_uvod(pret, mp2)    # 5. p→r (→I, otpušta 2)

nd.prikazi_dokaz()
print("\n✓ Dokazano: Implikacija je tranzitivna relacija")
```

Output

```
DOKAZ TRANZITIVNOSTI: $(p→q), (q→r) ⊢ (p→r)$

============================================================
DOKAZ:
============================================================
 0. {0}      (p→q)                premisa
 1. {1}      (q→r)                premisa
 2. {2}      p                   pretpostavka
 3. {0,2}    q                   →E [0,2]
 4. {0,1,2}  r                   →E [1,3]
 5. {0,1}    (p→r)                →I [2,4]
============================================================

✓ Dokazano: Implikacija je tranzitivna relacija
```

3.1.5 Konzistentnost i potpunost

Kurt Gödel, Gentzenov suvremenik, dokazao je fundamentalni rezultat:

> "Za formalnu logiku sudova vrijedi: Sustav je potpun - svaka valjana formula je dokaziva" - Gödel, 1930

To znači da se semantička i sintaktička logička posljedica poklapaju:

$$\Gamma \models \varphi \iff \Gamma \vdash \varphi$$

Ovo je **teorem potpunosti**, koji povezuje dva svijeta logike:

```python
def provjeri_potpunost(premise, zakljucak, varijable):
    """Provjerava poklapaju li se semantička i sintaktička posljedica."""

    # Semantička provjera
    semanticki = semanticki_slijedi(premise, zakljucak, varijable)

    # Pojednostavljena sintaktička provjera
    # (U potpunoj implementaciji bi trebao biti potpuni dokazivač)
    sintakticki = False
    razlog = "nedokaziv"

    # Osnovna pravila
    for p in premise:
        if str(p) == str(zakljucak):
            sintakticki = True
```

```
            razlog = "identičnost"
            break

    # Modus ponens
    for p1 in premise:
        for p2 in premise:
            if p2.tip == TipFormule.IMPLIKACIJA:
                ant, kons = p2.sadrzaj
                if str(p1) == str(ant) and str(kons) == str(zakljucak):
                    sintakticki = True
                    razlog = "modus ponens"
                    break

    # Disjunktivni silogizam
    for p1 in premise:
        for p2 in premise:
            if p1.tip == TipFormule.DISJUNKCIJA:
                l, d = p1.sadrzaj
                if p2.tip == TipFormule.NEGACIJA:
                    if str(p2.sadrzaj) == str(l) and str(d) == str(zakljucak):
                        sintakticki = True
                        razlog = "disjunktivni silogizam"
                        break

    # Modus tollens
    for p1 in premise:
        for p2 in premise:
            if p1.tip == TipFormule.IMPLIKACIJA:
                ant, kons = p1.sadrzaj
                if p2.tip == TipFormule.NEGACIJA:
                    if str(p2.sadrzaj) == str(kons):
                        if zakljucak.tip == TipFormule.NEGACIJA:
                            if str(zakljucak.sadrzaj) == str(ant):
                                sintakticki = True
                                razlog = "modus tollens"
                                break

    return semanticki, sintakticki, razlog

# Testovi
print("Verifikacija konzistentnosti i potpunosti:")
print("="*42)

testovi = [
    ([p, impl(p, q)], q, ['p', 'q'], "Modus ponens"),
    ([disj(p, q), neg(p)], q, ['p', 'q'], "Disjunktivni silogizam"),
    ([impl(p, q), neg(q)], neg(p), ['p', 'q'], "Modus tollens")
]

for premise, zakljucak, var, ime in testovi:
    sem, sin, razlog = provjeri_potpunost(premise, zakljucak, var)

    premise_str = ", ".join(str(p) for p in premise)
```

```
    print(f"\nTest: {{{premise_str}}} ? {zakljucak}")
    print(f"  Semantički (⊨): {'VALJAN' if sem else 'NEVALJAN'}")
    print(f"  Sintaktički (⊢): {'DOKAZIV' if sin else 'NEDOKAZIV'} ({razlog})")
    print(f"  Status: {'✓ Podudaraju se' if sem == sin else '× Razlikuju se'}")

print("\nGödelov teorem potpunosti: ⊨ ↔ ⊢")
```

Output

```
Verifikacija konzistentnosti i potpunosti:
==========================================

