Programirati znači dokazati

Andrej Ivašković

Matematička gimnazija

NEDELJA$^4_{\text{INFORMATIKE}}$

29. mart 2018.

Pojam *proof assistant*

- Jedna neobična i zanimljiva oblast matematike i teorijskog računarstva se tiče formalizacije matematike i dokaza.
- Dokazi koje matematičari pišu uglavnom nisu formalni, ali jesu rigorozni. Greške mogu da se potkradu!
- Postoje programski jezici sa specifičnom namenom da omoguće matematičarima da budu sigurni da su njihovi dokazi zapravo formalni i potpuni, među kojima su Agda, Coq i slični.
- Ovakvi jezici se nazivaju *proof assistants*.

Tipovi u proof assistant jezicima

- Ono što je uočljivo kod jezika poput Agda jeste jako složen i moćan **sistem tipova**.
- Tipovi o kojima je reč nisu prosti brojevi ili samo nizovi, već daleko fleksibilniji i moćniji sistemi.
- Programeri imaju ogromnu fleksibilnost pri definisanju sopstvenih tipova.

Kari–Hauard izomorfizam

- Ovi *proof assistant* jezici imaju izuzetno moćne i fleksibilne sisteme tipova zbog toga što se ispostavlja da postoji veoma prirodna i jasna analogija (zapravo bijekcija!) između tipova i teorema – **Kari–Hauard izomorfizam**.

- Osnovna ideja ove veze se ogleda u tome da tip nekog izraza (ili programa) odgovara nekom tvrđenju, a izraz tog tipa je primer dokaza tog istog tvrđenja.

- Veza je između **teorije tipova** i takozvane **konstruktivne logike**, te će iz dokaza moći i da se konstruišu programi.

Plan predavanja

- Najpre ćemo se baviti idejama **konstruktivne logike**.
- Nakon toga ćemo posmatrati pojam **induktivnog sistema** i videti kako je konstruktivna logika modelovana jednim takvim sistemom.
- Nakon toga ćemo videti kako **teorija tipova** takođe može da se modeluje induktivnim sistemom.
- Odavde će Kari-Hauardov izomorfizam doći "prirodno".
- Nećemo posmatrati nijedan proof assistant jezik.

Šta znači konstruktivnost?

- U matematici postoje **konstruktivni dokazi**, koji dokazuju postojanje nekih objekata ($\exists x$) eksplicitnom konstrukcijom.
- Ovi dokazi, dakle, ne koriste identitet $(\exists x)(P(x)) \equiv \neg(\forall x)(\neg P(x))$ i suprotne pretpostavke.
- **Konstruktivna logika** je inspirisana ovim idejama (nećemo moći da se oslonimo na pravila duple negacije i zakona isključenja trećeg).

Primer nekonstruktivnog dokaza

- Postoje iracionalni brojevi a i b takvi da je a^b racionalan.
- Dokaz:
  - Posmatrajmo $a = \sqrt{2}$ i $b = \sqrt{2}^{\sqrt{2}}$.
  - a je iracionalan, a b je ili racionalan ili nije.
  - Ako je b racionalan, onda je tvrđenje već dokazano.
  - Ako je b iracionalan, tada je

    $$b^a = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = (\sqrt{2})^{\sqrt{2}\cdot\sqrt{2}} = (\sqrt{2})^2 = 2$$ racionalan.

  - Dakle, ovakvi brojevi postoje (ili $\sqrt{2}$ i $\sqrt{2}$ ili $\sqrt{2}^{\sqrt{2}}$ i $\sqrt{2}$)
- Primetimo da naš dokaz nema eksplicitno konstruisan primer!

Primer konstruktivnog dokaza

- Postoje iracionalni brojevi a i b takvi da je a^b racionalan.
- Dokaz:
  - Posmatrajmo $a = \sqrt{2}$ i $b = 2 \log_2 3$.
  - I a i b su iracionalni, što može da se pokaže konstruktivnim dokazima.
  - Međutim, $a^b = \left(\sqrt{2}\right)^{2 \log_2 3} = 2^{\log_2 3} = 3$, što je racionalan broj.
  - Dakle, ovakvi brojevi a i b postoje.
- U konstruktivnoj matematici postoji razlika između iracionalnog broja i realnog broja koji nije racionalan – nećemo se baviti ovim.

