

*Wir müssen wissen — wir werden wissen!*

# Sadržaj

<b>Uvod</b>	<b>3</b>
<b>1. Coq</b>	<b>4</b>
1.1. Što je Coq?	4
1.2. Programiranje u Coqu	6
1.3. Hijerarhija tipova	9
1.4. Propozicije i tipovi, dokazi i termi	10
1.5. Ograničenja tipskog sustava	12
<b>2. Logika prvog reda s induktivnim definicijama</b>	<b>15</b>
2.1. Sintaksa	16
2.2. Semantika	23
2.3. Standardni modeli	26
<b>3. Sistem sekvenata s induktivnim definicijama</b>	<b>31</b>
3.1. Strukturalna pravila	33
3.2. Propozicijska pravila	34
3.3. Pravila za kvantifikatore	35
3.4. Pravila indukcije	36
3.5. Produkcijaska pravila	39
3.6. Dokaz i teorem	39
3.7. Adekvatnost	41
<b>4. Ciklički dokazi</b>	<b>43</b>
4.1. Primjer: lijeve liste	43
4.2. Primjer: $CLKID^\omega$	45

<b>Zaključak</b>	<b>48</b>
<b>Literatura</b>	<b>49</b>
<b>Sažetak</b>	<b>51</b>
<b>Abstract</b>	<b>52</b>

# Uvod

Coq je interaktivni dokazivač teorema u kojem korisnik iskazuje matematičke tvrdnje tipovima te ih dokazuje programiranjem. Teorija tipova koju Coq implementira temelji se na računu induktivnih konstrukcija koji pak je proširenje  $\lambda$ -računa tipovima. Iskazivanje tvrdnji moguće je zahvaljujući sorti Prop te zavisnim tipovima. Zahvaljujući jeziku taktika moguće je i automatizirano dokazivanje nekih tvrdnji, a osim za dokazivanje, Coq se može koristiti i za pisanje dokazano točnih programa.

Središnji pojam ovog rada je logika prvog reda s induktivnim definicijama  $FOL_{ID}$ , koju je prvi uveo James Brotherston, a u kojoj se neki predikati definiraju na induktivni način produkcijama. Zbog induktivnih predikata nužno je sa semantičkog stajališta promatrati posebne vrste struktura u kojima interpretacije induktivnih predikata imaju smisla s obzirom na njihove definicije. Sa sintaksnog stajališta promatra se Gentzenov sistem sekvenata proširen pravilima za induktivne predikate.

Ovaj rad sastoji se od četiri dijela. U prvom dijelu prikazujemo osnovne Coqa kao programskog jezika te osnove njegove teorijske pozadine, prvenstveno hijerarhije tipova i Curry–Howardove korespondencije. U drugom dijelu čitatelj se upoznaje s logikom  $FOL_{ID}$  kroz pojmove formula, struktura te standardnih modela. U trećem dijelu definiramo dokazni sustav  $LKID$  za logiku  $FOL_{ID}$  te dokazujemo neka njegova svojstva. Sve definicije i tvrdnje u drugom i trećem dijelu popraćene su svojim formalizacijama u Coqu. Konačno, u četvrtom dijelu ilustriramo pojam cikličkog dokaza primjerima u Coqu i u dokaznom sustavu  $CLKID^{\omega}$ . Cijeli rad prožet je ilustrativnim primjerima, kako neformalnih definicija i iskaza, tako i njihovih formalnih reprezentacija u Coqu.

# 1. Coq

U ovom poglavlju dajemo pregled visoke razine programskog sustava Coq. Prvo ćemo objasniti što je uopće Coq, u kojem je kontekstu nastao, i od kojih komponenti se sastoji. Zatim ćemo dati kratak pregled programiranja u Coqu, nakon čega ćemo se baviti naprednijim konceptima i spomenuti neka ograničenja. Za širi opseg gradiva, čitatelja upućujemo na knjige *Coq'Art* [1], *Software Foundations* [2, 3, 4] i *Certified Programming with Dependent Types* [5] te na službenu dokumentaciju [6].

## 1.1. Što je Coq?

Alat za dokazivanje Coq<sup>1</sup>, punog naziva *The Coq Proof Assistant*, programski je sustav pomoću kojeg korisnici mogu dokazivati matematičke tvrdnje, a može se koristiti i kao funkcijski programski jezik sa zavisnim tipovima. Alat se temelji na  $\lambda$ -računu i teoriji tipova, a prva je inačica implementirana 1984. godine [6]. Ovaj rad koristi inačicu 8.18 iz rujna 2023. godine.

Program Coq može se pokrenuti u interaktivnom ili u skupnom načinu rada. Interaktivni način rada pokreće se naredbom `coqtop`, a korisniku omogućuje rad u ljusci sličnoj ljuskama `bash` i `python`. Interaktivna ljuska (također poznata pod imenom *toplevel*) služi unosu definicija te iskazivanju i dokazivanju tvrdnji. Skupni način rada pokreće se naredbom `coqc`, a korisniku omogućuje provjeru tipova i prevođenje izvornih datoteka u strojno čitljive formate. Kod formaliziranja i dokazivanja, korisnik će najčešće koristiti interaktivni način rada, po mogućnosti kroz neku od dostupnih razvojnih okolina.<sup>2</sup>

Kao programski jezik, Coq se sastoji od više podjezika različitih namjena, od kojih

---

<sup>1</sup><https://coq.inria.fr/>

<sup>2</sup>Autor rada koristio je paket *Proof General* za uređivač teksta *Emacs*. Druge često korištene okoline su *VsCoq* i *CoqIDE*.

spominjemo *Vernacular*, *Gallinu* i *Ltac*.

**Vernacular** („govorni jezik”) je jezik naredbi kojima korisnik komunicira sa sustavom (i u interaktivnom i u skupnom načinu rada); svaka Coqova skripta (datoteka s nastavkom `.v`) je niz naredbi. Često korištene naredbe su `Check`, `Definition`, `Inductive`, `Fixpoint` i `Lemma`. Pomoću naredbi za iskazivanje tvrdnji, kao što je `Lemma`, Coq ulazi u način dokazivanja (*proof mode*).

**Gallina** je Coqov strogo statički tipiziran specifikacijski jezik. Dokazi svih tvrdnji predstavljeni su interno kao programi u Gallini. Kako se glavnina programiranja u Coqu svodi upravo na programiranje u Gallini, posvećujemo joj idući odjeljak.

**Ltac** je Coqov netipizirani jezik za definiciju i korištenje taktika. Taktike su pomoćne naredbe kojima se u načinu dokazivanja konstruira dokaz. Može se reći da je Ltac jezik za metaprogramiranje Galline. Primjeri taktika su `intros`, `destruct`, `apply` i `rewrite`.

Pogledajmo primjer za ilustraciju odnosa ta tri jezika.

```
1 Lemma example_lemma: 1 + 1 = 2.  
2 Proof.  
3   cbn. reflexivity.  
4 Qed.
```

Ključne riječi `Lemma`, `Proof` i `Qed` dio su Vernaculara, izraz `example_lemma: 1+1=2` dio je Galline, a pomoćne naredbe `cbn` i `reflexivity` dio su Ltaca.

Jezgra je programskog sustava Coq algoritam za provjeru tipova (*type checking*) — svaka tvrdnja koja se dokazuje iskazana je tipovima. Ostatak sustava u načelu služi za knjigovodstvo i poboljšanje korisničkog iskustva.<sup>3</sup> Nužno je da jezgra sustava bude relativno mala kako bismo se mogli uvjeriti u njenu točnost. U suprotnom, možemo li biti sigurni da su naše dokazane tvrdnje doista istinite?

Prve inačice Coqa implementirale su samo račun konstrukcija [7] koji je proširenje  $\lambda$ -računa polimorfnim i zavisnim tipovima te tipskim konstruktorima. Kasnije je dodana podrška za induktivno i koinduktivno definirane tipove [8, 9], a danas se može reći da Coq implementira polimorfni kumulativni račun induktivnih konstrukcija [10]. Coq se, osim kao dokazivač teorema, može koristiti i za programiranje sa zavisnim tipovima. U

---

<sup>3</sup>I jezgra i ostatak sustava implementirani su u OCamlu.

toj sferi još su istaknuti jezici Agda<sup>4</sup>, Idris<sup>5</sup> i Lean<sup>6</sup>. Coq se između njih ističe po usmjerenosti prema dokazivanju, posebno po korištenju taktika (jezik Ltac) i nepredikativnoj sorti Prop (o kojoj će kasnije biti riječi). Još jedna prednost Coqa je mehanizam *ekstrakcije* pomoću kojeg korisnik može proizvoljnu funkciju prevesti u jezik niže razine apstrakcije.<sup>7</sup> Mehanizam ekstrakcije nije dokazano točan, no poželjno je da izvorne funkcije budu ekvivalentne ekstrahiranim pa se radi na verifikaciji ekstrakcije [10].

## 1.2. Programiranje u Coqu

Gallina je funkcijski programski jezik, što znači da su funkcije prvoklasni objekti — one mogu biti argumenti i povratne vrijednosti drugih funkcija. Dodatno, iteracija se ostvaruje rekurzijom te ne postoje tradicionalne varijable, već se koriste nepromjenjiva (*immutable*) imena. Za uvod u funkcijsko programiranje, čitatelja upućujemo na knjigu *Programming in Haskell* [11]. Primjeri koje ćemo vidjeti u ostatku ovog odjeljka oslanjanju se na tipove i funkcije definirane u Coqovoj standardnoj knjižnici.<sup>8</sup>

Gallina je strogo statički tipiziran jezik, što znači da se svakom termu prilikom prevođenja dodjeljuje tip<sup>9</sup>. Naredbom `Check` možemo provjeriti tip nekog terma ili doznati da termu Coq ne može dodijeliti tip. Dalje u radu pod „term” mislimo na dobro formirane terme, odnosno na one kojima Coq može dodijeliti tip. Kažemo da je term *stanovnik* tipa koji mu je dodijeljen. Za tip kažemo da je *nastanjen*, odnosno *nenastanjen*, ako postoji, odnosno ne postoji, stanovnik tog tipa. Kako su u Coqu i tipovi termi, radi razumljivosti i zvučnosti umjesto „tip tipa” kažemo „sorta tipa”.

Kao i u ostalim jezicima, kod programiranja u Coqu korisnik se oslanja na dostupne primitivne izraze, od kojih su najvažniji:

- `forall` za konstrukciju funkcijskih tipova i zavisnih produkata,
- `match` za rad sa stanovnicima induktivnih tipova te
- `fun`, `fix` i `cofix` za definiciju funkcija.

Naredbom `Inductive` definira se *induktivni* tip te se automatski za njega generiraju

---

<sup>4</sup><https://wiki.portal.chalmers.se/agda/>

<sup>5</sup><https://www.idris-lang.org/>

<sup>6</sup><https://lean-lang.org/>

<sup>7</sup>Trenutno su podržani Haskell, OCaml i Scheme.

<sup>8</sup><https://coq.inria.fr/library/>

<sup>9</sup>Tipovi su kolekcije objekata na kojima je moguće provoditi srodne operacije.

principi *indukcije* i *rekurzije*.

```
1 Inductive nat : Set :=  
2 | 0 : nat  
3 | S : nat -> nat.
```

Ovim kodom definirali smo tri terma:

- `nat` (tip prirodnih brojeva) je term sorte `Set`,
- `0` (broj nula) je term tipa `nat` i
- `S` (funkcija sljedbenika) je term tipa `nat → nat`.

Za term `nat` kažemo da je konstruktor tipa (*type constructor*), a za terme `0` i `S` kažemo da su konstruktori objekata (*object constructors*).

Jedna od osnovnih naredbi za imenovanje novih terma je naredba `Definition`.

```
1 Definition negb (b : bool) : bool :=  
2   match b with  
3   | false => true  
4   | true  => false  
5   end.
```

U gornjem kodu definirana je funkcija `negb` čiji se argument `b` tipa `bool` destrukturira te se vraća njegova negacija, također tipa `bool`. Važno je napomenuti da izrazi koji počinju s `match t` ((*pattern matching*)), gdje je `t` stanovnik tipa `T`, moraju imati po jednu granu za svaki konstruktor tipa `T`.<sup>10</sup> U ovom su primjeru konstante `false` i `true` jedini konstruktori tipa `bool`. Funkcija `negb` je tipa `bool → bool`.

```
1 Definition mult_zero_r : Prop := forall (n : nat), n * 0 = 0.
```

Ovdje je definirana propozicija (tip) imena `mult_zero_r` kao tvrdnja univerzalno kvantificirana po prirodnim brojevima.

Rekurzija nad induktivnim tipovima može se ostvariti naredbom `Fixpoint`, koja u pozadini koristi naredbu `Definition` te izraz `fix`.

---

<sup>10</sup>U sprezi s uvjetom strukturalne rekurzije, ovime je osigurana totalnost svake funkcije.



```

1 Fixpoint plus (n m : nat) {struct n} : nat :=
2   (* Definition plus := fix plus (n m : nat) {struct n} : nat := *)
3   match n with
4   | 0 => m
5   | S n' => S (plus n' m)
6   end.

```

U ovom primjeru definirana je funkcija `plus` koja prima dva argumenta tipa `nat`. Funkcija je rekurzivna s obzirom na prvi argument što je vidljivo iz oznake `{struct n}`. Napominjemo da su induktivni tipovi dobro utemeljeni, to jest svaki term induktivnog tipa je konačan.

Osim induktivnih, u Coqu postoje i koinduktivni tipovi, koji nisu dobro utemeljeni, zbog čega za njih nije moguće definirati principe indukcije i rekurzije. Umjesto rekurzije, koinduktivni tipovi koriste se u korekurzivnim funkcijama. Standardni primjer koinduktivnog tipa je beskonačna lista.

```

1 Set Primitive Projections.
2 CoInductive Stream (A : Type) :=
3   Cons { hd : A; tl : Stream A; }.

```

Ovime smo definirali familiju tipova `Stream` indeksiranu tipskom varijablom `A`. Svaki `Stream` ima glavu i rep koji je također `Stream`.

Stanovnici koinduktivnih tipova konstruiraju se korekurzivnim funkcijama naredbom `CoFixpoint`, koja u pozadini koristi naredbu `Definition` te izraz `cofix`.

```

1 CoFixpoint from (n : nat) : Stream nat := Cons _ n (from (S n)).
2   (* Definition from := cofix from (n : nat) := Cons _ n (from (S n)) *)

```

Ovime je definirana funkcija `from` koja za ulazni argument `n` vraća niz prirodnih brojeva od `n` na dalje.

Razlika induktivnih i koinduktivnih tipova može se sumirati epigramom:

„Induktivni tipovi su domene rekurzivnih funkcija,  
koinduktivni tipovi su kodomene korekurzivnih funkcija”.

Time se želi reći da se termi induktivnih tipova destruktuiraju u rekurzivnim funkcijama, dok se termi koinduktivnih tipova konstruiraju u korekurzivnim funkcijama.

### 1.3. Hijerarhija tipova

U usporedbi s tradicionalnim programskim jezicima, Coqov tipski sustav je ekspresivniji jer dopušta tipove koji mogu ovisiti o termima. Takvi tipovi se u Coqu konstruiraju izrazom oblika `forall`. Primjer jednog takvog tipa je „lista duljine  $n$ ”, gdje je  $n$  neki prirodan broj. Njegovi su stanovnici  $n$ -torke, a on ovisi o stanovniku drugog tipa (u ovom slučaju, o  $n$  tipa `nat`).