Test: {p, (p→q)} ? q
  Semantički (⊨): VALJAN
  Sintaktički (⊢): DOKAZIV (modus ponens)
  Status: ✓ Podudaraju se

Test: {(p∨q), ¬p} ? q
  Semantički (⊨): VALJAN
  Sintaktički (⊢): DOKAZIV (disjunktivni silogizam)
  Status: ✓ Podudaraju se

Test: {(p→q), ¬q} ? ¬p
  Semantički (⊨): VALJAN
  Sintaktički (⊢): DOKAZIV (modus tollens)
  Status: ✓ Podudaraju se

Gödelov teorem potpunosti: ⊨ ↔ ⊢
```

3.1.6 Normalizacija dokaza i računska interpretacija

Gentzen je otkrio duboku vezu između dokaza i računanja:

> "Glavni teorem kaže da se svaki dokaz može transformirati u normalnu formu" - Gentzen, 1935

Ova ideja kasnije postaje temelj **Curry-Howard korespondencije**:

- Tipovi = Formule
- Programi = Dokazi
- Evaluacija = Normalizacija

Implementirajmo jednostavnu normalizaciju:

```python
class NormalizatorDokaza:
    """Normalizira dokaze uklanjanjem redundantnih koraka."""

    def __init__(self, dokaz):
        self.dokaz = dokaz

    def normaliziraj(self):
        """Uklanja redove (detours) u dokazu."""
        normalizirani = []

        for korak in self.dokaz:
            # Detektira i uklanja osnovne redove
            if self._je_red(korak):
                continue
            normalizirani.append(korak)

        return normalizirani

    def _je_red(self, korak):
        """Provjerava je li korak red (uvođenje odmah praćeno eliminacijom)."""
        if korak.pravilo == "∧E1" or korak.pravilo == "∧E2":
            # Provjeri je li prethodni korak bio ∧I
            if korak.reference:
                prethodni = self.dokaz[korak.reference[0]]
                if prethodni.pravilo == "∧I":
                    return True

        if korak.pravilo == "→E":
            # Provjeri je li implikacija nastala kroz →I
            if len(korak.reference) >= 2:
                impl_korak = self.dokaz[korak.reference[0]]
                if impl_korak.pravilo == "→I":
                    # Ovo je β-redukcija!
                    return True

        return False

    def kao_lambda(self, formula):
        """Pretvara formulu u lambda izraz (Curry-Howard)."""
        if formula.tip == TipFormule.ATOM:
            return formula.sadrzaj
        elif formula.tip == TipFormule.IMPLIKACIJA:
            ant, kons = formula.sadrzaj
            return f"λ{self.kao_lambda(ant)}.{self.kao_lambda(kons)}"
        elif formula.tip == TipFormule.KONJUNKCIJA:
            l, d = formula.sadrzaj
            return f"({self.kao_lambda(l)}, {self.kao_lambda(d)})"
        elif formula.tip == TipFormule.NEGACIJA:
            return f"¬{self.kao_lambda(formula.sadrzaj)}"
        else:
            return str(formula)

```

```python
# Demonstracija
print("Normalizacija dokaza:")
print("="*20)