Konstruktivna logika

- Konstruktivna logika je slična logici sa kojom već imamo iskustva, ali neka pravila više ne važe.
- Pojam negacije postoji, ali neka očekivana svojstva više ne važe.
- **Zakon isključenja trećeg:** $p \vee \neg p$
- **Dupla negacija:** $p \leftrightarrow \neg\neg p$
- Naša interpretacija logičkih veznika se sada značajno menja.

Konstruktivna interpretacija logičkih veznika

- Dokaz **implikacije** $\phi \to \psi$ je sada konstrukcija koja, polazeći od dokaza za ϕ, formira dokaz za ψ.
- Dokaz **konjunkcije** $\phi \wedge \psi$ je konstrukcija koja je uređeni par dokaza za ϕ i dokaza za ψ.
- Dokaz **disjunkcije** $\phi \vee \psi$ je konstrukcija koja je dokaz ϕ ili za ψ.
- Dokaz **negacije** $\neg\phi$ je sada konstrukcija koja, polazeći od nekog (hipotetičkog) dokaza za ϕ, izvodi kontradikciju ($\bot$).
- Dokaz **univerzalne kvantifikacije** $(\forall x)(P(x))$ je konstrukcija koja konstruiše sve objekte a koji mogu da budu vrednost x u dokaze za $P(a)$.
- Dokaz **egzistencijalne kvantifikacije** $(\exists x)(P(x))$ je konstrukcija koja konstruiše uređeni par nekog eksplicitnog objekta a koji može da bude vrednost x i dokaza za $P(a)$.

Formalni sistem zaključivanja

- Posmatraćemo logiku na formalan način – ne postoje tablice istinosti, već *izvodimo* tačne formule koristeći isključivo aksiome i jasna **pravila zaključivanja**.
- Zato koristimo $\vdash \phi$ da bismo naglasili da neka logička formula Φ može da se izvede bez dodatnih pretpostavki.
- Ako je skup pretpostavki Φ, tada $\Phi \vdash \phi$ znači da se iz tog skupa pretpostavki može izvesti Φ. Na primer, $p, q \vdash p \wedge q$ (ne pišemo zagrade radi jednostavnosti).

Pravila zaključivanja

- Pravila zaključivanja (i kada su opšta i kada ih instanciramo) pišemo kao "razlomke", gde su premise iznad crte, a zaključak ispod:

$$\frac{\text{svi } A \text{ su } B \quad X \text{ je } A}{X \text{ je } B}$$

$$\frac{\text{svi ljudi su smrtni} \quad \text{Aristotel je čovek}}{\text{Aristotel je smrtan}}$$

- Ovo rezultuje dokazima koji imaju oblik stabla:

$$\dfrac{\dfrac{\text{sve životinje su smrtne} \quad \text{svi ljudi su životinje}}{\text{svi ljudi su smrtni}} \quad \text{Aristotel je čovek}}{\text{Aristotel je smrtan}}$$

Deo formalnog sistema konstruktivne logike

$$\frac{}{\Phi \vdash \phi} \quad \text{ako } \phi \in \Phi$$

$$\frac{\Psi \vdash \phi}{\Phi \vdash \phi} \quad \text{ako } \Psi \subseteq \Phi$$

$$\frac{}{\phi \wedge \psi \vdash \phi}$$

$$\frac{}{\phi \wedge \psi \vdash \psi}$$

$$\frac{\Phi \vdash \phi \quad \Phi \vdash \psi}{\Phi \vdash \phi \wedge \psi}$$

Deo formalnog sistema konstruktivne logike

$$\frac{\Phi, \phi \vdash \psi}{\Phi \vdash \phi \to \psi}$$

$$\frac{\Phi \vdash \phi \to \psi \quad \Phi \vdash \phi}{\Phi \vdash \psi}$$

$$\frac{\Phi \vdash \phi}{\Phi \vdash (\forall p)(\phi)}$$ ako se p ne pojavljuje u ϕ

$$\frac{\Phi \vdash (\forall p)(\phi)}{\Phi \vdash \phi[\psi/p]}$$ gde je ψ bilo koja formula

Kuda dalje?