Spomenuli smo da su i tipovi termi te im se može dodijeliti sorta. Postoji li najveća sorta, odnosno postoji li tip `Type` čiji su stanovnici svi tipovi? Prisjetimo se, svaki term ima svoj tip. Kada bi takav `Type` postojao, tada bi vrijedilo `Type : Type`, što može dovesti do paradoksa samoreferenciranja.<sup>11</sup> Takav dokazivač teorema bio bi inkonzistentan te bismo njime mogli dokazati kontradikciju, čime dokazivač efektivno gubi svoju svrhu. Umjesto jedne „velike” sorte `Type`, u Coqu postoji rastući niz sorti `Typen` za sve prirodne brojeve  $n$ , takav da vrijedi `Typen : Typem` kad god vrijedi  $n < m$ . Ilustracija ove **kumulativne hijerarhije tipova** prikazana je na slici 1.1. Dvije najvažnije sorte u Coqu su `Set` i `Prop`.

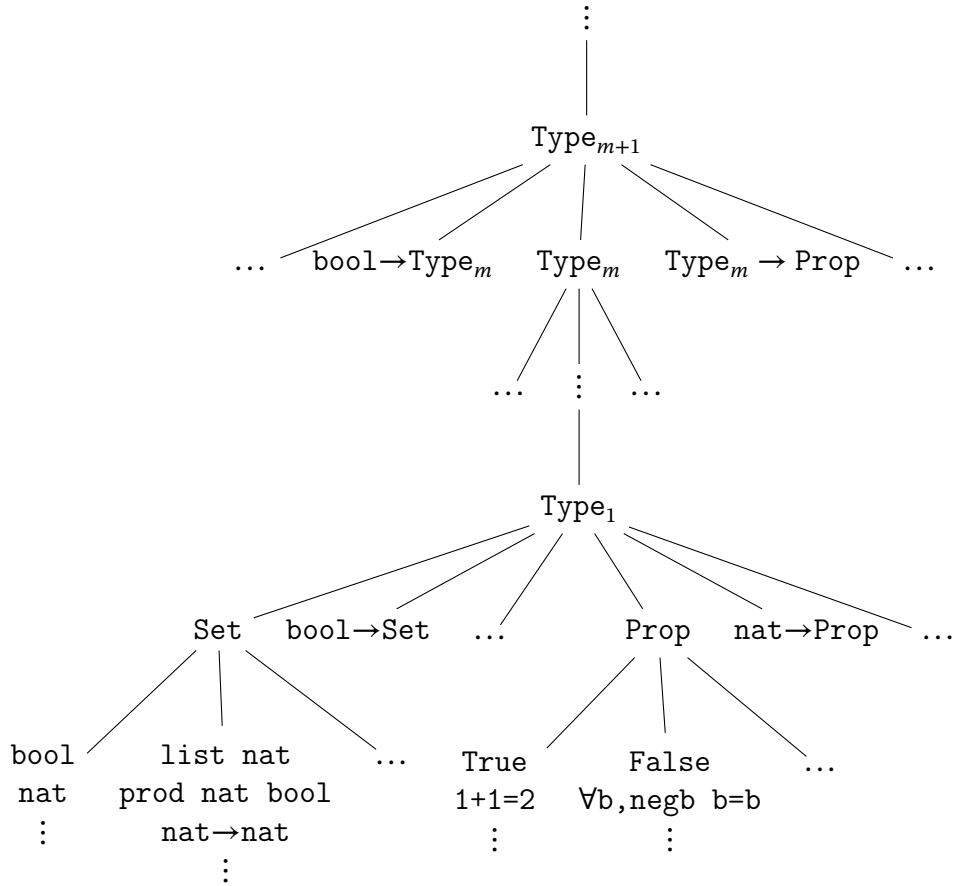
Naziv `Set` sinonim je za sortu `Type0`, a njene stanovnike zovemo **malim tipovima**. Primjerice, tipovi `nat` i `bool` su mali tipovi. Dodatno, tipovi funkcija koje primaju i vraćaju male tipove su također mali tipovi. Također su i produkti, sume, liste i stabla malih tipova ponovo mali tipovi. Intuitivno se može reći da su mali oni tipovi s čijim se stanovnicima može efektivno računati. Stanovnike malih tipova nazivamo **programima**.

Stanovnici sorte `Prop` su **propozicije** (izjave). Za razliku od programa, s propozicijama ne možemo efektivno računati, ali ih možemo dokazivati. Stanovnici propozicija su njihovi **dokazi**.

Za dokazivače teorema, poželjna je mogućnost definicije predikata (propozicija) nad proizvoljnim tipovima. Zbog toga u Coqu prilikom definicije terma sorte `Prop` možemo kvantificirati po proizvoljno velikim tipovima, što uključuje i sortu `Prop`. Tako je ovdje `not` predikat nad sortom `Prop`.

1 **Definition** `not (P : Prop) := forall (Q : Prop), P -> Q.`

<sup>11</sup>U naivnoj logici to je Epimenidov paradoks („Ova je rečenica lažna.”), a u naivnoj teoriji skupova to je Russellov paradoks, u naivnoj teoriji tipova to je Girardov paradoks.



**Slika 1.1.** Kumulativna hijerarhija tipova

Valja primijetiti da kod definicije predikata `not` kvantificiramo po `Prop`, no za svaku propoziciju `P` je term `not P` također tipa `Prop`, pa domena funkcije `not P` uključuje sam term `not P`. Ovaj stil kvantifikacije omogućuje nam definiciju proizvoljnih propozicijskih veznika.<sup>12</sup> Kažemo da je sorta `Prop` **nepredikativna**. S druge strane, sorta `Set` je **predikativna**, to jest *ne dopušta* kvantifikaciju po `Set` i ostalim proizvoljno velikim tipovima. Kada bi sorta `Prop` bila predikativna, ne bismo mogli definirati predikat `not: Prop → Prop`.

## 1.4. Propozicije i tipovi, dokazi i termi

U decimalnom zapisu broja  $\pi$ , barem jedna znamenka pojavljuje se beskonačno mnogo puta. Doista, kada bi se svaka znamenka pojavljivala samo konačno mnogo puta, broj  $\pi$  bio bi racionalan. Međutim, nije jasno *koja* znamenka ima to svojstvo. Možda ih ima više. Štoviše, vjerujemo da su *sve* znamenke takve. Da bismo odgovorili na to pitanje,

<sup>12</sup>U praksi su veznici definirani induktivno.

morali bismo prebrojiti sve znamenke broja  $\pi$ , što nije moguće u konačno mnogo koraka.

Sličnim pitanjima bavili su se logičari dvadesetog stoljeća. *Klasični* logičari bi gornju tvrdnju smjesta prihvatili, dok bi *konstruktivisti* tražili konkretnu znamenku. Između ostalog, ovakva razmatranja rezultirala su fundamentalnim uvidom u povezanost programiranja i dokazivanja. Naime, želimo li dokazati konjunkciju, dovoljno je zasebno dokazati njene konjunkte. S druge strane, želimo li konstruirati par objekata, dovoljno je zapakirati prvi i drugi objekt u konstruktor para. Na sličan način, želimo li dokazati implikaciju, dovoljno je pretpostaviti njen antecedent te pomoću njega dokazati konzekvens. Ako pak želimo konstruirati funkciju, smijemo uzeti njen argument i pomoću njega konstruirati povratnu vrijednost. Dodatno, nemoguće je dokazati laž, a istina trivijalno vrijedi. S druge strane, ako induktivni tip nema konstruktore, onda je nenastanjen jer nije moguće definirati vrijednost tog tipa. Ako pak tip ima barem jedan konstruktor bez argumenata, tada postoje i njegovi stanovnici. Kroz ove primjere vidimo fenomen **Curry–Howardove korespondencije**, koju možemo sažeti epigramom:

„Propozicije su tipovi, dokazi su programi.”

Time se dokazivanje svodi na programiranje. Pogledi na dvije strane ovog „novčića” mogu se vidjeti u tablici 1.1.

Dokazivanje	Programiranje
propozicija	tip
dokaz	program
laž	prazan tip
istina	nastanjen tip
konjunkcija	produktni tip
disjunkcija	zbrojni tip
implikacija	funkcijski tip
univerzalna kvantifikacija	zavisni produkt
egzistencijalna kvantifikacija	zavisna suma

Tablica 1.1. Sličnosti dokazivanja i programiranja

Za bolju ilustraciju, prikazujemo princip matematičke indukcije u Coqu. Prisjetimo se, za proizvoljni predikat  $P$  na prirodnim brojevima, princip matematičke indukcije glasi:

$$P(0) \wedge \forall n, (P(n) \rightarrow P(n + 1)) \rightarrow \forall n, P(n).$$

Tvrđnju dokazujemo analizom broja  $n$ . Ako je  $n = 0$ , tvrdnja slijedi iz *baze* indukcije. Ako je pak  $n = n' + 1$  za neki  $n'$ , tada rekurzivno konstruiramo dokaz za  $P(n')$ , a tražena tvrdnja slijedi primjenom *koraka* indukcije na rekurzivno konstruirani dokaz.

```

1 Definition nat_ind (P : nat -> Prop)
2   (baza : P 0)
3   (korak : forall m, P m -> P (S m))
4   : forall n, P n :=
5   fix F (n : nat) : P n :=
6     match n with
7     | 0 => baza
8     | S m => korak m (F m)
9   end.

```

Za term `nat_ind` kažemo da je dokazni term (*proof term*) za tvrdnju matematičke indukcije. Princip matematičke indukcije je samo poseban slučaj **principa indukcije**, koji Coq automatski generira za svaki induktivno definiran tip pri njegovoj definiciji.

## 1.5. Ograničenja tipskog sustava

Kao što smo već vidjeli, Coqov tipski sustav je izražajniiji od tipskih sustava uobičajenih programskih jezika. Međutim, kako bi se sačuvala poželjna svojstva algoritma provjere tipova, ipak se tipski sustav mora ograničiti.

**Uvjet pozitivnosti** odnosi se na definiciju induktivnih i koinduktivnih tipova. Ovo ograničenje zabranjuje *negativne* pojave tipa kojeg definiramo u argumentima njegovih konstruktora.

```

1 Inductive Lam :=
2   | LamVar (n : nat)
3   | LamApp (M N : Lam)
4   | LamAbs (M : Lam -> Lam).

```

Pokretanje primjera iznad rezultira greškom `Non strictly positive occurrence of "Lam" in "(Lam -> Lam) -> Lam"` — drugim riječima, tip `Lam` se javlja negativno u konstrukturu `LamAbs`, odnosno kao argument funkcije koja je parametar konstruktora. Ovaj uvjet štiti korisnika od inkonzistentnosti, a za točnu definiciju pozitivnosti čitatelja upućujemo na dokumentaciju.<sup>13</sup> Uz uvjet pozitivnosti za induktivne tipove vezan je

<sup>13</sup><https://coq.inria.fr/doc/v8.18/refman/language/core/inductive.html#well-formed-inductive-definitions>

**uvjet strukturalne rekurzije.** Ovim uvjetom osigurava se totalnost rekurzivno definirane funkcije tako da se argument po kojem je funkcija rekurzivna strukturalno smanjuje u svakom koraku rekurzije (funkcija se poziva samo na pravom podtermu originalnog argumenta).

**Uvjet produktivnosti** odnosi se na definiciju korekurzivnih funkcija, a dualan je uvjetu strukturalne rekurzije. Ovaj uvjet također štiti korisnika od inkonzistentnosti, a glasi: svaki korekurzivni poziv smije se pojaviti samo kao izravni argument konstruktora koinduktivnog tipa čiji element definiramo (poziv funkcije mora stvoriti pravi nadterm originalnog poziva).<sup>14</sup> Zbog tog uvjeta, iduća definicija nije moguća.

```
1 Set Primitive Projections.
2 CoInductive NatStream := {
3     nat_hd : nat;
4     nat_tl : NatStream;
5 } .
6
7 CoFixpoint foo : NatStream := foo.
```

Greška koju sustav javlja glasi `Unguarded recursive call in "foo"`, što znači da se korekurzivni poziv `foo` ne javlja kao izravni argument konstruktora. S druge strane, definicija

```
1 CoFixpoint bar : NatStream := { | nat_hd := 0; nat_tl := bar | }.
```

je sasvim legalna.

Posljednje ograničenje koje spominjemo vezano je uz irelevantnost dokaza (*proof irrelevance*). Naime, mnogi teoremi mogu se dokazati na više načina, ali pojedini dokaz (dakle, postupak kojim smo od pretpostavki došli do konkluzije) *nije bitan*. Matematičarima su bitni samo iskaz teorema i činjenica da se teorem *može dokazati*. Sam postupak dokazivanja smatra se „implementacijskim detaljem”. Upravo zato analiza dokaza ima smisla samo kada se dokazuje, ali ne i kada se programira. U našoj terminologiji to znači da se *pattern matching* nad dokazima smije provoditi samo kod definiranja terma sorte `Prop`. U suprotnom, mogli bismo definirati programe koji ovise o *konkretnom dokazu*, umjesto o iskazanom teoremu. **Ograničenje eliminacije propozicije** nastalo je

<sup>14</sup><https://coq.inria.fr/doc/v8.18/refman/language/core/coinductive.html#co-recursive-functions-cofix>

radi omogućavanja ekstrakcije — svi termini sorte Prop se „brišu” prilikom prevođenja iz Coqovog tipskog sustava u tipske sustave niže razine apstrakcije. Kada ovog ograničenja ne bi bilo, ekstrakcija u jednostavnije jezike naprosto ne bi bila moguća, jer bi prilikom ekstrakcije bilo nužno zadržati i sve dokaze uz svu kompleksnost njihovih tipova.

## 2. Logika prvog reda s induktivnim definicijama

U ovom poglavlju predstavljamo glavne rezultate diplomskog rada: formalizaciju logike prvog reda s induktivnim definicijama  $FOL_{ID}$  te dokaznog sustava  $LKID$ , koje je prvi uveo Brotherston [12]. Definicije, leme i dokazi u ovom poglavlju preuzete su iz Brotherstonove disertacije [13]. Za općeniti uvod u logiku čitatelja upućujemo na knjigu *Matematička logika* [14].

Prvo ćemo definirati sintaksu i semantiku logike  $FOL_{ID}$ , nakon čega ćemo definirati njene standardne modele. Zatim ćemo prikazati dokazni sustav  $LKID$  te konačno dokazati dio adekvatnosti sustava  $LKID$  s obzirom na standardnu semantiku, što je ujedno i glavni rezultat ovog diplomskog rada.

Svaka definicija i lema u ovom poglavlju bit će popraćena svojom formalizacijom u Coqu. Jedan je od ciljeva diplomskog rada prikazati primjene Coqa u matematici, zbog čega leme nećemo dokazivati „na papiru”, već se dokaz svake leme može pronaći u repozitoriju rada.<sup>1</sup> Zainteresiranom čitatelju predlažemo interaktivni prolazak kroz dokaze lema u nekoj od razvojnih okolina za Coq.

