Theodore: Interaktivni dokazivač teorema za logiku prvog reda

Andrija Urošević

12. septembar 2024.

Sažetak

Interaktivni dokazivači teorema donose preciznost i poverljivost oblastima koji se u velikoj meri oslanjaju na formalnu korektnost. Sa jedne strane, omogućavaju korektnost softvera i hardvera za sisteme čija je tačnost od ključnog značaja. Dok sa druge strane, omogućavaju matematičku strogost pri formulaciji i dokazivanju matematičih teorema. U ovom radu će biti predstavljen interaktivni dokazivač teorema Theodore koji podržava formulaciju i dokazivanje teorema unutar okvira logike prvog reda.

1 Uvod

Theodore je interaktivni dokazivač teorema koji podržava formulisanje i dokazivanje teorema unutar okvira logike prvog reda. Implementiran je u funkcionalnom programskom jeziku Haskell. Pokreće se i izvršava u interaktivnom okruženju za Haskell: GHCi. Zbog toga, Theodore se može smatrati kao nadogradnja funkcionalnog programskog jezika Haskell. Još jedna karakteristika, interaktivni dokazivač teorema Theodore je minimalnost, odnosno jedino omogućava dokazivanje teorema primenom pravila prirodne dedukcije.

Formulizacija teoreme, u interaktivnom dokazivaću teorema Theodore, se sastoji u formulisanju liste pretpostavki *Assumptions* (od kojih svaka predstavlja formulu logike prvog reda), kao i zaključka *Conclusion* koji, takođe, predstavlja formulu logike provog reda. Odnosno, iz liste pretpostavki $[\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n]$ izvodimo zaključak $\mathcal{C}$. To preciznije možemo da zapišemo kao:

$$[\text{thm\_x}]\ \frac{\mathcal{H}_1 \qquad \mathcal{H}_2 \qquad \ldots \qquad \mathcal{H}_n}{\mathcal{C}}$$

Shodno tome, teorema thm_x predstavlja jedno pravilo zaključivanja. Sa druge strane, teorema thm_x predstavlja pravilo zaključivanja samo onda kada se odgovarajući cilj (koji postavlja teorema thm_x) dokaže.

Interaktivni dokazivač teorema Theodore pravila prirodne dedukcije predstavlja kao funkcije koje transformišu trenutnu formulaciju teoreme, odnosno transformišu cilj koji treba dokazati. Na primer, cilj se može podeliti u više manjih podciljeva, ili jedan od podciljeva biva izmenjen u skladu sa pravilom prirodne dedukcije. Zbog toga, dokaz teoreme thm_x predstavlja kompoziciju funkcija gde svaka od funkcija predstavlja jedno pravilo prirodne dedukcije.

1.1 Drugi interaktivni dokazivači teorema

Jedan od prvih interaktivni dokazivača teorema je AUTOMATH, koji je kreirao de Brujn (1967) [4]. AUTOMATH se zasniva na *prosto tipiziranom lambda računu* koga je uveo Čerč [5, 6]. Prosto tipizirani lambda račun je nastao po uzoru na *teoriju tipova* koju je originalno uveo Rasel (1908) [24], pokušavajući da izbegne paradokse pri zasnivanju matematike preko teorije skupova.

Teorija tipova prati *intuicionistički pristup* koji je nastao po Brauverovoj doktrini [3]. Brauver je smatrao da se celokupna matematika, uključujući i sam koncept dokaza, zasniva na konceptu *konstrukcije* (računa/algoritma/programa klasifikovanog tipom). Sa druge strane, bavljenje matematikom predstavlja fundamentalno ljudsku aktivnos, odnosno sposobnost izvršavanja algoritma u svrhu izvođenja mentalne konstrukcije je fundamentalno ljudska.