- Ova pravila su navedena samo da bi ilustrovala kako se koristi neki formalan sistem – zaključivanje brzo postaje nepregledno.
- Ispostavlja se da su jedini neophodni veznici $\forall$ i $\rightarrow$, sve ostalo možemo da predstavimo preko njih (uključujući i $\top$ i $\bot$).
- Pogledaćemo sada rigorozan (ne formalan) dokaz $\vdash p \wedge (q \vee r) \rightarrow (p \wedge q) \vee (p \wedge r)$.

$\vdash p \wedge (q \vee r) \to (p \wedge q) \vee (p \wedge r)$

- Dovoljno je dokazati $p \wedge (q \vee r) \vdash (p \wedge q) \vee (p \wedge r)$.
- Jasno je da $p \wedge (q \vee r) \vdash p$ i $p \wedge (q \vee r) \vdash q \vee r$.
- Posmatramo dva moguća slučaja:
  - Ako $p \wedge (q \vee r) \vdash q$, tada na osnovu prethodnog važi $p \wedge (q \vee r) \vdash p \wedge q$.

    Dakle, $p \wedge (q \vee r) \vdash (p \wedge q) \vee (p \wedge r)$.
  - Ako $p \wedge (q \vee r) \vdash r$, tada na osnovu prethodnog važi $p \wedge (q \vee r) \vdash p \wedge r$.

    Dakle, $p \wedge (q \vee r) \vdash (p \wedge q) \vee (p \wedge r)$.
- Time je ovo tvrđenje dokazano.

Da li ovo liči na program?

- Primetimo da ovaj dokaz ima pretežno linijsku strukturu, uz jedno grananje.
- Jasno su definisani nekakvi koraci u dolasku do svih zaključaka.
- Postoje eksplicitne konstrukcije za svaki korak, nigde nije korišćena nikakva negacija ili suprotna pretpostavka.
- Ovo je kao program u sred kog je nalazi jedan IF koji razdvaja dva slučaja.
- Vratićemo se ovome malo kasnije!

Teorija tipova

- **Teorija tipova** je oblast teorijskog računarstva koja se bavi sistemima tipova u programskim jezicima i algoritmima provere i otkrivanja tipova.
- Želimo da nam sistem tipova premesti veliki broj problema u toku izvršavanja na vreme kompajliranja.
- Ova oblast je posebno značajna u funkcionalnim programskim jezicima (poput OCaml, Haskell).
- Mi ćemo da se fokusiramo na proceduralan jezik čija sintaksa podseća na C i izmišljen je za potrebe ovog predavanja.

Programski jezik

- Programski jezik će sintaksno podsećati na C++ bez pokazivača, ulazno-izlaznih rutina i bilo čega posebno komplikovanog.
- Argumenti se prosleđuju po vrednosti (nikada referenci).
- Kastovanje više ne postoji, tipovi su strogi.
- Neke stvari dodajemo. . .

Unije

- Zadržavamo **unije** iz C++, gde jedna promenljiva može da sadrži podatak jednog od nekoliko različitih tipova.
- Menjamo sintaksu i način upotrebe: dodajemo ključnu reč `match` koja vrši grananje koje zavisi od toga koji je tip promenljive tipa unije; drugačije vršimo dodelu.

```
union Data {
  I: int i;
  C: char c;  };
.................................
Data x = Data.int(7);
.................................
match (x) {
  I: x.i = 42;
  C: x.c = 'p'; }
```

Parovi

- U STL postoji `std::pair<S,T>` za uređene parove, i mi ćemo imati `pair<S,T>` tipovnu konstrukciju.

- Dodajemo `.first` i `.second`, a vrednosti zapisujemo kao `x,y`.

```
pair<int,char> x = {1, 'c'};
x.first = 42;
pair<char,char> y = {y.second, 'x'};
```

Parametarski polimorfizam

- Od **templejtova** zadržavamo samo mogućnost prosleđivanja tipova, izbacujemo `class`.
- Ovo nam omogućava pisanje polimorfnih funkcija, ali i polimorfnih unija.

```
template<T> T ident(T x) { return x; }
template<T> pair_up<int,T> f(T a) { ... }
.................................
template<T>
union Tree {
  Leaf:   T val;
  Branch: pair<T, pair<Tree<T>, Tree<T> > > node;
};
.................................
f<char>('a');
```

Konstruisanje funkcija

- Postoji i eksplicitan tip funkcija: na primer, ako `f` uzima `int` i `char` argument, a vraća `int`, tip ćemo zapisivati kao `int(int,char)`.