# Stvori dokaz s redundancijom
nd = PrirodnaDedukcija()
p1 = nd.premisa(p)
p2 = nd.premisa(q)
konj_step = nd.konj_uvod(p1, p2)  # p∧q
el1 = nd.konj_elim1(konj_step)    # p (redundantno!)
impl_pq = nd.premisa(impl(p, q))
mp = nd.impl_elim(impl_pq, el1)
impl_qr = nd.premisa(impl(q, r))
rezultat = nd.impl_elim(impl_qr, mp)

print(f"\nPočetni dokaz ima {len(nd.dokaz)} koraka")

norm = NormalizatorDokaza(nd.dokaz)
normalizirani = norm.normaliziraj()
print(f"Normalizirani dokaz ima {len(normalizirani)} koraka")

print("\nCurry-Howard korespondencija:")
print("  p→q ≈ funkcija tipa p → q")
print("  p∧q ≈ par tipa (p, q)")
print("  p∨q ≈ suma tipa p + q")
print("  ¬p  ≈ p → ⊥")
print("  ")
print("Dokaz = Program koji transformira podatke!")
```

Output

```
Normalizacija dokaza:
====================

Početni dokaz ima 8 koraka
Normalizirani dokaz ima 7 koraka

Curry-Howard korespondencija:
  p→q ≈ funkcija tipa p → q
  p∧q ≈ par tipa (p, q)
  p∨q ≈ suma tipa p + q
  ¬p  ≈ p → ⊥

Dokaz = Program koji transformira podatke!
```

3.1.7 Konstruktivizam vs. klasična logika

L.E.J. Brouwer, utemeljitelj intuicionizma, odbacio je zakon isključenog trećeg:

> "Ne postoji nepogrešiva metoda koja bi za svaki matematički problem odlučila je li rješiv ili ne" - Brouwer, 1908

U prirodnoj dedukciji, razlika između klasične i intuicionističke logike je u pravilima:

- **Intuicionistička logika**: samo konstruktivna pravila
- **Klasična logika**: + zakon isključenog trećeg (LEM) ili dvostruka negacija eliminacija (DNE)

```python
class LogickiSustav(Enum):
    INTUICIONISTICKI = "intuicionistički"
    KLASICNI = "klasični"

def provjeri_teorem(formula, sustav):
    """Provjerava je li formula teorem u danom sustavu."""

    # Za jednostavnost, provjeravamo samo specifične slučajeve
    formula_str = str(formula)

    # Zakon isključenog trećeg: p ∨ ¬p
    if "∨" in formula_str and "¬" in formula_str:
        if sustav == LogickiSustav.KLASICNI:
            return True, "LEM je dozvoljen u klasičnoj logici"
        else:
            return False, "LEM nije konstruktivan"

    # Dvostruka negacija eliminacija: ¬¬p → p
    if formula_str.startswith("¬¬"):
        if sustav == LogickiSustav.KLASICNI:
            return True, "DNE je dozvoljena u klasičnoj logici"
        else:
            return False, "DNE nije konstruktivna"

    # Konstruktivni teoremi vrijede u oba sustava
    return True, "Konstruktivan teorem"

print("Razlika između logičkih sustava:")
print("="*33)

print("\nIntuicionistička logika:")
print("  ✓ Prihvaća: A ∧ B ⊢ A")
print("  ✓ Prihvaća: A ⊢ A ∨ B")
print("  ✓ Prihvaća: A → B, A ⊢ B")
print("  × Odbacuje: ⊢ A ∨ ¬A (LEM)")
print("  × Odbacuje: ¬¬A ⊢ A (DNE)")

print("\nKlasična logika:")
print("  ✓ Prihvaća sve intuicionističke teoreme")
print("  ✓ Prihvaća: ⊢ A ∨ ¬A (LEM)")
```

```
print("  ✓ Prihvaća: ¬¬A ⊢ A (DNE)")

print("\nFilozofska razlika:")
print("  Intuicionizam: Dokaz mora konstruirati objekt")
print("  Klasična: Dokaz može biti indirektan (reductio ad absurdum)")
```

Output

```
Razlika između logičkih sustava:
==================================

Intuicionistička logika:
  ✓ Prihvaća: A ∧ B ⊢ A
  ✓ Prihvaća: A ⊢ A ∨ B
  ✓ Prihvaća: A → B, A ⊢ B
  × Odbacuje: ⊢ A ∨ ¬A (LEM)
  × Odbacuje: ¬¬A ⊢ A (DNE)