Prije no što krenemo na formalizaciju, valja prokomentirati odnos matematičkog i Coqovog vokabulara što se tiče riječi „skup”. U matematici pojam „skup” može imati dva značenja; prvo se odnosi na skupove kao *domene diskursa*, dok se drugo odnosi na skupove kao *predikate*, odnosno podskupove domene diskursa. Primjerice, skup prirodnih brojeva  $\mathbb{N}$  je domena diskursa kada je riječ o svim prirodnim brojevima te zbog toga pišemo  $n \in \mathbb{N}$  umjesto  $\mathbb{N}(n)$ . S druge strane, skup svih parnih prirodnih brojeva  $E$  je podskup skupa  $\mathbb{N}$ , a može se interpretirati kao predikat parnosti na prirodnim brojevima

---

<sup>1</sup><https://github.com/mihohren/diploma-thesis>



te možemo pisati  $E(n)$  umjesto  $n \in E$ . U Coqu se skupovi kao domene diskursa formaliziraju tipovima sorte  $\text{Set}^2$ , dok se skupovi kao predikati formaliziraju funkcijama iz domene diskursa u sortu  $\text{Prop}$ . Na primjer, tip prirodnih brojeva  $\text{nat}$  je sorte  $\text{Set}$ , a predikat  $\text{Nat.Even}$  je tipa  $\text{nat} \rightarrow \text{Prop}$ .

## 2.1. Sintaksa

Kao i u svakom izlaganju logike, na početku je potrebno definirati sintaksu.

**Definicija 1.** *Signatura prvog reda s induktivnim predikatima* (kratko: *signatura*), s oznakom  $\Sigma$ , je skup simbola od kojih razlikujemo *funkcijske*, *obične predikatne* i *induktivne predikatne* simbole. Mjesnost simbola reprezentiramo funkcijom iz odgovarajućeg skupa simbola u skup  $\mathbb{N}$ , a označujemo ju s  $|f|$  za funkcijske, odnosno s  $|P|$  za predikatne simbole.

```

1 Structure signature := {
2   FuncS : Set;
3   fun_ar : FuncS -> nat;
4   PredS : Set;
5   pred_ar : PredS -> nat;
6   IndPredS : Set;
7   indpred_ar : IndPredS -> nat;
8 }.

```

**Primjer 1.** Funkcijski simboli Peanove aritmetike su redom simbol nule  $o$  (mjesnosti 0), simbol sljedbenika  $s$  (mjesnosti 1) te simboli zbroja  $+$  i umnoška  $\cdot$  (mjesnosti 2).

```

1 Inductive Func__PA :=
2 | PA_zero
3 | PA_succ
4 | PA_add
5 | PA_mult.
6 Definition fun_ar__PA (s : Func__PA) : nat :=
7   match s with
8   | PA_zero => 0
9   | PA_succ => 1
10  | PA_add  => 2
11  | PA_mult => 2
12 end.

```

<sup>2</sup>Ili tipovima sorte  $\text{Type}$  kada nam trebaju veliki tipovi, najčešće kao posljedica parametrizacije općenitih tipova.

Simbol jednakosti (= (mjesnosti 2)) je jedini obični predikatni simbol Peanove aritmetike.

```
1 Inductive Pred__PA := PA_eq.
2 Definition pred_ar__PA (s : Pred__PA) : nat := 2.
```

Definiramo induktivne predikatne simbole *Nat*, *Even* i *Odd* mjesnosti 1 koje ćemo interpretirati redom kao predikate „biti prirodan”, „biti paran” te „biti neparan”.

```
1 Inductive IndPred__PA :=
2 | PA_Nat
3 | PA_Even
4 | PA_Odd.
5 Definition indpred_ar__PA (s : IndPred__PA) : nat := 1.
```

Konačno, *proširena Peanova signatura*, s oznakom  $\Sigma_{PA}$ , je signatura za Peanovu aritmetiku prvog reda s prethodno definiranim induktivnim predikatnim simbolima.

```
1 Definition  $\Sigma\_PA$  : signature
2 := { |
3   FuncS := Func__PA;
4   fun_ar := fun_ar__PA;
5   PredS := Pred__PA;
6   pred_ar := pred_ar__PA;
7   IndPredS := IndPred__PA;
8   indpred_ar := indpred_ar__PA;
9 | }.
```

U ostatku poglavlja promatramo jednu proizvoljnu, ali fiksiranu signaturu  $\Sigma$ . Fiksiranje nekog proizvoljnog objekta je česta pojava u matematici, prvenstveno zato što fiksirane argumente ne trebamo spominjati eksplicitno u kasnijim definicijama i iskazima. Coq omogućuje fiksiranje naredbom `Context`, pod uvjetom da se korisnik nalazi u okolini `Section`.<sup>3</sup> Većina definicija i lema u ovom radu su napisane upravo unutar takvih okolina.

**Definicija 2.** *Varijabla* je prirodan broj. *Skup svih terma* konstruiramo rekurzivno na način:

1. svaka varijabla je term;
2. ako je  $f$  funkcijski simbol mjesnosti  $n$  te su  $t_1, \dots, t_n$  termi<sup>4</sup>, onda je  $f(t_1, \dots, t_n)$  također term.

<sup>3</sup><https://coq.inria.fr/doc/v8.18/refman/language/core/sections.html>

<sup>4</sup>Primijetimo, broj terma ovisi o mjesnosti funkcijskog simbola. U Coqovoj implementaciji ovog „konstruktor” možemo vidjeti da je on zavisnog tipa.

```

1 Inductive term : Set :=
2 | var_term : var -> term
3 | TFunc : forall (f : FuncS  $\Sigma$ ), vec term (fun_ar f) -> term.

```

Uobičajene prezentacije logike prvog reda za skup varijabli uzimaju proizvoljni skup  $\mathcal{V}$ , no za formalizaciju je pogodniji skup prirodnih brojeva  $\mathbb{N}$ . Umjesto eksplicitne kvantifikacije po nekoj varijabli  $v$ , implicitno ćemo kvantificirati po varijabli 0. Ovaj pristup kvantifikaciji<sup>5</sup>, nazvan „de Bruijnovo indeksiranje” [15], bitno olakšava rad sa supstitucijama. O samoj implementaciji de Bruijnovog indeksiranja više se može pročitati u knjizi *Types and Programming Languages* [16]. Za potrebe ovog rada koristili smo program *Autosubst2*<sup>6</sup> [17, 18] za automatsko generiranje tipova terma i formula te pripadajućih funkcija supstitucija i pomoćnih lema.

Princip indukcije za term potrebno je ručno definirati. Naime, induktivni tip term je ugniježđen po konstruktoru TFunc zato što se javlja oмотan oko drugog induktivnog tipa<sup>7</sup> kao argument. Za ugniježdene induktivne tipove, Coq generira neprikladne principe indukcije jer ne zna izraziti tvrdnju „predikat vrijedi za sve ugniježdene elemente”.

```

1 Lemma term_ind
2   : forall P : term  $\Sigma$  -> Prop,
3     (forall v, P (var_term v)) ->
4     (forall f args, (forall st, V.In st args -> P st) ->
5       P (TFunc f args)) ->
6     forall t : term  $\Sigma$ , P t.

```

**Definicija 3.** Skup svih varijabli terma  $t$ , s oznakom  $TV(t)$ , konstruiramo rekurzivno na način:

1.  $TV(v) := \{v\}$  za varijablu  $v$  i
2.  $TV(f(t_1, \dots, t_n)) := \bigcup_{1 \leq i \leq n} TV(t_i)$  za  $n$ -mjesni funkcijski simbol  $f$  i terme  $t_1, \dots, t_n$ .

```

1 Inductive TV : term -> var -> Prop :=
2 | TVVar : forall v, TV (var_term v) v
3 | TVFunc : forall f args v st,
4   V.In st args -> TV st v -> TV (TFunc f args) v.

```

<sup>5</sup>ili općenitije, vezivanju varijabli

<sup>6</sup><https://github.com/uds-psl/autosubst2>

<sup>7</sup>ovdje vec

**Definicija 4.** Skup svih formula konstruiramo rekurzivno na način:

1. ako je  $Q$  (obični ili induktivni) predikatni simbol mjesnosti  $n$  te su  $t_1, \dots, t_n$  termi, onda je  $Q(t_1, \dots, t_n)$  atomarna formula;
2. ako je  $\varphi$  formula, onda su  $\neg\varphi$  i  $\forall\varphi$  također formule;
3. ako su  $\varphi$  i  $\psi$  formule, onda je  $\varphi \rightarrow \psi$  također formula.

```

1 Inductive formula : Set :=
2   | FPred (P : PredS  $\Sigma$ ) : vec (term  $\Sigma$ ) (pred_ar P) -> formula
3   | FIndPred (P : IndPredS  $\Sigma$ ) : vec (term  $\Sigma$ ) (indpred_ar P) -> formula
4   | FNeg : formula -> formula
5   | FImp : formula -> formula -> formula
6   | FAll : formula -> formula.

```

Ostale veznike definiramo kao sintaksne pokrate.

```

1 Definition FAnd ( $\varphi \psi$  : formula) : formula := FNeg (FImp  $\varphi$  (FNeg  $\psi\varphi \psi$  : formula) : formula := FImp (FNeg  $\varphi$ )  $\psi$ .
3 Definition FExist ( $\varphi$  : formula) : formula := FNeg (FAll (FNeg  $\varphi$ 
```

**Primjer 2.** U proširenoj Peanovoj signaturi, svojstvo „svaki prirodan broj je paran ili neparan” možemo izraziti formulom  $\forall x, \text{Nat}(x) \rightarrow \text{Even}(x) \vee \text{Odd}(x)$ .

```

1 Definition every_nat_is_even_or_odd : formula  $\Sigma_{\text{PA}}$  :=
2   FAll
3     (FImp
4       (FIndPred PA_Nat [var_term 0])
5       (FOr
6         (FIndPred PA_Even [var_term 0])
7         (FIndPred PA_Odd [var_term 0])))).

```

Ovdje vidimo učinak de Bruijnovog indeksiranja na zapis formula; umjesto eksplicitnog navođenja varijable  $x$  kod kvantifikacije, implicitno kvantificiramo po varijabli 0.

**Definicija 5.** Skup slobodnih varijabli formule  $\varphi$ , s oznakom  $FV(\varphi)$ , konstruiramo rekurzivno na način:

1.  $FV(P(u_1, \dots, u_n)) := \bigcup_{1 \leq i \leq n} TV(u_i)$ ,
2.  $FV(\neg\varphi) := FV(\varphi)$ ,
3.  $FV(\varphi \rightarrow \psi) := FV(\varphi) \cup FV(\psi)$ ,
4.  $FV(\forall\varphi) := \{v \mid v + 1 \in FV(\varphi)\}$ .

```

1 Inductive FV : formula -> var -> Prop :=
2 | FV_Pred : forall R args v st,
3   V.In st args -> TV st v -> FV (FPred R args) v
4 | FV_IndPred : forall R args v st,
5   V.In st args -> TV st v -> FV (FIndPred R args) v
6 | FV_Imp_l : forall F G v, FV F v -> FV (FImp F G) v
7 | FV_Imp_r : forall F G v, FV G v -> FV (FImp F G) v
8 | FV_Neg : forall F v, FV F v -> FV (FNeg F) v
9 | FV_All : forall F v, FV F (S v) -> FV (FAll F) v.

```

**Definicija 6.** *Supstitucija* je svaka funkcija iz skupa  $\mathbb{N}$  u skup svih terma. Supstituciju  $\sigma$  možemo promatrati kao niz terma  $t_0, t_1, t_2, \dots$ . Tada je *pomaknuta supstitucija*, s oznakom  $t \cdot \sigma$ , supstitucija koja odgovara nizu  $t, t_0, t_1, t_2, \dots$ , za neki term  $t$ .

Domena supstitucije može se rekurzivno proširiti na skup svih terma i skup svih formula.

```

1 Fixpoint subst_term (σ : var -> term) (t : term) : term :=
2   match t with
3   | var_term v => σ v
4   | TFunc f args => TFunc f (V.map (subst_term σ) args)
5   end.

```

```

1 Fixpoint subst_formula
2   (σ : var -> term Σ) (φ : formula)
3   : formula :=
4   match φ return formula with
5   | FPred P args => FPred P (V.map (subst_term σ) args)
6   | FIndPred P args => FIndPred P (V.map (subst_term σ) args)
7   | FNeg ψ => FNeg (subst_formula σ ψ)
8   | FImp ψ ξ => FImp (subst_formula σ ψ) (subst_formula σ ξ)
9   | FAll ψ => FAll (subst_formula (up_term_term σ) ψ)
10  end.

```

Ovdje funkcija `up_term_term` brine da supstitucija  $\sigma$  mijenja samo one varijable koje nisu vezane. Pišemo  $\varphi[\sigma]$  za primjenu supstitucije  $\sigma$  na formulu  $\varphi$ . Često korištene supstitucije formula su supstitucija varijable  $x$  termom  $t$  u formuli  $\varphi$ , s oznakom  $\varphi[t/x]$ , te supstitucija svake varijable  $n$  u formuli  $\varphi$  varijablom  $n + 1$ , s oznakom  $\varphi^\uparrow$ . Iste notacije koristimo i za supstitucije na termima, listama terma i listama formula.

Konačno, potrebno je definirati sintaksu za indukciju. U Coqu su definicije induktivnih propozicija proizvoljne do na ograničenje pozitivnosti, no radi jednostavnosti u  $FOL_{ID}$  su moguće samo induktivne definicije s atomarnim formulama, a pišemo ih u stilu prirodne dedukcije:

$$\frac{Q_1 \mathbf{u}_1 \dots Q_n \mathbf{u}_n \quad P_1 \mathbf{v}_1 \dots P_m \mathbf{v}_m}{P \mathbf{t}}.$$

Ovdje su  $Q_1, \dots, Q_n$  obični predikatni simboli,  $P_1, \dots, P_m$  i  $P$  su induktivni predikatni simboli, a podebljani znakovi predstavljaju  $n$ -torke terma, gdje je  $n$  mjesnost odgovarajućeg predikata.

**Definicija 7.** Produkcija je uređena četvorka

1. liste parova običnih predikatnih simbola i  $n$ -torki terma odgovarajućih duljina,
2. liste parova induktivnih predikatnih simbola i  $n$ -torki terma odgovarajućih duljina,
3. induktivnog predikatnog simbola  $P$  mjesnosti  $m$  i
4.  $m$ -torke terma.

```

1 Record production :=
2   mkProd {
3     preds : list {P: PredS Σ & vec (term Σ) (pred_ar P)};
4     indpreds : list {P: IndPredS Σ & vec (term Σ) (indpred_ar P)};
5     indcons : IndPredS Σ;
6     indargs : vec (term Σ) (indpred_ar indcons);
7   }.

```

Prvi i drugi član četvorke zovemo *premisama*, a treći i četvrti *konkluzijom*. U ostatku rada odabiremo neki podskup skupa svih produkcija koji zovemo *skupom induktivnih definicija*, a označavamo s  $\Phi$ .

```

1 Definition IndDefSet := production -> Prop.

```

U idućim primjerima definiramo produkcije za induktivne predikatne simbole u proširenoj Peanovoj signaturi.