Intuicionistički pristup, dalje razvijaju Kolmogorov [13], Hejting [10], Kari i Hauvard [11]. Pored njih, u velikoj meri je zaslužan Per Martin Luf, koji dalje razvija teoriju tipova u *intuicionističku teoriju tipova* (engl. *intuitionistic type theory*) ili *zavisnu teoriju tipova* (engl. *dependent type theory*) [14, 15, 16, 17, 18]. Zavisnu teoriju tipova mnogi interaktivni dokazivači uzimaju kao polaznu tačku koju dalje nadograđuju.

Neki od poznatih interaktivnih dokazivača teorema uključuju: IDRIS [2], ISABELLE-HOL [21], HOL LIGHT [9], LEAN [20], LEAN4 [19], COQ [25], AGDA [22, 23] i NUPRL [7]. Većina ovih interaktivnih dokazivača teorema kao osnovu ima neku od teorije tipova. Zbog toga, tokom razvoja interaktivnih dokazivača teorema dolazi do potrebe da se teorije tipova klasifikuju. Generalnu klasifikaciju tipskih sistema dao je Berendregt (1991) kroz koncept *lambda kocke* [1].

1.2 Organizacija rada

Ostatak rada je organizovan u 3 poglavlja. Poglavlje 2 uvodi osnovne pojmove logike prvog reda, kao i pravila prirodne dedukcije. Poglavlje 3 opisuje interaktivni dokazivač teorema THEODORE, njegovu instalaciju, način korišćenja, kao i detalje implementacije. U poslednjem poglavlju 4 izvodi se zaključak.

2 Osnovi pojmovi

THEDORE interaktivni dokazivač teorema omogućava formulaciju i dokazivanje teorema u okviru logike prvog reda pravilima prirodne dedukcije. Zbog toga u ovom poglavlju uvodimo logiku provog reda, kao i pravila prirodne dedukcije.

2.1 Logika prvog reda

Simbole logike provog reda čine logički simboli, funkcijski i relacijski simboli. Svakom funkcijsom i relacijskom simbolu dodeljujemo funkciju arnosti koja kazuje koliko argumenata ima odredjena funkcija, odnosno relacija. Funkcijske simboli arnosti 0 nazivamo konstante, dok relacijske simbole arnosti 0 nazivamo promenljive. Preciznije, definišemo signaturu $\mathcal{L}$ kao:

Definicija 2.1 (Signatura)**.** Signaturu $\mathcal{L} = (\Sigma, \Pi, ar)$ čini skup funkcijskih simbola Σ, skup relacijskih simbola Π i funkcija arnosti $ar : \Sigma \cup \Pi \to \mathbb{N}$.

Sintaksu iskazne logike definišemo pomoću pojmova term, atomička formula i formula.

Definicija 2.2 (Term)**.** Skup termova signature $\mathcal{L}$ je najmanji skup koji zadovoljava:

- Promenljva je term.
- Ako je c simbol konstante signature $\mathcal{L}$, onda je c term.
- Ako su $t_1, \ldots, t_k$ termovi i f funkcijski simobl signature $\mathcal{L}$ arnosti k, onda je $f(t_1, \ldots, t_k)$ term.

Definicija 2.3 (Atomičke formule)**.** Skup atomičkih formula signature $\mathcal{L}$ je najmanji skup koji zadovoljava:

- Logičke konstante ($\top$ i $\bot$) su atomičke formule.
- Iskazno slovo (relacijski simbol arnosti 0) je atomička formula.
- Ako je p relacijski simbol arnosti k signature $\mathcal{L}$ i $t_1, \ldots, t_n$ termovi signature $\mathcal{L}$, onda je $p(t_1, \ldots, t_k)$ atomička formula.

Definicija 2.4 (Formule)**.** Skup formula je najmanji skup koji zadovoljava:

- Atomičke formule su formule.
- Ako je A formula, onda je $\neg A$ formula.
- Ako su A i B formule, onda su i $A \wedge B$, $A \vee B$, $A \implies B$, $A \iff B$ fromule.
- Ako je A formula i x promenljiva, onda su i $\forall x.A$ i $\exists x.A$ formule.