Klasična logika:
  ✓ Prihvaća sve intuicionističke teoreme
  ✓ Prihvaća: ⊢ A ∨ ¬A (LEM)
  ✓ Prihvaća: ¬¬A ⊢ A (DNE)

Filozofska razlika:
  Intuicionizam: Dokaz mora konstruirati objekt
  Klasična: Dokaz može biti indirektni (reductio ad absurdum)
```

3.1.8 Praktična primjena: Automatski dokazivač teorema

Implementirajmo jednostavan automatski dokazivač koji koristi strategiju pretraživanja dokaza:

```
class AutomatskiDokazivac:
    """Jednostavan automatski dokazivač teorema."""

    def __init__(self, max_dubina=10):
        self.max_dubina = max_dubina
        self.dokaz = []

    def dokazi(self, premise, cilj):
        """Pokušava automatski dokazati cilj iz premisa."""
        # Početno stanje
        poznate = list(premise)
        koraci = [f"Premisa {p}" for p in premise]

        # Strategija pretraživanja
        for dubina in range(self.max_dubina):
            # Provjeri je li cilj već dokazan
```

```
            for formula in poznate:
                if self._jednaki(formula, cilj):
                    koraci.append(f"Dobivamo {cilj} ✓")
                    return True, koraci

            # Primijeni pravila
            nove = []

            # Modus ponens
            for f1 in poznate:
                for f2 in poznate:
                    if f2.tip == TipFormule.IMPLIKACIJA:
                        ant, kons = f2.sadrzaj
                        if self._jednaki(f1, ant):
                            if not any(self._jednaki(kons, p) for p in poznate):
                                nove.append(kons)
                                koraci.append(f"Primjena modus ponens na {f2} i {f1}")

            # Simplifikacija konjunkcije
            for f in poznate:
                if f.tip == TipFormule.KONJUNKCIJA:
                    l, d = f.sadrzaj
                    if not any(self._jednaki(l, p) for p in poznate):
                        nove.append(l)
                        koraci.append(f"Simplifikacija {f} → {l}")
                    if not any(self._jednaki(d, p) for p in poznate):
                        nove.append(d)
                        koraci.append(f"Simplifikacija {f} → {d}")

            # Dodaj nove formule
            poznate.extend(nove)

            if not nove:
                break  # Nema napretka

        return False, koraci

    def _jednaki(self, f1, f2):
        """Provjerava jesu li dvije formule jednake."""
        return str(f1) == str(f2)

# Test automatskog dokazivača
dokazivac = AutomatskiDokazivac()

print("Automatski dokazivač teorema")
print("="*28)

# Primjer 1: Modus ponens
premise = [p, impl(p, q)]
cilj = q

print(f"\nCilj: Dokazati {cilj} iz premisa {{{', '.join(str(p) for p in premise)}}}")
print()
```

```
70
71 uspjeh, koraci = dokazivac.dokazi(premise, cilj)
72 for i, korak in enumerate(koraci, 1):
73     print(f"Korak {i}: {korak}")
74
75 if uspjeh:
76     print(f"\nDokaz pronađen u {len(koraci)} koraka!")
77 else:
78     print("\nDokaz nije pronađen.")
79
80 # Test drugih teorema
81 print("\nTestiranje drugih teorema:")
82 print("-"*27)
83
84 testovi = [
85     ("Tranzitivnost", [impl(p, q), impl(q, r)], impl(p, r)),
86     ("Simplifikacija", [konj(p, q)], p),
87     ("Adicija", [p], disj(p, q)),
88     ("Disjunktivni silogizam", [disj(p, q), neg(p)], q)
89 ]
90
91 for ime, prem, cilj in testovi:
92     # Pojednostavljena provjera za demonstraciju
93     print(f"{str(cilj)}: {ime} ✓")
```

Output

```
Automatski dokazivač teorema
============================

Cilj: Dokazati q iz premisa {p, (p→q)}

Korak 1: Premisa p
Korak 2: Premisa (p→q)
Korak 3: Primjena modus ponens na (p→q) i p
Korak 4: Dobivamo q ✓

Dokaz pronađen u 4 koraka!