**Primjer 3.** Za broj kažemo da je prirodan ako je 0 ili je sljedbenik nekog prirodnog broja.

$$\frac{}{Nat(0)} (PA\_prod\_N\_zero)$$

```

1 Definition PA_prod_N_zero : production Σ_PA.
2   refine (mkProd nil nil PA_Nat _).
3   refine [TFunc PA_zero []].
4 Defined.

```

$$\frac{Nat(x)}{Nat(s(x))} (PA\_prod\_N\_succ)$$

```

1 Definition PA_prod_N_succ : production  $\Sigma\_PA$ .
2   refine (mkProd nil _ PA_Nat _).
3   - refine (cons _ nil). exists PA_Nat; refine [var_term 0].
4   - refine [TFunc PA_succ [var_term 0]].
5 Defined.

```

*Napomena.* U primjeru 3 vidimo alternativni način definiranja u Coqu. Korisnik umjesto definiranja Gallininih terma ulazi u način dokazivanja te naredbom `refine` postepeno konstruira („profinjuje”) rezultatni term, ostavljajući „rupe” \_ koje će popuniti kasnijim primjenama naredbe `refine`. Konačni rezultat možemo vidjeti naredbom `Print`.

`Print PA_prod_N_succ.`

```

PA_prod_N_succ =
{|
  preds := Datatypes.nil;
  indpreds := (PA_Nat; [var_term 0]) :: Datatypes.nil;
  indcons := PA_Nat;
  indargs := [TFunc PA_succ [var_term 0]]
|}
: production

```

**Primjer 4.** Za broj kažemo da je paran ako je 0 ili je sljedbenik nekog neparnog broja.

$$\frac{}{Even(o)} (PA\_prod\_E\_zero)$$

```

1 Definition PA_prod_E_zero : production  $\Sigma\_PA$ .
2   refine (mkProd nil nil PA_Even _).
3   refine [ TFunc PA_zero []].
4 Defined.

```

$$\frac{Odd(x)}{Even(s(x))} (PA\_prod\_E\_succ)$$

```

1 Definition PA_prod_E_succ : production  $\Sigma\_PA$ .
2   refine (mkProd nil _ PA_Even _).
3   - refine (cons _ nil). exists PA_Odd; refine [var_term 0].
4   - refine [TFunc PA_succ [var_term 0]].
5 Defined.

```

Za broj kažemo da je neparan ako je sljedbenik nekog parnog broja.

$$\frac{Even(x)}{Odd(s(x))} (PA\_prod\_O\_succ)$$

```

1 Definition PA_prod_0_succ : production  $\Sigma\_PA$ .
2   refine (mkProd nil _ PA_Odd _).
3   - refine (cons _ nil). exists PA_Even; refine [var_term 0].
4   - refine [TFunc PA_succ [var_term 0]].
5 Defined.

```

Skup prethodnih pet produkcija nazivamo skupom induktivnih definicija za proširenu Peanovu signaturu, s oznakom  $\Phi_{PA}$ .

```

1 Inductive  $\Phi\_PA$  : production  $\Sigma\_PA$  -> Prop :=
2 | ID_N_zero :  $\Phi\_PA$  PA_prod_N_zero
3 | ID_N_succ :  $\Phi\_PA$  PA_prod_N_succ
4 | ID_E_zero :  $\Phi\_PA$  PA_prod_E_zero
5 | ID_E_succ :  $\Phi\_PA$  PA_prod_E_succ
6 | ID_O_succ :  $\Phi\_PA$  PA_prod_O_succ.

```

## 2.2. Semantika

**Definicija 8.** *Struktura prvog reda* (kratko: *struktura*) je uređena četvorka skupa  $M$  koji nazivamo *nosačem* te interpretacijā funkcijskih, običnih predikatnih i induktivnih predikatnih simbola. Funkcijski simboli mjesnosti  $n$  interpretiraju se kao  $n$ -mjesne funkcije, a predikatni simboli mjesnosti  $n$  kao  $n$ -mjesne relacije na nosaču. Koristit ćemo ime nosača kao sinonim za čitavu strukturu, a interpretacije označavati s  $f^M$  odnosno  $P^M$ .

```

1 Structure structure := {
2   domain :> Set;
3   interpF (f : FuncS  $\Sigma$ ) : vec domain (fun_ar f) -> domain;
4   interpP (P : PredS  $\Sigma$ ) : vec domain (pred_ar P) -> Prop;
5   interpIP (P : IndPredS  $\Sigma$ ) : vec domain (indpred_ar P) -> Prop;
6 }.

```

**Primjer 5.** Prije no što možemo definirati strukturu za Peanovu aritmetiku s induktivnim predikatima, odnosno strukturu za signaturu  $\Sigma_{PA}$  i skup induktivnih definicija  $\Phi_{PA}$ , potrebno je definirati odgovarajuće interpretacije induktivnih predikatnih simbola u Coqu, u skladu s primjerima 3 i 4.



```

1 Inductive NAT : nat -> Prop :=
2 | NO : NAT 0
3 | NS : forall n, NAT n -> NAT (S n).
4
5 Inductive EVEN : nat -> Prop :=
6 | EO : EVEN 0
7 | ES : forall n, ODD n -> EVEN (S n)
8 with ODD : nat -> Prop :=
9 | OS : forall n, EVEN n -> ODD (S n).

```

Ovdje su predikati EVEN i ODD definirani mehanizmom simultane rekurzije.

Konačno možemo definirati strukturu  $M_{PA}$  s nosačem  $\mathbb{N}$  te uobičajenom interpretacijom funkcijskih i običnih predikatnih simbola. Induktivne predikatne simbole *Nat*, *Even* i *Odd* interpretiramo redom Coqovim predikatima NAT, EVEN i ODD.

```

1 Definition M_PA : structure Σ_PA.
2   refine (Build_structure nat _ _ _).
3   - intros f; destruct f.
4     + intros. exact 0.
5     + intros n. exact (S (V.hd n)).
6     + intros xy. exact (V.hd xy + V.hd (V.tl xy)).
7     + intros xy. exact (V.hd xy * V.hd (V.tl xy)).
8   - intros P args; destruct P.
9     exact (V.hd args = V.hd (V.tl args)).
10  - intros P args; destruct P.
11    + exact (NAT (V.hd args)).
12    + exact (EVEN (V.hd args)).
13    + exact (ODD (V.hd args)).
14 Defined.

```

*Napomena.* U primjeru 5 koristimo taktiku *exact* koja je specijalizacija taktike *refine*, a koristimo ju u zadnjem koraku konstrukcije terma. Nadalje, taktika *intros x* služi uvođenju imena *x* u kontekst, a u našem je slučaju ekvivalentna taktici *refine (fun x => \_)*. Konačno, taktika *destruct* služi analizi po slučajevima, a u pozadini koristi *pattern matching*. Kako su interpretacije funkcijskih simbola funkcije koje primaju vektore, koristimo Coqove funkcije *V.hd* i *V.tl* za dohvaćanje glave, odnosno repa vektora.

**Definicija 9.** Neka je  $M$  proizvoljna struktura. *Okolina*  $\rho$  za  $M$  je proizvoljna funkcija iz skupa varijabli (prirodnih brojeva) u nosač strukture.

```

1 Definition env := var -> M.

```

Okolina se može interpretirati kao niz  $d_0, d_1, d_2, \dots$ . Tada je *pomaknuta okolina*, s oznakom  $d \cdot \rho$ , niz  $d, d_0, d_1, d_2, \dots$  za neki  $d \in M$ . Proširenje domene okoline  $\rho$  na skup svih terma zovemo *evaluacijom*.

```

1 Fixpoint eval (ρ : env) (t : term Σ) : M :=
2   match t with
3   | var_term x => ρ x
4   | TFunc f args => interpF f (V.map (eval ρ) args)
5   end.

```

Pišemo  $t^\rho$  za evaluaciju terma  $t$  u okolini  $\rho$ . Istu notaciju koristimo i za evaluaciju  $n$ -torki terma.

**Definicija 10.** Neka je  $M$  proizvoljna struktura te  $\rho$  okolina za  $M$ . *Istinitost* formule  $\varphi$  u okolini  $\rho$ , s oznakom  $\rho \models \varphi$ , definiramo rekurzivno na način:

1. ako je  $P$  (obični ili induktivni) predikatni simbol mjesnosti  $n$  te su  $u_1, \dots, u_n$  termi, onda vrijedi  $\rho \models P(u_1, \dots, u_n)$  ako i samo ako vrijedi  $P^M(u_1^\rho, \dots, u_n^\rho)$ ,
2. vrijedi  $\rho \models \neg\varphi$  ako i samo ako ne vrijedi  $\rho \models \varphi$  (što još pišemo  $\rho \not\models \varphi$ ),
3. vrijedi  $\rho \models \varphi \rightarrow \psi$  ako i samo ako vrijedi  $\rho \not\models \varphi$  ili  $\rho \models \psi$ ,
4. vrijedi  $\rho \models \forall\varphi$  ako i samo ako za sve  $d \in M$  vrijedi  $d \cdot \rho \models \varphi$ .

```

1 Fixpoint Sat (ρ : env M) (F : formula Σ) : Prop :=
2   match F with
3   | FPred P args => interpP P (V.map (eval ρ) args)
4   | FIndPred P args => interpIP P (V.map (eval ρ) args)
5   | FNeg G => ~ Sat ρ G
6   | FImp F G => Sat ρ F -> Sat ρ G
7   | FAll G => forall d, Sat (d .: ρ) G
8   end.

```

**Primjer 6.** Formula  $\forall x, \text{Nat}(x) \rightarrow \text{Even}(x) \vee \text{Odd}(x)$  je istinita u strukturi  $M_{PA}$  bez obzira na njenu okolinu.

```

1 Lemma every_nat_is_even_or_odd_Sat :
2   forall (ρ : env M_PA), ρ ⊨ every_nat_is_even_or_odd.

```

**Lema 1.** Neka su  $\varphi, \sigma, M$  i  $\rho$  redom proizvoljna formula, supstitucija, struktura i okolina za  $M$ . Tada vrijedi  $\rho \models \varphi[\sigma]$  ako i samo ako vrijedi  $(t \mapsto t^\rho) \circ \sigma \models \varphi$ .

```

1 Lemma strong_form_subst_sanity2 :
2   forall (Σ : signature) (φ : formula Σ) (σ : var -> term Σ)
3     (M : structure Σ) (ρ : env M),
4     ρ ⊨ (subst_formula σ φ) <-> (σ >> eval ρ) ⊨ φ.

```

Kompoziciju supstitucije  $\sigma$  i evaluacije  $t \mapsto t^\rho$  možemo nazvati *semantičkom* supstitucijom jer prvo provodi *sintaksnu* supstituciju  $\sigma$  nakon čega provodi evaluaciju. Tada možemo neformalno reći da sintaksna i semantička supstitucija komutiraju pod relacijom istinitosti.

## 2.3. Standardni modeli

Želimo ograničiti semantička razmatranja na samo one strukture koje „imaju smisla” za induktivne predikate. Prisjetimo se, predikatni simbol  $P$  mjesnosti  $n$  interpretira se na strukturi  $M$  podskupom skupa  $M^n$ . Indukciju smatramo dokazivanjem u razinama pa ima smisla promatrati *razine interpretacije* induktivnog predikata, gdje je nulta razina prazan skup, a svaku iduću razinu konstruiramo pomoću produkcija induktivnog skupa definicija i prethodnih razina. Tako je prva razina onaj podskup kojeg možemo dobiti najviše jednom primjenom produkcija, druga je razina onaj podskup kojeg možemo dobiti pomoću najviše dviju primjena produkcija, i tako dalje. Na taj se način, korak po korak, gradi *smisljena* interpretacija induktivnih predikata. Zbog mogućih međuovisnosti induktivnih predikata, razine interpretacije moramo definirati simultano. Ovaj odjeljak posvećujemo formalizaciji ovih pojmova.

**Definicija 11.** Neka je  $M$  proizvoljna struktura te neka je  $pr$  proizvoljna produkcija induktivnog skupa definicija  $\Phi$ , primjerice:

$$\frac{Q_1 \mathbf{u}_1 \dots Q_n \mathbf{u}_n \quad P_1 \mathbf{v}_1 \dots P_m \mathbf{v}_m}{P \mathbf{t}} .$$

Neka je  $I$  proizvoljna interpretacija induktivnih predikatnih simbola. Tada definiramo  $\varphi_{pr}(I)$  kao skup svih  $|P|$ -torki  $\mathbf{d}$  elemenata nosača  $M$  za koje postoji okolina  $\rho$  za  $M$  takva da:

- za sve  $i \in \{1, \dots, n\}$  vrijedi  $\mathbf{u}_i^\rho \in Q_i^M$ ,
- za sve  $j \in \{1, \dots, m\}$  vrijedi  $\mathbf{v}_j^\rho \in f(P_j)$  i
- $\mathbf{d} = \mathbf{t}^\rho$ .

```

1 Definition  $\varphi_{pr}$ 
2   (pr : production  $\Sigma$ )
3   (interp : InterpInd)
4   (ds : vec M (indpred_ar (indcons pr)))
5   : Prop :=
6     exists ( $\rho$  : env M),
7     (forall Q us, List.In (Q; us) (preds pr) ->
8       interpP Q (V.map (eval  $\rho$ ) us)) /\
9     (forall P ts, List.In (P; ts) (indpreds pr) ->
10      interp P (V.map (eval  $\rho$ ) ts)) /\
11      ds = V.map (eval  $\rho$ ) (indargs pr).

```

Operator  $\varphi_{pr}$  je formalizacija ideje primjene produkcije. Nadalje, potrebno je definirati operator koji će uzeti u obzir sve produkcije koje se odnose na  $P$ . Definiramo  $\varphi_P(f)$  kao uniju svih  $\varphi_{pr'}(f)$  gdje je  $pr'$  produkcija u kojoj se  $P$  javlja u konkluziji.

```

1 Definition  $\varphi_P$ 
2   (P : IndPredS  $\Sigma$ )
3   (interp : InterpInd)
4   : vec M (indpred_ar P) -> Prop.
5   refine (fun ds => _).
6   refine (@ex (production  $\Sigma$ ) (fun pr => _)).
7   refine (@ex (P = indcons pr /\  $\Phi$  pr) (fun '(conj Heq H $\Phi$ ) => _)).
8   rewrite Heq in ds.
9   exact ( $\varphi_{pr}$  pr interp ds).
10 Defined.

```

Konačno, definiramo operator skupa definicija  $\varphi_\Phi$  kao preslikavanje koje svakom induktivnom predikatnom simbolu  $P$  pridružuje skup  $\varphi_P(f)$ .

```

1 Definition  $\varphi_\Phi$  (interp : InterpInd) : InterpInd :=
2   fun P =>  $\varphi_P$  P interp.

```

Operator  $\varphi_\Phi$  omogućuje simultanu primjenu produkcija.

*Napomena.* Kako je funkcija  $f$  bila uvedena na samom početku prethodne definicije, u stvari definicija operatora  $\varphi_\Phi$  glasi  $\varphi_\Phi(f)(P) := \varphi_P(f)$ .