Za razliku od iskazne logike gde semantiku definišemo pomoću valuacije i interpretacije u valuaciji, semantiku logike prvog reda definišemo preko pojmova $\mathcal{L}$-strukture, valuacije, kao i interpretacije za datu $\mathcal{L}$-strukturu i valuaciju.

Definicija 2.5 ($\mathcal{L}$-strukture)**.** Neka je dat jezik $\mathcal{L}$. $\mathcal{L}$-strukturu $\mathcal{D}$ čini:

- Neprazan skup objekata D.
- Za svaki funkcijski simbol f arnosti k, njegova interpretacija $f_{\mathcal{D}} : D^k \to D$.
- Za svaki relacijski simbol p arnosti k, njegova interpretacija $p_{\mathcal{D}} \subseteq D^k$

Definicija 2.6 (Valuacija)**.** Za datu signaturu $\mathcal{L}$, $\mathcal{L}$-strukturu $\mathcal{D}$ sa domenom D i skup promenljivih V. Valuacija nad V je funkcija $v : V \to D$.

Definicija 2.7 (Vrednost terma)**.** Neka je dat jezik $\mathcal{L}$, $\mathcal{L}$-struktura $\mathcal{D}$ i valuacija v. Tada vrednost terma t obeležavamo kao $\mathcal{D}_v(t)$ i definišemo kao:

- Ako je term t promenljiva x, onda je $\mathcal{D}_v(x) = v(x)$.
- Ako je term t konstantni simbol c, onda je $\mathcal{D}_v(c) = c_{\mathcal{D}}$.
- Ako je term t funkcijski simbol arnosti k, odnosno t je oblika $f(t_1, \ldots, t_k)$, onda je $\mathcal{D}_v(f(t_1, \ldots, t_k)) = f_{\mathcal{D}}(\mathcal{D}_v(t_1), \ldots, \mathcal{D}_v(t_k))$.

Definicija 2.8 (Vrednost formule)**.** Neka je dat jezik $\mathcal{L}$, $\mathcal{L}$-struktura $\mathcal{D}$ i valuacija v. Tada vrednost formule F obeležavamo kao $\mathcal{D}_v(F)$ i definišemo kao:

- Ako je formula F konstanta $\top$, tada $\mathcal{D}_v(\top) = 1$.
- Ako je formula F konstanta $\bot$, tada $\mathcal{D}_v(\top) = 0$.
- Ako je formula F atomička formula $p(t_1, \ldots, t_k)$, tada je $\mathcal{D}_v(p(t_1, \ldots, v_k)) = 1$ akko $(\mathcal{D}_v(t_1), \ldots, \mathcal{D}_v(t_k)) \in p_{\mathcal{D}}$, gde su $\mathcal{D}_v(t_1), \ldots, \mathcal{D}_v(t_k)$ vrednosti termova $t_1, \ldots, t_k$.
- Ako je formula F oblika $\neg F'$, onda je $\mathcal{D}_v(\neg F') = 1$ akko $\mathcal{D}_v(F') = 0$.
- Ako je formula F oblika $F_1 \wedge F_2$, onda je $\mathcal{D}_v(F_1 \wedge F_2) = 1$ akko $\mathcal{D}_v(F_1) = 1$ i $\mathcal{D}_v(F_2) = 1$.
- Ako je formula F oblika $F_1 \vee F_2$, onda je $\mathcal{D}_v(F_1 \vee F_2) = 1$ akko $\mathcal{D}_v(F_1) = 1$ ili $\mathcal{D}_v(F_2) = 1$.
- Ako je formula F oblika $F_1 \implies F_2$, onda je $\mathcal{D}_v(F_1 \implies F_2) = 1$ akko $\mathcal{D}_v(F_1) = 0$ ili $\mathcal{D}_v(F_2) = 1$.
- Ako je formula F oblika $F_1 \iff F_2$, onda je $\mathcal{D}_v(F_1 \iff F_2) = 1$ akko $\mathcal{D}_v(F_1) = \mathcal{D}_v(F_2)$.
- Ako je formula F oblika $\exists x.F$, onda je $\mathcal{D}_v(\exists x.F) = 1$ akko postoji valuacija v' dobijena od v samo izmenom vrednosti promenljive x tako da $\mathcal{D}_{v'}(F) = 1$.
- Ako je formula F oblika $\forall x.F$, onda je $\mathcal{D}_v(\forall x.F) = 1$ akko za svaku valuaciju v' dobijenu od v samo izmenom vrednosti promenljive x tako da $\mathcal{D}_{v'}(F) = 1$.