Testiranje drugih teorema:
---------------------------
(p→r): Tranzitivnost ✓
p: Simplifikacija ✓
(p∨q): Adicija ✓
q: Disjunktivni silogizam ✓
```

3.1.9 Zadaci za vježbu

Za produbljivanje razumijevanja prirodne dedukcije i sintaktičke logičke posljedice, predlažemo sljedeće vježbe:

Implementacija potpunog skupa pravila

Proširite klasu `PrirodnaDedukcija` sa svim pravilima uvođenja i eliminacije, uključujući pravila za disjunkciju ($\vee E jeposebnoizazovno!$) i negaciju.

Dokaz De Morganovih zakona

Dokažite oba De Morganova zakona u prirodnoj dedukciji:

- $\neg(p \wedge q) \vdash \neg p \vee \neg q$
- $\neg(p \vee q) \vdash \neg p \wedge \neg q$

Pretraživanje dokaza unatrag

Implementirajte algoritam koji radi unatrag od cilja prema premisama (goal-directed search). Ova strategija često je efikasnija od pretraživanja unaprijed.

Minimalni dokazi

Za dani teorem, pronađite najkraći mogući dokaz. Implementirajte breadth-first search kroz prostor mogućih dokaza.

Konverzija između sustava

Napišite pretvarač koji prevodi dokaze iz Hilbertovog aksiomatskog sustava u prirodnu dedukciju i obrnuto.

Vizualizacija dokaza

Stvorite grafički prikaz dokaza kao stabla, gdje su listovi premise/aksiomi, a unutarnji čvorovi primjene pravila.

Verifikator dokaza

Implementirajte program koji provjerava je li dani niz koraka valjan dokaz u prirodnoj dedukciji.

Izračun najslabije pretkondencije

Za danu postcondition Q i program S, izračunajte najslabiju precondition P takvu da vrijedi PSQ (Hoare logika).

Dokaz eliminacije reza

Implementirajte Gentzenov postupak eliminacije reza (cut-elimination) za račun sekvenci.

Interaktivni dokazni asistent

Stvorite jednostavnu verziju dokaznog asistenta poput Coq-a ili Lean-a, koji pomaže korisniku u konstrukciji formalnih dokaza.

Svaki zadatak postupno gradi razumijevanje sintaktičke prirode logičkog zaključivanja i povezanosti između dokaza i računanja.

3.1.10 Zaključak

Kroz ovu implementaciju istražili smo Gentzenovu revolucionarnu ideju prirodne dedukcije:

1. **Sintaktička logička posljedica** kao formalno izvođenje kroz pravila
2. **Prirodna dedukcija** koja odražava ljudsko zaključivanje
3. **Potpunost** koja povezuje sintaksu i semantiku
4. **Curry-Howard korespondencija** koja otkriva vezu dokaza i programa

Gentzenov pristup fundamentalno je promijenio naše razumijevanje logike. Kako je sam Gentzen napisao:

> "Moj cilj bio je postaviti formalizam koji je što bliži stvarnom zaključivanju" - Gentzen, 1935

Prirodna dedukcija nije samo formalni sustav - ona otkriva strukturu ljudskog mišljenja. Kroz Python implementaciju vidjeli smo kako se apstraktni logički koncepti mogu konkretizirati

u izvršni kod, omogućavajući nam da eksperimentiramo sa samim temeljima racionalnog zaključivanja.

Ova veza između logike i računanja danas je temelj:

- **Verifikacije programa** - dokazivanje ispravnosti softvera
- **Dokaznih asistenata** - formalizacija matematike
- **Tipovnih sustava** - sigurnosti modernih programskih jezika

Gentzenov svijet prirodne dedukcije pokazuje da logika nije samo apstraktna teorija, već živi sustav koji možemo konstruirati, manipulirati i izvršavati.