**Primjer 7.** Ilustrirajmo operator  $\varphi_\Phi$  na strukturi  $M_{PA}$  i skupu induktivnih definicija  $\Phi_{PA}$ . Prisjetimo se, u proširenoj Peanovoj signaturi imamo tri induktivna predikatna simbola;  $Nat$ ,  $Even$  i  $Odd$ , i svi su oni jednomjesni. Dakle, domena i kodomena operatora  $\varphi_\Phi$  u ovom primjeru je skup  $\mathcal{P}(\mathbb{N})^3$ , gdje prva, druga i treća projekcija odgovaraju redom

interpretacijama predikata  $Nat$ ,  $Even$  i  $Odd$ . Nadalje, vrijedi:

$$\varphi_{\Phi_{PA}, M_{PA}}(N, E, O) = (\{0\} \cup \{n + 1 \mid n \in N\}, \{0\} \cup \{o + 1 \mid o \in O\}, \{e + 1 \mid e \in E\}).$$

Prema notaciji u prethodnoj napomeni, funkcija  $f$  je ovdje preslikavanje

$$(Nat \mapsto N, Even \mapsto E, Odd \mapsto O)$$

koje možemo reprezentirati uređenom trojkom  $(N, E, O)$ . Indeksiranje induktivnim predikatnim simbolom odgovara prethodno navedenim projekcijama pa vrijedi primjerice:

$$\varphi_{\Phi_{PA}, M_{PA}}(N, E, O)(Nat) = \{0\} \cup \{n + 1 \mid n \in N\}.$$

**Definicija 12.** Neka su  $U$  i  $V$  proizvoljne interpretacije induktivnih predikatnih simbola. Kažemo da je interpretacija  $U$  *podinterpretacija* interpretacije  $V$  i pišemo  $U \sqsubseteq V$  ako za svaki induktivni predikatni simbol  $P$  vrijedi  $U(P) \subseteq V(P)$ .

```
1 Definition subinterp (U V : InterpInd) := forall P v, U P v -> V P v.
2 Notation "U ⊆ V" := (subinterp U V) (at level 11).
```

**Propozicija 1.** Operator  $\varphi_\Phi$  je monoton.

```
1 Definition monotone (F : InterpInd -> InterpInd) :=
2   forall U V : InterpInd, U ⊆ V -> F U ⊆ F V.
3
4 Proposition φ_Φ_monotone : monotone φ_Φ.
```

**Definicija 13.** Neka je  $M$  proizvoljna struktura. Definiramo *aproksimaciju* skupa induktivnih definicija  $\Phi$  razine  $\alpha \in \mathbb{N}$ , s oznakom  $\varphi_\Phi^\alpha$ , rekurzivno na način:

1.  $\varphi_\Phi^0(P) := \emptyset$
2.  $\varphi_\Phi^{\alpha+1} := \varphi_\Phi(\varphi_\Phi^\alpha)$ .

Neformalno možemo reći da operator  $\varphi_\Phi$  *poboljšava* aproksimaciju prethodne razine.

```
1 Fixpoint φ_Φ_n (α : nat) : InterpInd :=
2   match α with
3   | 0 => fun _ _ => False
4   | S α => φ_Φ (φ_Φ_n α)
5   end.
```

Tada je *aproksimant* induktivnog predikatnog simbola  $P$  razine  $\alpha$  upravo  $\varphi_{\Phi}^{\alpha}(P)$ .

*Napomena.* Brotherston je definirao aproksimaciju razine  $\alpha$  za pojedini induktivni predikatni simbol kao uniju poboljšanih aproksimacija svih nižih razina. Takva je definicija ekvivalentna našoj.

**Lema 2.** Za svaki prirodni broj  $\alpha$  i induktivni predikatni simbol  $P$  vrijedi

$$\varphi_{\Phi}^{\alpha}(P) = \bigcup_{\beta < \alpha} \varphi_{\Phi}(\varphi_{\Phi}^{\beta}(P)).$$

```
1 Lemma  $\varphi\_Phi\_n\_characterization$  : forall  $\alpha$  P v,
2    $\varphi\_Phi\_n \alpha P v \leftrightarrow$  exists  $\beta$ ,  $\beta < \alpha \wedge \varphi\_Phi (\varphi\_Phi\_n \beta) P v$ .
```

**Primjer 8.** Za svaki prirodni broj  $\alpha$ , a u skladu s primjerom 7, aproksimant razine  $\alpha$  induktivnog predikatnog simbola  $Nat$  je skup svih prirodnih brojeva strogo manjih od  $\alpha$ , odnosno vrijedi:

$$\varphi_{\Phi_{PA}, M_{PA}}^{\alpha}(Nat) = \{0, 1, \dots, \alpha - 1\}.$$

```
1 Lemma approximations_of_PA_Nat : forall  $\alpha$  n,
2    $\varphi\_Phi\_n \Phi\_PA M\_PA \alpha PA\_Nat [n] \leftrightarrow n < \alpha$ .
```

**Definicija 14.** *Aproksimacija razine  $\omega$* , s oznakom  $\varphi_{\Phi}^{\omega}$ , je (za svaki pojedini induktivni predikatni simbol) unija aproksimacija razina manjih od  $\omega$ .

```
1 Definition  $\varphi\_Phi\_omega$  : InterpInd := fun P v => exists  $\alpha$ ,  $\varphi\_Phi\_n \alpha P v$ .
```

**Primjer 9.** Aproksimant razine  $\omega$  induktivnog predikatnog simbola  $Nat$  je cijeli skup prirodnih brojeva  $\mathbb{N}$ .

```
1 Lemma NAT_ $\varphi\_Phi\_omega$ : forall n,  $\varphi\_Phi\_omega \Phi\_PA M\_PA PA\_Nat [n]$ .
```

**Definicija 15.** Neka je  $U$  proizvoljna interpretacija induktivnih predikatnih simbola. Kažemo da je interpretacija  $U$  *prefiksna točka* ako je poboljšanje interpretacije  $U$  podinterpretacija od  $U$ , odnosno vrijedi  $\varphi_{\Phi}(U) \sqsubseteq U$ . Za interpretaciju induktivnih predikatnih simbola  $V$  kažemo da je *najmanja prefiksna točka* ako je  $V$  prefiksna točka te je podinterpretacija svake druge prefiksne točke.

```

1 Definition prefixed (U : InterpInd) :=  $\varphi_\Phi$  U  $\sqsubseteq$  U.
2 Definition least (P : InterpInd -> Prop) (U : InterpInd) :=
3   P U /\ (forall V, P V -> U  $\sqsubseteq$  V).

```

**Lema 3.** Aproksimacija razine  $\omega$  je najmanja prefiksna točka operatora  $\varphi_\Phi$ .

```

1 Lemma  $\varphi_\Phi_\omega$ _least_prefixed : least prefixed  $\varphi_\Phi_\omega$ .

```

**Definicija 16.** Kažemo da je struktura  $M$  *standardni model* za  $\Phi$  ako interpretira svaki induktivni predikatni simbol njegovim aproksimantom razine  $\omega$ .

```

1 Definition standard_model ( $\Phi$ : IndDefSet  $\Sigma$ ) (M : structure  $\Sigma$ ) : Prop :=
2   forall (P : IndPredS  $\Sigma$ ) ts, interpIP P ts <->  $\varphi_\Phi_\omega$   $\Phi$  M P ts.

```

Prema lemi 3, standardni model je najmanja struktura koja ima smisla za zadani skup induktivnih definicija.

**Propozicija 2.** Struktura  $M_{PA}$  je standardni model za skup induktivnih definicija  $\Phi_{PA}$ .

```

1 Lemma standard_model__PA : standard_model  $\Phi_{PA}$  M__PA.

```

### 3. Sistem sekvenata s induktivnim definicijama

Cilj je ovog odjeljka definirati dokazni sustav  $LKID$  za logiku  $FOL_{ID}$ . Ovaj dokazni sustav temelji se na Gentzenovu računu sekvenata, a proširen je lijevim i desnim pravilima za induktivne predikate. Prije no što definiramo  $LKID$ , potrebno je definirati pojam sekvente i međusobne zavisnosti induktivnih predikata.

**Definicija 17.** Neka su  $\Gamma$  i  $\Delta$  proizvoljne liste formula. Tada je *sekventa* uređeni par  $(\Gamma, \Delta)$ , a označavamo ju s  $\Gamma \vdash \Delta$ .

```
1 Inductive sequent : Set :=  
2 | mkSeq ( $\Gamma \Delta$  : list (formula  $\Sigma$ )).
```

Sekventom  $\Gamma \vdash \Delta$  konceptualno tvrdimo da istinitost *svake* tvrdnje u  $\Gamma$  povlači istinitost *neke* tvrdnje u  $\Delta$ . Kažemo da je sekventa  $\Gamma \vdash \Delta$  *dokaziva*, ili da  $\Gamma$  *izvodi*  $\Delta$ , ako u sustavu  $LKID$  postoji *dokaz* za  $\Gamma \vdash \Delta$ . Neformalno možemo reći da je dokaz sekvente  $\Gamma \vdash \Delta$  konačno stablo u kojem je sekventa  $\Gamma \vdash \Delta$  korijen te svaki čvor stabla slijedi iz svoje djece u skladu s pravilima izvoda sustava  $LKID$ . Kasnije ćemo precizirati pojam dokaza.

**Definicija 18.** Neka su  $P_i$  i  $P_j$  proizvoljni induktivni predikatni simboli. Kažemo da je simbol  $P_i$  u relaciji *Prem* sa simbolom  $P_j$  ako postoji produkcija u skupu induktivnih definicija  $\Phi$  takva da se simbol  $P_i$  javlja u njenoj konkluziji te se simbol  $P_j$  javlja u njenim premisama.

```
1 Definition Prem (Pi Pj : IndPredS  $\Sigma$ ) :=  
2   exists pr,  $\Phi$  pr /\  
3     indcons pr = Pi /\  
4     exists ts, List.In (Pj; ts) (indpreds pr).
```



**Definicija 19.** Definiramo relaciju  $Prem^*$  kao refleksivno i tranzitivno zatvorenje relacije  $Prem$  na skupu induktivnih predikatnih simbola.

```
1 Definition Prem_star := clos_refl_trans (IndPredS  $\Sigma$ ) Prem.
```

Za induktivne predikatne simbole  $P$  i  $Q$  kažemo da su *međusobno zavisni* ako vrijedi  $Prem^*(P, Q)$  i  $Prem^*(Q, P)$ .

```
1 Definition mutually_dependent (P Q : IndPredS  $\Sigma$ ) :=  
2   Prem_star P Q /\ Prem_star Q P.
```

Za međusobno zavisne induktivne predikatne simbole kažemo još da su definirani simultanom rekurzijom. Simultana rekurzija se koristi za formalnu definiciju međusobno zavisnih induktivnih predikata i u mnogo složenijim sustavima od  $FOL_{ID}$ ; primjerice, Coqovi predikati `EVEN` i `ODD` iz primjera 5 definirani su simultanom rekurzijom.

**Primjer 10.** U skupu induktivnih definicija  $\Phi_{PA}$ , induktivni predikatni simboli *Even* i *Odd* su međusobno zavisni.

```
1 Example mut_dep_E_O :  
2   mutually_dependent  $\Phi_{PA}$  PA_Even PA_Odd.
```

S druge strane, induktivni predikatni simboli *Nat* i *Even* nisu međusobno zavisni.

```
1 Example not_mut_dep_N_E :  
2   ~ mutually_dependent  $\Phi_{PA}$  PA_Nat PA_Even.
```

**Lema 4.** Međusobna zavisnost je relacija ekvivalencije na skupu induktivnih predikatnih simbola.

```
1 Lemma mutually_dependent_equiv : Equivalence mutually_dependent.
```

Sustav *LKID* sastoji se od četiri vrste pravila izvoda: strukturalna, propozicijska, pravila za kvantifikatore i pravila za induktivne predikate. Navest ćemo ih tim redom, a definirati kroz više blokova Coqova koda. Napominjemo da se sustav *LKID* razlikuje od Gentzenova sustava *LK* na nekoliko mjesta, zbog implementacijskih detalja te zbog induktivnih definicija. Za neko pravilo izvoda kažemo da je *dopustivo* ako sve što možemo dokazati (pod nekim pretpostavkama) korištenjem tog pravila, možemo dokazati pod istim pretpostavkama i bez njegova korištenja.

```
1 Inductive LKID : sequent -> Prop := (* nastavlja se *)
```

### 3.1. Strukturalna pravila

Strukturalna su pravila navedena na prikazu 1, a služe baratanju strukturom sekvente. Iako je sekventa implementirana kao par lista formula, te liste se ponašaju kao skupovi u strukturalnim pravilima jer za sadržanost jedne liste u drugoj koristimo Coqov predikat `incl` koji se oslanja na predikat `In`. Posljedično, pravilo slabljenja *Wk* povlači pravila permutacije i kontrakcije.

$$\begin{array}{c}
 \frac{\Gamma \cap \Delta \neq \emptyset}{\Gamma \vdash \Delta} (Ax) \\
 \frac{\Gamma' \vdash \Delta' \quad \Gamma' \subseteq \Gamma \quad \Delta' \subseteq \Delta}{\Gamma \subseteq \Delta} (Wk) \\
 \frac{\Gamma \vdash \varphi, \Delta \quad \varphi, \Gamma \vdash \Delta}{\Gamma \vdash \Delta} (Cut) \\
 \frac{\Gamma \vdash \Delta}{\Gamma[\sigma] \vdash \Delta[\sigma]} (Subst)
 \end{array}$$

Prikaz 1: Strukturalna pravila sustava *LKID*.

```

2  (* Strukturalna pravila. *)
3  | Ax : forall Γ Δ φ, In φ Γ -> In φ Δ -> LKID (Γ ⊢ Δ)
4  | Wk : forall Γ' Δ' Γ Δ,
5      Γ' ⊆ Γ ->
6      Δ' ⊆ Δ ->
7      LKID (Γ' ⊢ Δ') ->
8      LKID (Γ ⊢ Δ)
9  | Cut : forall Γ Δ φ,
10     LKID (Γ ⊢ φ :: Δ) ->
11     LKID (φ :: Γ ⊢ Δ) ->
12     LKID (Γ ⊢ Δ)
13 | Subst : forall Γ Δ,
14     LKID (Γ ⊢ Δ) ->
15     forall σ, LKID (map (subst_formula σ) Γ ⊢ map (subst_formula σ) Δ)

```

**Primjer 11.** Prošireno pravilo aksioma je dopustivo.

$$\frac{}{\varphi, \Gamma \vdash \varphi, \Delta} (AxExtended)$$

**Lemma** `AxExtended` : `forall Γ Δ φ, LKID (φ :: Γ ⊢ φ :: Δ)`.

Nadalje, pravilo permutacije je dopustivo.

$$\frac{\Gamma' \vdash \Delta' \quad \Gamma' \sim \Gamma \quad \Delta' \sim \Delta}{\Gamma \vdash \Delta} (Perm)$$

Ovdje znakom  $\sim$  označujemo relaciju permutacije na listama.

```

Lemma Perm : forall  $\Gamma'$   $\Delta'$   $\Gamma$   $\Delta$ ,
  Permutation  $\Gamma'$   $\Gamma$  ->
  Permutation  $\Delta'$   $\Delta$  ->
  LKID ( $\Gamma' \vdash \Delta'$ ) ->
  LKID ( $\Gamma \vdash \Delta$ ).