2.2 Pravila prirodne dedukcije

Intuicionistička pravila prirodne dedukcije predstavljaju kolekciju pravila zaključivanja, koja možemo podeliti u dve grupe: pravila uvođenja i pravila eliminisanja. Sa druge strane, možemo ih podeliti u grupe po logičkim veznicima. Prvi put formalnu sintaksu uvode Gencen i Jaskovski (1934) [8, 12].

Pravila uvodjenja i elimenacije konjunkcije:

$$[\wedge_I]\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$[\wedge_{E1}]\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad [\wedge_{E2}]\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

Pravila uvodjenja i eleminacije disjunkcije:

$$[\vee_{I1}]\ \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad [\vee_{I1}]\ \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$[\vee_E]\ \frac{\Gamma \vdash A \vee B \qquad \Gamma,\, A \vdash X \qquad \Gamma,\, B \vdash X}{\Gamma \vdash X}$$

Pravila uvodjenja i eliminacije implikacije:

$$[\Longrightarrow_I]\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Longrightarrow B} \qquad [\Longrightarrow_E]\ \frac{\Gamma \vdash A \qquad \Gamma \vdash A \Longrightarrow B}{\Gamma \vdash B}$$

Pravila uvodjenja i eliminacije negacije:

$$[\neg_I]\ \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \qquad [\neg_E]\ \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}$$

Pravila uvodjenja i eliminacije logičkih konstanti:

$$[\top_I]\ \frac{\Gamma}{\Gamma \vdash \top} \qquad [\bot_E]\ \frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

Pravila uvodjenja i eliminacije univerzalnog kvantifikatora:

$$[\forall_I]\ \frac{\Gamma \vdash A[x \to y]}{\Gamma \vdash \forall x.A} \qquad [\forall_E]\ \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x \to t]}$$

Pravila uvodjenja i eliminacije egzistencijalnog kvantifikatora:

$$[\exists_I]\ \frac{\Gamma \vdash A[x \to t]}{\Gamma \vdash \exists x.A} \qquad [\exists_E]\ \frac{\Gamma \vdash \exists x.A \qquad \Gamma, A[x \to y] \vdash B}{\Gamma \vdash B}$$

Pored intuicionističkih pravila postoje i tri klasična pravila:

$$[\text{LEM}]\ \frac{}{\Gamma \vdash A \vee \neg A}$$
$$[\text{double neg}]\ \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$
$$[\text{contr}]\ \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}$$

3 Theodore

Interaktivni dokazivač teorema Theodore je javno dostupan i može se preuzeti na sledećoj adressi: `https://github.com/andrija-urosevic/theodore`. Za pokretanje interaktivno dokazivača teorema Theodore potrebno je instalirati kompajler za funkcionalni programski jezik Haskell (GHC — Glasgow Haskell Compiler), kao i njegovo interaktivno okruženje GHCi. Nakon instaliranja kompajlera GHC i interaktivnog okruženja GHCi, Theodore se pokreće tako što se u interaktivnom orkuženju GHCi učita fajl `NatDed.hs`, koji se nalazi u direktorijumu `src`.

3.1 Logika prvog reda unutar Theodore

Tip svih termova možemo definisati kao tip svih promenljivih koje su okarakterisane imenom, kao i tip svih funkcija koje su okarakterisane imenom i listom termova. Konstante su funkcije arnosti 0, odnostno funkcije okarakterisane imenom i praznom listom. To u Haskell-u zapisujemo kao:

```
data Term
    = Var String
    | Fun String [Term]
    deriving (Eq, Ord)

mkConstTerm :: String -> Term
mkConstTerm c = Fun c []
```

Na primer, term $(p + 0) + (p * q)$ možemo definisati kao:

```
bitArithTerm :: Term
bitArithTerm =
    Fun "+"
    [ Fun "+" [ Var "p", mkConstTerm "0" ]
    , Fun "*" [ Var "p" , Var "q" ] ]
```

Atomičke formule logike provog reda možemo definisati kao:

```
Rel String [Term]
```

Jasno je da su logičke promenljive relacije arnosti 0, odnosno relacije okarakterisane imenom i praznom listom. Zbog toga, tip svih formula logike provog reda definišemo kao:

```
data Formula
     = Top
     | Bot
     | Rel String [Term]
     | Neg Formula
     | Conj Formula Formula
     | Disj Formula Formula
     | Impl Formula Formula
     | Eqiv Formula Formula
     | Alls String Formula
     | Exis String Formula
     deriving (Eq, Ord)

mkVar :: String -> Fromula
mkVar p = Rel p []
```

Na primer, formulu $\forall x.\mathsf{even}(x) \wedge \mathsf{lt}(x, 3) \implies \mathsf{eq}(x, 0)$ možemo zapisati kao:

```
babyFormula :: Formula
babyFormula =
    Alls "x" (Impl
        (Conj
            (Rel "even" [Var "x"])
            (Rel "lt" [Var "x", mkConstTerm "three"])
        )
        (Rel "eq" [Var "x", mkConstTerm "zero"])
    )
```

Da bi definisali $\mathcal{L}$-strukturu, potrebno je prvo uvesti sledeće tipske sinonime:

```
type Assigment a    = Map.Map String a
type FnInterp a     = Assigment ([a] -> a)
type RelInterp a    = Assigment ([a] -> Bool)
```

Dodela Assigmenta predstavlja mapu koja ključ tipa String slika u element tipa a. Dalje, funkcijskim simbolima dodeljujemo funkciju [a] -> a, a relacijskim simbolima dodeljujemo funkciju [a] -> Bool.

Podesetimo se da $\mathcal{L}$-strukturu čini domen D, interpretaciju funkcijskih simbola $f_{\mathcal{D}} : D^k \to D$, i interpretaciju relacijskih simbola $p_{\mathcal{D}} \subseteq D^k$. Tip $\mathcal{L}$-strukture ćemo nazivati Model, a domen D univerzum, koga karakteriše lista elemenata. U HESKELL-u to zapisujemo kao:

```
data Model a =
    Model { univ :: [a]
          , fn :: FnInterp a
          , rel :: RelInterp a }
```

Primer $\mathcal{L}$-strukture aritmetike možemo definisati tako što za domen/univerzum uzmemo listu cifara od $[0, \ldots, 9]$. Konstante $0, \ldots, 9$ interpretiraćemo na uobičajeni način. Funkcije sabiranja i množenja, takođe, interpretiramo na uobičajeni način, ali po modulu 10 kako bi ispoštovali domen. Relacijske simbole jednako (eq), manje (lt), i neparno (even), takođe, interpretiramo na uobičajeni način. To u HASKELL-u zapisujemo kao:

```
babyArithModel :: Model Int
babyArithModel = Model
    [0..9]
    ( Map.fromList
    [ ("+",     \args -> args !! 0 + args !! 1 `mod` 10)
    , ("*",     \args -> args !! 0 * args !! 1 `mod` 10)
    , ("zero",  \_ -> 0)
    , ("one",   \_ -> 1)
    , ("two",   \_ -> 2)
    , ("three", \_ -> 3)
    , ("four",  \_ -> 4)
    , ("five",  \_ -> 5)
    , ("six",   \_ -> 6)
    , ("seven", \_ -> 7)
    , ("eight", \_ -> 8)
    , ("nine",  \_ -> 9) ]
    )
    ( Map.fromList
    [ ("eq",    \args -> args !! 0 == args !! 1)
    , ("lt",    \args -> args !! 0 < args !! 1)
    , ("even",  \args -> even (args !! 0)) ]
    )
```