```

Konačno, lijeva i desna pravila kontrakcije su dopustiva.

$$\frac{\varphi, \varphi, \Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta} (ContrL)$$

$$\frac{\Gamma \vdash \varphi, \varphi, \Delta}{\Gamma \vdash \varphi, \Delta} (ContrR)$$

```

Lemma ContrL : forall  $\Gamma$   $\Delta$   $\varphi$ ,
  LKID ( $\varphi :: \varphi :: \Gamma \vdash \Delta$ ) -> LKID ( $\varphi :: \Gamma \vdash \Delta$ ).
Lemma ContrR : forall  $\Gamma$   $\Delta$   $\varphi$ ,
  LKID ( $\Gamma \vdash \varphi :: \varphi :: \Delta$ ) -> LKID ( $\Gamma \vdash \varphi :: \Delta$ ).

```

## 3.2. Propozicijska pravila

Propozicijska su pravila navedena na prikazu 2, a služe rasuđivanju s implikacijama i negacijama. Kao što smo rekli, ostale propozicijske veznike shvaćamo kao pokrate. Ova su pravila identična u sustavima *LK* i *LKID*.

$$\frac{\Gamma \vdash \varphi, \Delta}{\neg \varphi, \Gamma \vdash \Delta} (NegL)$$

$$\frac{\varphi, \Gamma \vdash \Delta}{\Gamma \vdash \neg \varphi, \Delta} (NegR)$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \psi, \Gamma \vdash \Delta}{\varphi \rightarrow \psi, \Gamma \vdash \Delta} (ImpL)$$

$$\frac{\varphi, \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \rightarrow \psi, \Delta} (ImpR)$$

Prikaz 2: Propozicijska pravila sustava *LKID*.

```

16 (* Propozicijska pravila. *)
17 | NegL : forall  $\Gamma \Delta \varphi$ , LKID ( $\Gamma \vdash \varphi :: \Delta$ ) -> LKID (FNeg  $\varphi :: \Gamma \vdash \Delta$ )
18 | NegR : forall  $\Gamma \Delta \varphi$ , LKID ( $\varphi :: \Gamma \vdash \Delta$ ) -> LKID ( $\Gamma \vdash$  FNeg  $\varphi :: \Delta$ )
19 | ImpL : forall  $\Gamma \Delta \varphi \psi$ ,
20   LKID ( $\Gamma \vdash \varphi :: \Delta$ ) -> LKID ( $\psi :: \Gamma \vdash \Delta$ ) ->
21   LKID (FImp  $\varphi \psi :: \Gamma \vdash \Delta$ )
22 | ImpR : forall  $\Gamma \Delta \varphi \psi$ ,
23   LKID ( $\varphi :: \Gamma \vdash \psi :: \Delta$ ) -> LKID ( $\Gamma \vdash$  (FImp  $\varphi \psi$ ) ::  $\Delta$ )

```

**Primjer 12.** Desno pravilo za konjunkciju je dopustivo.

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta} (AndR)$$

```

Lemma AndR : forall  $\Gamma \Delta \varphi \psi$ ,
  LKID ( $\Gamma \vdash \varphi :: \Delta$ ) -> LKID ( $\Gamma \vdash \psi :: \Delta$ ) ->
  LKID ( $\Gamma \vdash$  FAnd  $\varphi \psi :: \Delta$ ).

```

Lijevo pravilo za disjunkciju je dopustivo.

$$\frac{\varphi, \Gamma \vdash \Delta \quad \psi, \Gamma \vdash \Delta}{\varphi \vee \psi, \Gamma \vdash \Delta} (OrL)$$

```

Lemma OrL : forall  $\Gamma \Delta \varphi \psi$ ,
  LKID ( $\varphi :: \Gamma \vdash \Delta$ ) -> LKID ( $\psi :: \Gamma \vdash \Delta$ ) ->
  LKID (FOr  $\varphi \psi :: \Gamma \vdash \Delta$ ).

```

### 3.3. Pravila za kvantifikatore

Pravila za kvantifikatore navedena su na prikazu 3, a odnose se na univerzalnu kvantifikaciju. Ova se pravila razlikuju od analognih u sustavu *LK* zbog korištenja de Bruijnovih indeksa kod kvantifikacije. U lijevom pravilu koristimo supstituciju  $t \cdot \sigma_{id}$  u formuli  $\varphi$ , gdje je  $\sigma_{id}$  supstitucija identiteta, kako bismo iskazali da se term  $t$  javlja na mjestu neke varijable  $x$ . Za desno pravilo, primijetimo da se varijabla 0 ne javlja u listama formula  $\Gamma^\dagger$  i  $\Delta^\dagger$  zbog pomicanja. Kako implicitno kvantificiramo po varijabli 0, a ona se javlja (eventualno) samo u formuli  $\varphi$ , smijemo univerzalno kvantificirati formulu  $\varphi$ .

$$\frac{\varphi[t \cdot \sigma_{id}], \Gamma \vdash \Delta}{\forall \varphi, \Gamma \vdash \Delta} (AllL)$$

$$\frac{\Gamma^\uparrow \vdash \varphi, \Delta^\uparrow}{\Gamma \vdash \forall \varphi, \Delta} (AllR)$$

Prikaz 3: Pravila za kvantifikatore sustava *LKID*.

```

24 (* Kvantifikatorska pravila. *)
25 | AllL : forall Γ Δ φ t,
26   LKID (subst_formula (t :: ids) φ :: Γ ⊢ Δ) ->
27   LKID (FA11 φ :: Γ ⊢ Δ)
28 | AllR : forall Γ Δ φ,
29   LKID (shift_formulas Γ ⊢ φ :: shift_formulas Δ) ->
30   LKID (Γ ⊢ (FA11 φ) :: Δ)

```

**Primjer 13.** Lijevo pravilo za egzistencijalnu kvantifikaciju je dopustivo.

$$\frac{\varphi, \Gamma^\uparrow \vdash \Delta^\uparrow}{\exists \varphi, \Gamma \vdash \Delta} (ExistL)$$

```

Lemma ExistL : forall Γ Δ φ,
  LKID (φ :: shift_formulas Γ ⊢ shift_formulas Δ) ->
  LKID (FExist φ :: Γ ⊢ Δ).

```

Preostaje nam definicija pravila za induktivne predikatne simbole, gdje ćemo prvo za svaki induktivni predikatni simbol definirati lijevo pravilo koje nazivamo *pravilo indukcije*, a zatim ćemo za svaku produkciju definirati desno pravilo koje nazivamo *produkcij-sko pravilo*.

### 3.4. Pravila indukcije

Neka je  $P_j$  proizvoljni induktivan predikatni simbol za koji definiramo pravilo indukcije. Svakom induktivnom predikatnom simbolu  $P_i$  dodjeljujemo proizvoljnu  $|P_i|$ -torku *induktivnih varijabli*  $\mathbf{z}_i$  te proizvoljnu *pretpostavku indukcije*  $G_i$  takvu da vrijedi  $G_i = P_i(\mathbf{z}_i)$  ako  $P_i$  i  $P_j$  nisu međusobno zavisni. Sada pravilo indukcije za induktivan predikatni simbol  $P_j$  glasi:

$$\frac{\text{male premise} \quad G_j[\mathbf{u}/\mathbf{z}_j], \Gamma \vdash \Delta}{P_j \mathbf{u}, \Gamma \vdash \Delta} (P_j Ind),$$

gdje sekventu  $G_j[\mathbf{u}/\mathbf{z}_i], \Gamma \vdash \Delta$  zovemo *velikom premisom*, a za svaku produkciju u skupu induktivnih definicija  $\Phi$  u čijoj se konkluziji javlja induktivni predikatni simbol  $P_i$  međusobno zavisano sa simbolom  $P_j$ , primjerice:

$$\frac{Q_1 \mathbf{u}_1 \dots Q_n \mathbf{u}_n \quad P_{i_1} \mathbf{v}_{i_1} \dots P_{i_m} \mathbf{v}_{i_m}}{P_i \mathbf{t}},$$

postoji *mala premisa* oblika:

$$Q_1 \mathbf{u}_1^{\uparrow k}, \dots, Q_n \mathbf{u}_n^{\uparrow k}, G_{i_1}[\mathbf{v}_{i_1}^{\uparrow k}/\mathbf{z}_{i_1}], \dots, G_{i_m}[\mathbf{v}_{i_m}^{\uparrow k}/\mathbf{z}_{i_m}], \Gamma \vdash G_i[\mathbf{t}^{\uparrow k}/\mathbf{z}_i], \Delta$$

gdje je  $k$  neki prirodan broj takav da se nijedan  $k' \geq k$  ne pojavljuje u listama formula  $\Gamma$  i  $\Delta$  te formuli  $P_j \mathbf{u}$  (takav postoji jer se u tih konačno mnogo formula sveukupno pojavljuje samo konačno mnogo varijabli), a  $s^{\uparrow k}$  označavamo supstituciju koja svaku varijablu  $n$  zamjenjuje varijablom  $n + k$ .

Ilustrirajmo pravilo indukcije primjerima u Peanovoj aritmetici. Kako se pravila za induktivne predikate sustava  $LKID$  odnose na proizvoljni skup induktivnih definicija  $\Phi$ , označimo s  $LKID_{PA}$  sustav koji se odnosi na skup induktivnih definicija  $\Phi_{PA}$ .

**Definition**  $LKID\_PA := LKID \ \Phi\_PA$ .

**Primjer 14.** Pravilo indukcije simbola  $Nat$  u sustavu  $LKID_{PA}$  glasi:

$$\frac{\Gamma \vdash G(o), \Delta \quad G(x), \Gamma \vdash G(s(x)), \Delta \quad G(t), \Gamma \vdash \Delta}{Nat(t), \Gamma \vdash \Delta} (NatInd),$$

gdje je  $G$  pretpostavka indukcije vezana uz induktivni predikatni simbol  $Nat$ , a  $x$  je varijabla koja se ne javlja u  $\Gamma, \Delta$  niti u  $Nat(t)$ .

**Primjer 15.** Pravilo indukcije simbola  $Even$  u sustavu  $LKID_{PA}$  glasi:

$$\frac{\Gamma \vdash G_E(o), \Delta \quad \Gamma, G_E(x) \vdash G_O(s(x)), \Delta \quad \Gamma, G_O(x) \vdash G_E(s(x)), \Delta \quad \Gamma, G_E(t) \vdash \Delta}{\Gamma, Even(t) \vdash \Delta} (EvenInd),$$

gdje su  $G_E$  i  $G_O$  pretpostavke indukcije vezane uz induktivne predikatne simbole  $Even$  i  $Odd$ , a  $x$  je varijabla koja se ne javlja u  $\Gamma, \Delta$  niti u  $Even(t)$ .

```

31 (* Pravila indukcije. *)
32 | Ind : forall  $\Gamma$   $\Delta$ 
33     (Pj : IndPredS  $\Sigma$ ) (u : vec _ (indpred_ar Pj))
34     (z_i : forall P, vec var (indpred_ar P))
35     (z_i_nodup : forall P, VecNoDup (z_i P))
36     (G_i : IndPredS  $\Sigma$  -> formula  $\Sigma$ )
37     (HG2 : forall Pi, ~mutually_dependent Pi Pj ->
38         G_i Pi = FIndPred
39             Pi
40             (V.map var_term (z_i Pi))),
41 let max $\Gamma$  := max_fold (map some_var_not_in_formula  $\Gamma$ ) in
42 let max $\Delta$  := max_fold (map some_var_not_in_formula  $\Delta$ ) in
43 let maxP := some_var_not_in_formula (FIndPred Pj u) in
44 let shift_factor := max maxP (max max $\Gamma$  max $\Delta$ ) in
45 let Fj := subst_formula
46     (finite_subst (z_i Pj) u)
47     (G_i Pj) in
48 let minor_premises :=
49     (forall pr (H $\Phi$ : $\Phi$  pr) (Hdep: mutually_dependent (indcons pr) Pj),
50     let Qs := shift_formulas_by
51         shift_factor
52         (FPreds_from_preds (preds pr)) in
53     let Gs := map (fun '(P; args) =>
54         let shifted_args :=
55             V.map
56                 (shift_term_by shift_factor)
57                 args in
58         let  $\sigma$  :=
59             finite_subst
60                 (z_i P)
61                 (shifted_args) in
62         let G := G_i P in
63         subst_formula  $\sigma$  G)
64         (indpreds pr) in
65     let Pi := indcons pr in
66     let ty := V.map
67         (shift_term_by shift_factor)
68         (indargs pr) in
69     let Fi := subst_formula
70         (finite_subst (z_i Pi) ty)
71         (G_i Pi) in
72     LKID (Qs ++ Gs ++  $\Gamma \vdash Fi :: \Delta$ )
73 in
74 minor_premises ->
75 LKID (Fj ::  $\Gamma \vdash \Delta$ ) ->
76 LKID (FIndPred Pj u ::  $\Gamma \vdash \Delta$ )

```

### 3.5. Produkcijska pravila

Neka je  $pr$  proizvoljna produkcija za koju definiramo produkcijsko pravilo. Možemo pretpostaviti da je  $pr$  oblika:

$$\frac{Q_1 \mathbf{u}_1 \dots Q_n \mathbf{u}_n \quad P_1 \mathbf{v}_1 \dots P_m \mathbf{v}_m}{Pt}.$$

Tada produkcijsko pravilo za produkciju  $pr$  (i proizvoljnu supstituciju  $\sigma$ ) glasi:

$$\frac{\Gamma \vdash Q_1 \mathbf{u}_1[\sigma], \Delta \quad \dots \quad \Gamma \vdash Q_n \mathbf{u}_n[\sigma] \quad \Gamma \vdash P_1 \mathbf{v}_1[\sigma], \Delta \quad \dots \quad \Gamma \vdash P_m \mathbf{v}_m[\sigma]}{\Gamma \vdash Pt[\sigma]} (prProd).$$

```

77 (* Produkcijska pravila. *)
78 | Prod : forall Γ Δ pr σ,
79   Φ pr ->
80   (forall Q us,
81     In (Q; us) (preds pr) ->
82     LKID (Γ ⊢ (FPred Q (V.map (subst_term σ) us) :: Δ))) ->
83   (forall P ts,
84     In (P; ts) (indpreds pr) ->
85     LKID (Γ ⊢ (FIndPred P (V.map (subst_term σ) ts) :: Δ))) ->
86   LKID ( Γ ⊢ FIndPred
87         (indcons pr)
88         (V.map (subst_term σ) (indargs pr))
89         :: Δ ).

```

**Primjer 16.** Produkcijsko pravilo za produkciju  $PA\_prod\_E\_succ$  glasi:

$$\frac{\Gamma \vdash Odd(x)[\sigma], \Delta}{\Gamma \vdash Even(s(x))[\sigma], \Delta} (PA\_prod\_E\_succProd)$$

gdje je  $x$  proizvoljna varijabla, a  $\sigma$  proizvoljna supstitucija. Supstitucijom  $\sigma$  kažemo da u produkcijama imena varijabli nisu bitna, već samo njihov redoslijed.

### 3.6. Dokaz i teorem

Prikazali smo sva pravila izvoda sustava  $LKID$  te konačno možemo definirati dokaz. Za proizvoljnu sekventu  $s := (\Gamma \vdash \Delta)$ , njen **dokaz** je stanovnik tipa  $LKID$   $s$ . Za formulu  $\varphi$  kažemo da je **dokaziva** u sustavu  $LKID$  te pišemo  $\vdash \varphi$  ako postoji dokaz sekvente  $[] \vdash [\varphi]$ .



```

1 Definition provable (proofsystem : sequent -> Prop) (φ : formula Σ) :=
2   proofsystem ([] ⊢ [φ]).