Primer jedne valuacije promenljivih, u kojoj promenljivoj x dodeljujemo vrednost 5, u HASKELL-u zapisujemo kao:

```
babyAssigment :: Assigment Int
babyAssigment = Map.fromList [ ("x", 5) ]
```

Trivijalna valuacija ni jednoj promenljivoj ne dodeljuje vrednosti iz domena:

```
trivAssigment :: Assigment Int
trivAssigment = Map.fromList []
```

Vrednost terma za datu interpretaciju funkcija i valuaciju u HASKELL-u definišemo kao:

```
evalTerm :: FnInterp a -> Assigment a -> Term -> a
evalTerm _  sigma (Var p)       = sigma Map.! p
evalTerm fn sigma (Fun f ts)    = fn Map.! f
                                  $ map (evalTerm fn sigma)
                                  $ ts
```

Vrednost formule za dati $\mathcal{L}$-strukturu i valuaciju u HASKELL-u definišemo kao:

```
evalFormula :: Model a -> Assigment a -> Formula -> Bool
evalFormula _     _     Top             = True
evalFormula _     _     Bot             = False
evalFormula model sigma (Rel r ts)      = (rel model) Map.! r
                                          $ map (evalTerm (fn model) sigma)
                                          $ ts
evalFormula model sigma (Neg f)         = not (evalFormula model sigma f)
evalFormula model sigma (Conj f1 f2)    = (evalFormula model sigma f1)
                                        && (evalFormula model sigma f2)
evalFormula model sigma (Disj f1 f2)    = (evalFormula model sigma f1)
                                        || (evalFormula model sigma f2)
evalFormula model sigma (Impl f1 f2)    = not (evalFormula model sigma f1)
                                        || (evalFormula model sigma f2)
evalFormula model sigma (Eqiv f1 f2)    = (not (evalFormula model sigma f1)
                                        || (evalFormula model sigma f2))
                                        && ((evalFormula model sigma f1)
                                        || not (evalFormula model sigma f2))
evalFormula model sigma (Alls x f)      = all (\val ->
                                              evalFormula
                                                  model
                                                  (Map.insert x val sigma)
                                                  f
                                          )
                                          $ univ model
evalFormula model sigma (Exis x f)      = any (\val ->
                                              evalFormula
                                                  model
                                                  (Map.insert x val sigma)
                                                  f
                                          )
                                          $ univ model
```

Sledeća izraz treba da se evaluira u False.

```
evalFormula babyArithModel trivAssigment babyFormula
```

Ukoliko uzmemo za x vrednost 1, u datoj $\mathcal{L}$-strukturi, važi da je $\mathsf{even}(x)$ kao i da $\mathsf{lt}(x, 3)$, ali ne važi $\mathsf{eq}(x, 0)$. Zbog toga, se prethodni izraz evaluira u False.

3.2 Prirodna dedukcija unutar Theodore

Da bi definisali pravila prirodne dedukcije, potrebno je definisati pojam teoreme/cilja. Pravila prirodne dedukcije će se primenjivati na cilj i tako će ga transformisati. Već je napomenuto u poglavlju 1 da je cilj predstavljen kao lista pretpostavki, zajedno sa zaključkom. Pored toga, zaključak kao i svaka od pojedinačnih pretpostavki predstavlja formulu logike prvog reda. Često imamo

potrebu da se referišemo na određenu pretpostavku iz liste pretpostavki. Zbog toga, pretpostavku karakterišemo imenom pretpostavke kao i odgovarajućom formulom logike prvog reda. To je unutar THEODORE-a definisano kao:

```
data Assumption = Assumption { name :: String
                             , formula :: Formula }

type Assumptions = [Assumption]
```