```

Neformalni pojam konačnog stabla formaliziran je induktivnim tipom LKID, dok njegovi konstruktori formaliziraju odgovarajuća pravila izvoda.

**Primjer 17.** Za proizvoljnu formulu  $\varphi$ , formula  $\varphi \vee \neg\varphi$  je dokaziva u sustavu LKID.

```

1 Lemma LKID_XM : forall φ, provable LKID (FOr φ (FNeg φ)).

```

$$\begin{array}{c}
\frac{}{\varphi \vdash \varphi} (Ax) \\
\frac{}{\vdash \neg\varphi, \varphi} (NegR) \\
\frac{}{\vdash \varphi, \neg\varphi} (Perm) \\
\frac{}{\vdash \varphi \vee \neg\varphi} (OrR)
\end{array}$$

**Primjer 18.** Za sustav LKID vrijedi *princip eksplozije*.

```

1 Lemma LKID_EXPLOSION : forall φ Δ, LKID ([FAnd φ (FNeg φ)] ⊢ Δ).

```

$$\begin{array}{c}
\frac{}{\varphi \vdash \varphi, \Delta} (Ax) \\
\frac{}{\neg\varphi, \varphi \vdash \Delta} (NegL) \\
\frac{}{\varphi, \neg\varphi \vdash \Delta} (Perm) \\
\frac{}{\varphi \wedge \neg\varphi \vdash \Delta} (AndL)
\end{array}$$

**Primjer 19.** Formula  $\forall x, Even(x) \rightarrow Even(s(s(x)))$  je dokaziva u sustavu  $LKID_{PA}$ .

```

1 Definition Even_succ_succ_Even : formula Σ_PA :=
2   let x := var_term 0 in
3   FAll
4     (FImp
5       (FIndPred PA_Even [x])
6       (FIndPred PA_Even [TFunc PA_succ
7         [TFunc PA_succ
8           [x]]]))).

```

Neformalni dokaz prikazujemo idućim stablom, a radi preglednosti pišemo  $Et$  umjesto  $Even(t)$  te  $Ot$  umjesto  $Odd(t)$  i ne pišemo zagrade u pozivu funkcijskog simbola  $s$ .

$$\begin{array}{c}
\frac{}{Ex \vdash Ex, Essx} (Ax) \quad \frac{}{Ex, Ossx \vdash Ossx} (Ax) \\
\frac{}{Ex \vdash Ossx, Essx} (PA\_prod\_0\_succProd) \quad \frac{}{Ex, Ossx \vdash Essx} (PA\_prod\_E\_succProd) \\
\frac{}{} (Cut) \\
\frac{}{Ex \vdash Essx} (ImpR) \\
\frac{}{\vdash Ex \rightarrow Essx} (AllR) \\
\vdash \forall x, Ex \rightarrow Essx
\end{array}$$

1 **Lemma** provable\_Even\_succ\_succ\_Even :  
 2 provable LKID\_\_PA Even\_succ\_succ\_Even.

**Primjer 20.** Formula  $\forall x, Nat(x) \rightarrow Even(x) \vee Odd(x)$  je dokaziva u sustavu  $LKID_{PA}$ . Neformalni dokaz prikazujemo idućim stablom koristeći iste pokrate kao u primjeru 19, te  $Nt$  za  $Nat(t)$ .

$$\begin{array}{c}
 \frac{}{\vdash Eo} (1) \quad \frac{}{Ey \vdash Ey} (Ax) \quad \frac{}{Oy \vdash Oy} (Ax) \\
 \vdash Eo, Ex \vee Ox \quad \frac{}{Ey \vdash Osy} (2) \quad \frac{}{Oy \vdash E sy} (3) \\
 \vdash Eo \vee Oo, Ex \vee Ox \quad \frac{}{Ey \vdash E sy \vee O sy} (OrR_2) \quad \frac{}{Oy \vdash E sy \vee O sy} (OrR_1) \\
 \vdash Eo \vee Oo, Ex \vee Ox \quad \frac{}{Ey \vee Oy \vdash E sy \vee O sy} (OrL) \\
 \vdash Eo \vee Oo, Ex \vee Ox \quad \frac{}{Ey \vee Oy \vdash E sy \vee O sy, Ex \vee Ox} (Wk) \quad \frac{}{Ex \vee Ox \vdash Ex \vee Ox} (Ax) \\
 \vdash Eo \vee Oo, Ex \vee Ox \quad \frac{}{Nx \vdash Ex \vee Ox} (ImpR) \quad \frac{}{Ex \vee Ox \vdash Ex \vee Ox} (NatInd) \\
 \vdash Eo \vee Oo, Ex \vee Ox \quad \frac{}{Nx \rightarrow Ex \vee Ox} (AllR) \\
 \vdash \forall x, Nx \rightarrow Ex \vee Ox
 \end{array}$$

Ovdje je pretpostavka indukcije formula  $Ex \vee Ox$ . Valja primijetiti da smo u dokazu koristili (označene brojevima) i produkcijska pravila za produkcije:

- (1) PA\_prod\_E\_zero,
- (2) PA\_prod\_0\_succ i
- (3) PA\_prod\_E\_succ.

1 **Lemma** provable\_every\_nat\_is\_even\_or\_odd :  
 2 provable LKID\_\_PA every\_nat\_is\_even\_or\_odd.

### 3.7. Adekvatnost

Za dokazivače teorema nužno je da dokazivost formule povlači njenu istinitost — u 1. poglavlju objasnili smo zašto. Ovo svojstvo dokaznog sustava naziva se *adekvatnost*, a ovdje ćemo je prikazati na primjeru sustava  $LKID$ . Za istinitost ćemo promatrati samo strukture koje su standardni modeli, jer u takvima interpretacije induktivnih predikatnih simbola imaju smisla. Napomenimo da je  $LKID$  adekvatan i za nestandardne modele.

**Definicija 20.** Kažemo da je sekventa  $\Gamma \vdash \Delta$  *istinita* i pišemo  $\Gamma \models \Delta$  ako i samo ako za svaki standardni model  $M$  i njegovu okolinu  $\rho$  vrijedi:

$$(\forall \varphi, \varphi \in \Gamma \rightarrow \rho \models \varphi) \rightarrow \exists \psi, \psi \in \Delta \wedge \rho \models \psi,$$

odnosno istinitost svake formule u  $\Gamma$  povlači istinitost neke formule u  $\Delta$ , u skladu s intuitivnim shvaćanjem sekventi.

```

1 Definition Sat_sequent (s : sequent) : Prop :=
2   let '( $\Gamma \vdash \Delta$ ) := s in
3   forall (M : structure  $\Sigma$ ), standard_model  $\Phi$  M -> forall ( $\rho$  : env M),
4     (forall  $\varphi$ , In  $\varphi \Gamma \rightarrow \rho \models \varphi$ ) -> exists  $\psi$ , In  $\psi \Delta \wedge \rho \models \psi$ .

```

Kažemo da vrijedi *lokalna adekvatnost* za pravilo izvoda ako istinitost (sekventi) premisa povlači istinitost sekvente konkluzije. Lokalna adekvatnost kaže da se istinitost čuva kod primjena pravila zaključivanja.

**Primjer 21.** Za pravilo *Ax* vrijedi lokalna adekvatnost, to jest ako u listama formula  $\Gamma$  i  $\Delta$  postoji neka zajednička formula  $\varphi$ , onda vrijedi  $\Gamma \models \Delta$ . Tada je *svjedok istinitosti* u  $\Delta$  upravo formula  $\varphi$ .

```

1 Lemma LS_Ax : forall  $\Gamma \Delta \varphi$ , In  $\varphi \Gamma \rightarrow$  In  $\varphi \Delta \rightarrow \Gamma \models \Delta$ .

```

Ako za svako pravilo izvoda vrijedi lokalna adekvatnost, indukcijom po strukturi dokaza slijedi i (globalna) adekvatnost. Glavni cilj ovog poglavlja bio je dokazati lokalne adekvatnosti svih pravila sustava *LKID*. Cilj je postignut za sva pravila osim pravila indukcije.

**Teorem 1.** Za svako pravilo izvoda sustava *LKID* koje nije induksijsko vrijedi lokalna adekvatnost. Posljedično, ako je sekventa dokaziva bez primjene induksijskih pravila, onda je ona istinita.

## 4. Ciklički dokazi

Svaki Coqov dokaz koji smo do sada prikazali je konačan term. Svi formalni dokazi u sustavu *LKID* (uključujući primjere 19 i 20) su konačni. Ovakve dokaze jednim imenom nazivamo *dokazima konačne dubine* jer su sve „grane” njihovih dokaznih stabala konačne. Međutim, možemo promatrati i *dokaze beskonačne dubine* kod kojih postoje beskonačne grane. Već smo u odjeljku 1.2. spomenuli koinduktivne tipove, a dokaz beskonačne dubine je tada „beskonačni” dokaz neke koinduktivno definirane tvrdnje. U ovom ćemo kratkom poglavlju ilustrirati dokaze beskonačne dubine te jednu njihovu posebnu formu: *cikličke* dokaze.

### 4.1. Primjer: lijene liste

Definirajmo za početak *lijene liste*, koje mogu biti konačne, a mogu biti i beskonačne. Zbog potencijalne beskonačnosti koristimo koinduktivnu definiciju tipa.

```
1 CoInductive LList (A : Type) :=  
2 | LNil : LList A  
3 | LCons : A -> LList A -> LList A.
```

Kažemo da je lijena lista *beskonačna* ako je njen rep opet beskonačna lijena lista.

```
1 CoInductive Infinite {A} : LList A -> Prop :=  
2 | Infinite_LCons : forall a l, Infinite l -> Infinite (LCons a l).
```

Ovdje konstruktor `Infinite_LCons` možemo shvatiti kao pravilo izvoda. Pokušajmo dokazati da je neka intuitivno beskonačna lista i formalno beskonačna. Definiramo funkciju `from` analogno istoimenoj funkciji iz odjeljka 1.2.

```
1 CoFixpoint from (n : nat) : LList nat := LCons n (from (S n)).
```

Intuitivno je jasno da je (za svaki  $n$ ) lijena lista `from n` beskonačna pa želimo dokazati

tvrdnju `forall n, Infinite (from n)`.

Prije glavnog dokaza, pogledajmo konkretan primjer na tvrdnji `Infinite (from 0)`. Po definiciji funkcije `from`, dovoljno je dokazati tvrdnju `Infinite (LCons 0 (from 1))`, a primjenom pravila `Infinite_LCons` dolazimo do tvrdnje `Infinite (from 1)`. Opet je po definiciji funkcije `from` dovoljno dokazati tvrdnju `Infinite (LCons 1 (from 2))`, a ponovnom primjenom pravila `Infinite_LCons` dolazimo do tvrdnje `Infinite (from 2)`. Ilustrirajmo ovaj postupak dokaznim stablom.

$$\begin{array}{c}
 \vdots \\
 \hline
 \text{Infinite (from 2)} \\
 \hline
 \text{Infinite (LCons 1 (from 2))} \quad (\text{Infinite\_LCons}) \\
 \hline
 \text{Infinite (from 1)} \\
 \hline
 \text{Infinite (LCons 0 (from 1))} \quad (\text{Infinite\_LCons}) \\
 \hline
 \text{Infinite (from 0)}
 \end{array}$$

Sada je jasno da ćemo dokazati originalnu tvrdnju tek nakon beskonačno mnogo primjena pravila `Infinite_LCons`. Ipak, lakše je dokazati tvrdnju `forall n, Infinite (from n)`. Nakon introdukcije varijable `n`, odmatanja definicije funkcije `from` i primjene pravila `Infinite_LCons` dolazimo do tvrdnje `Infinite (from (S n))`.

$$\begin{array}{c}
 * \\
 \hline
 \text{Infinite (from (S n))} \\
 \hline
 \text{Infinite (LCons n (from (S n)))} \quad (\text{Infinite\_LCons}) \\
 \hline
 \text{Infinite (from n)} \\
 \hline
 \text{forall n, Infinite (from n)}
 \end{array}$$

Između tvrdnji `Infinite (from n)` i `Infinite (from (S n))` „skinut” je prvi član beskonačne liste, čime je napravljen *napredak* u dokazu. Bez napretka nema smisla ponavljati prethodni postupak beskonačno mnogo puta jer se tvrdnja ne mijenja — u suprotnom, mogli bismo svaku tvrdnju  $\varphi$  dokazati trivijalnim dokazom  $\frac{*}{\varphi}$ . Uz napredak u dokazivanju ima smisla pozivati se na *tvrdnju koja se dokazuje* te takve dokaze nazivamo *cikličkima*. Dokaz tvrdnje `forall n, Infinite (from n)` jedan je primjer cikličkog dokaza, gdje je simbolom  $*$  označeno cikličko „pozivanje” dokaza. U Coqu, napredak je osiguran uvjetom produktivnosti spomenutim u odjeljku 1.5.

Pažljiv čitatelj će primijetiti da ista argumentacija vrijedi i za tvrdnju `Finite (from n)`, gdje je predikat `Finite` definiran na idući način.

```

1 Inductive Finite {A} : LList A -> Prop :=
2 | Finite_LNil : Finite LNil
3 | Finite_LCons : forall a l, Finite l -> Finite (LCons a l).

```

Međutim, beskonačne liste po definiciji nisu konačne. Zašto ipak gornji dokazi *ne* pro-  
laze za predikat *Finite*? Riječ je o načinu definicije; predikat *Infinite* definiran je  
koinduktivno te njegovi stanovnici *nisu* dobro utemeljeni, dok je predikat *Finite* defini-  
ran induktivno, a njegovi stanovnici *jesu* dobro utemeljeni. Štoviše, kada bismo predikat  
*Finite* definirali koinduktivno, on bi postao trivijalan, to jest vrijedio bi za proizvoljnu  
lijenu listu.

## 4.2. Primjer: $CLKID^\omega$

Brotherston je formalizirao ideju cikličkih dokaza u dokaznom sustavu  $CLKID^\omega$ , pod-  
sustavu dokaznog sustava  $LKID^\omega$ , koji je pak proširenje sustava  $LKID$  potencijalno be-  
skonačnim dokaznim stablima. Dokazni sustav  $CLKID^\omega$  ograničen je u odnosu na sus-  
tav  $LKID^\omega$  tako što svako (potencijalno beskonačno) dokazno stablo smije imati *najviše*  
*konačno mnogo* različitih podstabala te svaka beskonačna grana tog stabla mora imati  
beskonačno mnogo „napredaka”. Dokazni sustav  $CLKID^\omega$  prikazat ćemo neformalno.