Sa druge strane cilj, se može razbiti na više podciljeva. Zbog toga, cilj definišemo kao listu podciljeva, gde svaki podcilj karakteriše lista pretpostavki i zaključak. To je unutar THEODORE-a definisano kao:

```
data Subgoal = Subgoal { assms :: Assumptions
                       , cncls :: Formula }

type Goal = [Subgoal]
```

Pretpostavimo da želimo da pokažemo teoremu $A \wedge B \implies B \wedge A$. Prvo definišemo odgovarajuću formulu:

```
fConjCommute = Impl
      (Conj (mkVar "A") (mkVar "B"))
      (Conj (mkVar "B") (mkVar "A"))
```

Dalje, definišemo odgovarajući cilj koji želimo da pokažemo:

```
thmConjCommute = [ Subgoal [] fConjCommute ]
```

Cilj thmConjCommute ima praznu listu pretpostavki, a za zaključak ima formulu fConjCommute. Kada ispišemo cilj thmConjCommute dobijamo:

```
Goal (1 subgoals):

1. subgoal
⊢ ((A ∧ B) → (B ∧ C))
```

Odgovarajući dokaz započinjemo uvođenjem implikacije. Uvođenje implikacije unutar THEODORE-a je definisano kao:

```
intro :: String -> Goal -> Goal
intro assmName []       = error "Nothing to apply intro to!"
intro assmName (g : gs) = case (cncls g) of
   Impl f1 f2 -> Subgoal [Assumption assmName f1] f2 : gs
   _          -> error "Invalid rule!"
```

Kompletna lista pravila prirodne dedukcije kao i odgovarajućih funkcija se nalazi u tabeli 1.

Kada primenimo intro "h" nad ciljem thmConjCommute, dobijamo:

```
Goal (1 subgoals):

1. subgoal
• (A ∧ B) (h)
⊢ (B ∧ C)
```

Dalje, možemo da primenimo eliminaciju konjunkcije, odnosno funkciju split "h" nad prehodno dobijenim ciljem. Dobijamo:

Prirodna dedukcija	THEODORE funkcija
exact	exact :: String $->$ Goal $->$ Goal
$\Longrightarrow_I$	intro :: String $->$ Goal $->$ Goal
$\wedge_I$	tear :: Goal $->$ Goal
$\vee_{I1}$	left :: Goal $->$ Goal
$\vee_{I2}$	right :: Goal $->$ Goal
$\Longleftrightarrow_I$	iff :: String $->$ Goal $->$ Goal
$\neg_I$	false :: String $->$ Goal $->$ Goal
$\wedge_E$	split :: String $->$ Goal $->$ Goal
$\vee_E$	cases :: String $->$ Goal $->$ Goal
$\Longrightarrow_E$	apply :: String $->$ Goal $->$ Goal
$\Longleftrightarrow_E$	equiv :: String $->$ Goal $->$ Goal
$\neg_E$	turn :: String $->$ Goal $->$ Goal
$\forall_E$	nedostaje
$\forall_I$	nedostaje
$\exists_E$	nedostaje
$\exists_I$	nedostaje

Tabela 1: Prabila prirodne dedukcije i njihove odgovarajuće funkcije unutar interaktivnog dokazivača THEODORE.

```
Goal (1 subgoals):

1. subgoal
• A (h1)
• B (h2)
⊢ (B ∧ A)
```

Nakon toga možemo primeniti uvođenje konjunkcije, odnosno funkciju tear nad prethodno dobijenim ciljem. Tada dobijamo 2 podcilja:

```
Goal (2 subgoals):

1. subgoal
• A (h1)
• B (h2)
⊢ B

2. subgoal
• A (h1)
• B (h2)
⊢ A
```

Prvi podcilj ne treba dalje dokazivati pa ga zatvaramo funkcijom exact "h2". Slično, drugi podcilj ne treba dalje dokazivati pa ga zatvaramo funkcijom exact "h1". Na kraju, dobijamo sledeću poruku koja nam kaže da smo završili dokaz:

```
Nothing to prove!
```

Celokupan dokaz možemo predstaviti i sačuvati kao kompoziciju funkcija, odnosno:

proofConjCommute = exact "h1" . exact "h2" . tear . **split** "h" . intro "h"

4 Zaključak

Interaktivni dokazivač teorema THEODORE omogućava dokazivanje teorema logike prvog reda, intuicionističkim pravilima prirodne dedukcije.