**Primjer 22.** Dokazujemo tvrdnju  $\forall x, Nat(x) \rightarrow Even(x) \vee Odd(x)$  u sustavu  $CLKID^\omega$ .  
Koristimo pokrate kao u primjeru 20.

$$\begin{array}{c}
\frac{Nx \vdash Ex, Ox (\dagger)}{Ny \vdash Ey, Oy} (Subst) \\
\frac{Ny \vdash Ey, Oy}{Ny \vdash Oy, Ey} (Perm) \\
\frac{Ny \vdash Oy, Ey}{Ny \vdash Oy, Osy} (PA\_prod\_0\_succProd) \\
\frac{Ny \vdash Oy, Osy}{Ny \vdash E sy, O sy} (PA\_prod\_E\_succProd) \\
\frac{Ny \vdash E sy, O sy}{x = sy, Ny \vdash Ex, Ox} (EqL) \\
\frac{\vdash Eo, Oo}{\vdash Ex, Ox} (PA\_prod\_E\_zeroProd) \quad \frac{x = sy, Ny \vdash Ex, Ox}{x = sy, Ny \vdash Ex, Ox} (Case N) \\
\frac{Nx \vdash Ex, Ox (\dagger)}{Nx \vdash Ex \vee Ox} (OrR) \\
\frac{Nx \vdash Ex \vee Ox}{\vdash Nx \rightarrow Ex \vee Ox} (ImpR) \\
\frac{\vdash Nx \rightarrow Ex \vee Ox}{\vdash \forall x, Nx \rightarrow Ex \vee Ox} (AllR)
\end{array}$$

Iako zapisan u konačnom stablu, ovaj dokaz je beskonačan. Čitajući dokaz od dna do  
vrha, primjećujemo nove oznake te nova pravila. Za početak, vidimo da su znakom  $\dagger$   
označene gornja i donja sekventa  $Nx \vdash Ex, Ox$ . Gornju sekventu nazivamo *pupoljkom*  
(*bud*), a donju nazivamo *pratiteljem* (*companion*). Može se reći da su pupoljak i nje-

gov pratitelj povezani bridom, tvoreći ciklus u dokaznom stablu — odavde dolazi naziv *ciklički* dokaz. U stvari se isječak stabla dokaza između pratitelja i pupoljka ponavlja *ad infinitum* pa je dokaz beskonačne dubine. Nadalje, vidimo novo pravilo izvoda *Case* koje odgovara rastavu po slučajevima, a koristi se umjesto pravila indukcije. Ako se između pupoljka i njegovog pratitelja javlja primjena pravila *Case*, kažemo da je napravljen *napredak*. Konačno, vidimo znak jednakosti uz kojeg je vezano lijevo pravilo za jednakost *EqL*. Znak jednakosti je za potrebe sustava  $LKID^\omega$  i  $CLKID^\omega$  (odnosno pravila *Case*) ugrađen u definiciju formule.<sup>1</sup>

Ciklički dokazi su oni dokazi beskonačne dubine u kojima je između svakog pupoljka i njegovog pratitelja napravljen napredak, odnosno primijenjeno je pravilo *Case*. U primjeru 22, pupoljak i njegov pratitelj javljaju se u istoj grani dokaza, no to nije nužno. Ovu pojavu možemo vidjeti na dokazu obrata sekvente  $Nx \vdash Ex \vee Ox$ .

**Primjer 23.** Dokazujemo sekventu  $Ex \vee Ox \vdash Nx$  u sustavu  $CLKID^\omega$ .

$$\frac{\frac{\frac{\vdash No}{x = o \vdash Nx} (EqL) \quad \frac{\frac{\frac{Ox \vdash Nx (\dagger)}{Oy \vdash Ny} (Subst) \quad \frac{Oy \vdash Nsy}{Oy \vdash Nsy} (2)}{x = sy, Oy \vdash Nx} (EqL) \quad \frac{Ex \vdash Nx (*)}{Ex \vdash Nx (*)} (Case E)}{\frac{Ex \vdash Nx (*)}{Ex \vee Ox \vdash Nx} (OrL)} \quad \frac{\frac{\frac{Ex \vdash Nx (*)}{Ey \vdash Ny} (Subst) \quad \frac{Ey \vdash Nsy}{Ey \vdash Nsy} (3)}{x = sy, Ey \vdash Nx} (EqL) \quad \frac{Ox \vdash Nx (\dagger)}{Ox \vdash Nx (\dagger)} (Case O)}{\frac{Ex \vee Ox \vdash Nx}{Ex \vee Ox \vdash Nx} (OrL)}$$

Ovdje je znakom  $\dagger$  označen par pupoljka i pratitelja sekvente  $Ox \vdash Nx$ , a znakom  $*$  je označen par pupoljka i pratitelja sekvente  $Ex \vdash Nx$ . Iako se odgovarajući parovi pupoljaka i pratitelja naizgled javljaju u različitim granama, uz dva „odmatanja” stabla dokaza vidimo da to u stvari nije slučaj.

$$\frac{\frac{\frac{\vdash No}{x = o \vdash Nx} (EqL) \quad \frac{\frac{\frac{Ex \vdash Nx (*)}{Ey \vdash Ny} (Subst) \quad \frac{Ey \vdash Nsy}{Ey \vdash Nsy} (2)}{x = sy, Ey \vdash Nx} (EqL) \quad \frac{Ox \vdash Nx (\dagger)}{Ox \vdash Nx (\dagger)} (Case O)}{\frac{Ex \vdash Nx (*)}{Ex \vdash Nx (*)} (Subst) \quad \frac{Oy \vdash Ny}{Oy \vdash Nsy} (2)} \quad \frac{\frac{\vdash No}{x = o \vdash Nx} (EqL) \quad \frac{\frac{\frac{Ox \vdash Nx (\dagger)}{Oy \vdash Ny} (Subst) \quad \frac{Oy \vdash Nsy}{Oy \vdash Nsy} (2)}{x = sy, Oy \vdash Nx} (EqL) \quad \frac{Ex \vdash Nx (\dagger)}{Ex \vdash Nx (\dagger)} (Case E)}{\frac{Ex \vee Ox \vdash Nx}{Ex \vee Ox \vdash Nx} (OrL)}$$

Pupoljak sekvente  $Ex \vdash Nx$  zamijenjen je isječkom stabla od pratitelja sekvente  $Ex \vdash Nx$  do pupoljka sekvente  $Ox \vdash Nx$ , Nadalje, pupoljak sekvente  $Ox \vdash Nx$  zamijenjen je

<sup>1</sup>Jednakost se u matematičkoj logici najčešće tretira kao predikatni simbol signature.

isječkom stabla od pratitelja sekvente  $Ox \vdash Nx$  do pupoljka sekvente  $Ex \vdash Nx$ . Dokaz ne bi bio moguć bez korištenja produkcijskih pravila za produkcije:

- (1) PA\_prod\_N\_zero i
- (2) PA\_prod\_N\_succ.

Važno je spomenuti da je svojstvo „biti ispravan dokaz u sustavu  $CLKID^\omega$ ” odlučivo te je na temelju toga napravljen ciklički dokazivač teorema CYCLIST<sup>2</sup> [19]. Za naš primjer proširene Peanove signature s produkcijama za induktivne predikatne simbole *Nat*, *Even* i *Odd*, definicije produkcija možemo zapisati u konfiguracijsku datoteka `fo.defs` na idući način.

```
Nat {
    true      => Nat(0) |
    Nat(x)    => Nat(s(x))
} ;
Even {
    true      => Even(0) |
    Odd(x)    => Even(s(x))
} ;
Odd {
    Even(x)   => Odd(s(x))
} ;
```

Zatim možemo pokrenuti dokazivač naredbom

```
./fo_prove -D fo.defs -p -S "Nat(x) |- Even(x) \/ Odd(x)"
```

i dobiti izlaz programa

```
0: Nat_0(x) |- Odd_2(x) \/ Even_1(x) (Nat L.Unf.) [1 <{(0, 1)}/{>, 2 <{(0, 1)}/{(0, 0)}>]
1: Nat_1(0) |- Odd_2(0) \/ Even_1(0) (Even R.Unf.) [3 <{(1, 1)}/{>]
2: Nat_1(0) |- T \/ Odd_2(0) (Id)
3: Nat_1(s(w)) /\ Nat_0(w) |- Odd_2(s(w)) \/ Even_1(s(w)) (Even R.Unf.) [4 <{(1, 1), (0, 0)}/{>]
4: Nat_1(s(w)) /\ Nat_0(w) |- Odd_4(w) \/ Odd_2(s(w)) (Odd R.Unf.) [5 <{(1, 1), (0, 0)}/{>]
5: Nat_1(s(w)) /\ Nat_0(w) |- Even_5(w) \/ Odd_4(w) (Weaken) [6 <{(0, 0)}/{>]
6: Nat_0(w) |- Odd_2(w) \/ Even_1(w) (Subst) [7 <{(0, 0)}/{>]
7: Nat_0(x) |- Odd_2(x) \/ Even_1(x) (Back1) [0] <pre={ (0, 0) }>
```

koji predstavlja dokaz sekvente  $Nat(x) \vdash Even(x) \vee Odd(x)$ . Možemo primijetiti da je ovaj generirani dokaz sličan našem dokazu iz primjera 22.

---

<sup>2</sup><http://www.cyclist-prover.org/>



## Zaključak

Cilj ovog rada bio je ponuditi pregled visoke razine na alat za dokazivanje Coq te ga primijeniti na formalizaciju logike prvog reda s induktivnim definicijama. U tu svrhu smo u prvom dijelu rada prikazali osnove Coqa i teorije na kojoj se on temelji, dok smo ostatak rada posvetili logici. Definirali smo osnovne pojmove poput formule, strukture, standardnog modela te dokaznog sustava i dokazali smo neka njihova svojstva. Kroz cijeli rad ilustrirali smo definicije i leme neformalnim i formaliziranim primjerima. Iako je ponekad bilo teže razlučiti bitne dijelove neformalne definicije, ipak se pokazalo da je Coq prikladan alat za formalizaciju logike te autor smatra da je cilj rada uspješno postignut.

Ovim radom smo tek zagrebali površinu formalizacije logike prvog reda s induktivnim definicijama. Prirodni nastavak rada je dokaz potpunosti sustava *LKID* pomoću Henkinovih modela, a nakon toga i formalizacija cikličkog dokaznog sustava *CLKID*<sup>ω</sup>. Nadalje, ima smisla formalizirati dokazivače teorema za sustave *LKID* i *CLKID*<sup>ω</sup>. Kako logika prvog reda nije odlučiva, takvi dokazivači teorema trebali bi uvesti dodatne pretpostavke (kao što je pretpostavka zatvorenog svijeta u Prologu) ili se usmjeriti prema interaktivnom dokazivanju. Za našu formalizaciju sustava *LKID*, ima smisla iskoristiti dodatne Coqove mogućnosti, posebno za automatizaciju dokaza jezikom taktika Ltac te za prikladniju notaciju terma, formula i dokaza.

Po autorovu mišljenju, računalna formalizacija matematike je odličan način razmišljanja i razumijevanja, posebno kada se koristi zajedno s tradicionalnim učenjem jer tada pojam koji se proučava postaje jasan u potpunosti — formalizacija nas tjera da budemo posebice pažljivi. Nadalje, u današnje vrijeme umjetne inteligencije i automatski generiranog koda, dokazivanje točnosti programa postaje bitnije no što je ikad bilo. Za kraj ćemo reći da je formalizacija proces koji je često naporan ali uvijek ispunjujuć.

## Literatura

- [1] Y. Bertot i P. Castéran, *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, 1. izd. Springer Berlin, 2010.
- [2] B. C. Pierce *et al.*, *Logical Foundations*. Elektronički udžbenik, 2024., verzija 6.5, <http://softwarefoundations.cis.upenn.edu>.
- [3] —, *Programming Language Foundations*. Elektronički udžbenik, 2024., verzija 6.5, <http://softwarefoundations.cis.upenn.edu>.
- [4] A. W. Appel, *Verified Functional Algorithms*. Elektronički udžbenik, 2023., verzija 1.5.4, <http://softwarefoundations.cis.upenn.edu>.
- [5] A. Chlipala, *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*, 1. izd. MIT Press, 2022.
- [6] The Coq Development Team, “The Coq Reference Manual, Release 8.18.0”, <https://coq.inria.fr/doc/v8.18/refman/>, 2023., pristupljeno: 25.06.2024.
- [7] T. Coquand i G. Huet, “The Calculus of Constructions”, *Information and Computation*, sv. 76, br. 2, str. 95–120, 1988.
- [8] F. Pfenning i C. Paulin-Mohring, “Inductively defined types in the Calculus of Constructions”, u *Proceedings of the 5th International Conference on Mathematical Foundations of Programming Semantics*, 1989., str. 209–228.
- [9] E. Giménez, “Codifying guarded definitions with recursive schemes”, u *Types for Proofs and Programs*, 1995., str. 39–59.

- [10] M. Sozeau, S. Boulrier, Y. Forster, N. Tabareau, i T. Winterhalter, “Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq”, *Proceedings of the ACM on Programming Languages*, sv. 4, br. POPL, 2019.
- [11] G. Hutton, *Programming in Haskell*, 2. izd. Cambridge University Press, 2016.
- [12] J. Brotherston, “Cyclic Proofs for First-Order Logic with Inductive Definitions”, u *Automated Reasoning with Analytic Tableaux and Related Methods*, 2005., str. 78–92.
- [13] —, “Sequent calculus proof systems for inductive definitions”, doktorska disertacija, School of Informatics, University of Edinburgh, 2006.
- [14] M. Vuković, *Matematička logika*, 1. izd. Element, 2009.
- [15] N. G. de Bruijn, “Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem”, *Indagationes Mathematicae (Proceedings)*, sv. 75, br. 5, str. 381–392, 1972.
- [16] B. C. Pierce, *Types and Programming Languages*, 1. izd. MIT Press, 2002.
- [17] K. Stark, “Mechanising Syntax with Binders in Coq”, doktorska disertacija, Faculty of Mathematics and Computer Science, Saarland University, 2020.
- [18] K. Stark, S. Schäfer, i J. Kaiser, “Autosubst 2: Reasoning with Multi-Sorted de Bruijn Terms and Vector Substitutions”, u *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2019., str. 166–180.
- [19] J. Brotherston, N. Gorogiannis, i R. L. Petersen, “A Generic Cyclic Theorem Prover”, u *Programming Languages and Systems*, 2012., str. 350–367.

# Sažetak

## Primjene Coq alata za dokazivanje u matematici i računarstvu

Miho Hren

Coq je interaktivni dokazivač teorema temeljen na teoriji tipova. U ovom radu primijenili smo alat Coq na zadatak formalizacije logike prvog reda s induktivnim definicijama  $FOL_{ID}$ . Nadalje, definirali smo standardne modele i dokazni sustav  $LKID$  za logiku  $FOL_{ID}$  te dokazali neka njihova svojstva. Također smo neformalno prikazali cikličke dokaze u kontekstu logike  $FOL_{ID}$ . Sve definicije i leme u radu ilustrirali smo primjerima te formalizacijom u Coqu.

**Ključne riječi:** Coq; logika prvog reda; induktivne definicije; dokaz

# Abstract

## Applications of the Coq Proof Assistant in mathematics and computer science

Miho Hren

Coq is an interactive theorem prover based on type theory. In this thesis we applied the Coq Proof Assistant to the task of formalizing first order logic with inductive definitions  $FOL_{ID}$ . Furthermore, we defined standard models and the  $LKID$  proof system for  $FOL_{ID}$ , and we proved some of their properties. We also informally presented cyclic proofs in the context of  $FOL_{ID}$ . We illustrated all the definitions and lemmas in this thesis with examples and formalizations in Coq.

**Keywords:** Coq; first order logic; inductive definitions; proof