Literatura

[1] Henkdrik Pieter Barendregt. "Introduction to generalized type systems". U: *Journal of Functional Programming* 1.2 (1991.), str. 124–154.

[2] Edwin Brady. "Idris, a General Purpose Dependently Typed Programming Language: Design and Implementation". U: *Journal of Functional Programming* 23.5 (2013.), str. 552–593.

[3] Luitzen Egbertus Jan Brouwer. "On the foundations of mathematics". U: *Collected works* 1 (1907.), str. 11–101.

[4] Nicolaas Govert de Bruijn. "AUTOMATH, a Language for Mathematics". U: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983., str. 159–200.

[5] Alonzo Church. "A formulation of the simple theory of types". U: *The journal of symbolic logic* 5.2 (1940.), str. 56–68.

[6] Alonzo Church. *The calculi of lambda-conversion*. 6. Princeton University Press, 1985.

[7] Robert L. Constable i dr. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.

[8] Gerhard Gentzen. "Untersuchungen über das logische schließen. I." U: *Mathematische zeitschrift* 35 (1935.).

[9] John Harrison. "HOL light: An overview". U: *International Conference on Theorem Proving in Higher Order Logics*. Springer. 2009., str. 60–66.

[10] Arend Heyting. *Intuitionism: an introduction*. Sv. 41. Elsevier, 1966.

[11] William Alvin Howard. "The Formulae-as-Types Notion of Construction". U: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[12] Stanisław Jaśkowski. "On the rules of suppositions in formal logic". U: (1934.).

[13] Andrej Kolmogoroff. "Zur deutung der intuitionistischen logik". U: *Mathematische Zeitschrift* 35.1 (1932.), str. 58–65.

[14] Per Martin-Löf. "An intuitionistic theory of types". U: *Twenty-five years of constructive type theory* 36 (1998.), str. 127–172.

[15] Per Martin-Löf. "An Intuitionistic Theory of Types: Predicative Part". U: Studies in Logic and the Foundations of Mathematics 80 (1975.), str. 73–118.

[16] Per Martin-Löf. "Constructive Mathematics and Computer Programming". U: Studies in Logic and the Foundations of Mathematics 104 (1982.), str. 153–175.

[17] Per Martin-Löf. "Philosophical aspects of intuitionistic type theory". U: *Unpublished notes by M. Wijers from lectures given at the Faculteit der Wijsbegeerte, Rijksuniversiteit Leiden* (1993.).

[18] Per Martin-Löf i Giovanni Sambin. *Intuitionistic type theory*. Sv. 9. Bibliopolis Naples, 1984.

[19] Leonardo de Moura i Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*. Microsoft Research i Karlsruhe Institute of Technology, 2021. URL: `https://lean-lang.org`.

[20] Leonardo de Moura i dr. *The Lean Theorem Prover*. Microsoft Research i Carnegie Mellon University, 2015. URL: `https://lean-lang.org`.

[21] Tobias Nipkow, Lawrence C. Paulson i Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Technische Universität München. Springer-Verlag, 2002. URL: `https://isabelle.in.tum.de`.

[22] Ulf Norell. "Dependently typed programming in Agda". U: *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*. Association for Computing Machinery, 2009., str. 230–266.

[23] Ulf Norell. "Towards a practical programming language based on dependent type theory". Doktorska teza. Chalmers University of Technology, 2007.

[24] Bertrand Russell. "Mathematical logic as based on the theory of types". U: *American journal of mathematics* 30.3 (1908.), str. 222–262.

[25] The Coq Development Team. *The Coq Proof Assistant*. Inria. URL: `https://coq.inria.fr`.