- Možemo da konstruišemo funkcijske vrednosti kojima ne dajemo eksplicitno ime pomoću `[]`.

```
int f(int x) { return x + 1; }
.....................................
int(int) g = f;
int(int,int) h = [](int x, int y){ return x + y; }
```

Rezonovanje o tipovima

- Dosta smo se obezbedili – nemamo mogućnost *segmentation fault*, *memory leak* i sličnih stvari.
- Sada ćemo se upoznati sa načinom da **formalno rezonujemo o tipovima izraza**, gde **izraz** podrazumeva i aritmetičke i logičke izraze, ali i nizove naredbi.
- Ispostavlja se da je i ovo formalan, induktivno definisan sistem.

Trojka rezonovanja o tipu izraza

- Za neki izraz e i tip τ pišemo $e : \tau$ ako e ima tip τ. Na primer, $3 : \texttt{int}$.
- Međutim, često postoje promenljive u nekom izrazu, i onda konačan tip i validnost izraza zavise od tipa te promenljive. Na primer, `{ x, 3 }` ima tip `pair<int,int>` (dalje $\texttt{int} \times \texttt{int}$) ako je `x` tipa `int`, a može da bude i `pair<char,int>` (dalje $\texttt{char} \times \texttt{int}$) ako je `x` tipa `char`.
- Uvodimo pojam **konteksta** Γ, funkcije iz skupa promenljivih u skup svih tipova (kome pridružujemo i `undef`). Pišemo $\Gamma \vdash e : \tau$.
- Dakle, $\texttt{x} : \texttt{char} \vdash \texttt{\{ x, 3 \}} : \texttt{char} \times \texttt{int}$.

Deo formalnog sistema tipova: osnove

$$\frac{}{\Gamma \vdash x : \tau} \quad \text{ako } \Gamma(x) = \tau$$

$$\frac{}{\Gamma \vdash n : \texttt{int}} \quad n \in \mathbb{Z}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \cup \Delta \vdash e_2 : \tau_2}{\Gamma \vdash e_1; e_2 : \tau_2} \quad \text{gde je } \Delta \text{ nova definicija u } e_1$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Deo formalnog sistema tipova: parovi

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \{e_1,\ e_2\} : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.\mathtt{first} : \tau_1}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.\mathsf{second} : \tau_2}$$

Za unije uvodimo tip $\tau_1 + \tau_2$ i ova pravila izgledaju malo ružnije u ovom jeziku.

Deo formalnog sistema tipova: funkcije

$$\frac{\Gamma, x_1 : \tau_1, \ldots, x_k : \tau_k \vdash e : \tau}{\Gamma \vdash \texttt{[]}(\tau_1\ x_1, \ \ldots, \ \tau_k\ x_k)\{e\} : \tau_1 \times \ldots \times \tau_k \to \tau}$$

$$\frac{\Gamma \vdash f : \tau_1 \times \ldots \times \tau_k \to \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_k : \tau_k}{\Gamma \vdash f(e_1, \ \ldots, \ e_k) : \tau}$$

Deo formalnog sistema tipova: polimorfizam

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{template<}\alpha\texttt{>}\ e : (\forall \alpha)(\tau)}$$

$$\frac{\Gamma \vdash e : (\forall \alpha)(\tau)}{\Gamma \vdash e\texttt{<}\tau'\texttt{>} : \tau[\tau'/\alpha]}$$

Kako koristimo ova pravila?

- Nije teško pokazati ovim sistemom da, na primer, prethodno uvedena `pair_up` funkcija ima tip $(\forall\alpha)(\alpha \to \texttt{int} \times \alpha)$.
- Provera da li su svi tipovi odgovarajući zato nije teška i može se implementirati u kompajleru.
- Sva pravila su **sintaksne prirode**.

Sličnosti između implikacija i funkcija

$$\frac{\Phi, \phi \vdash \psi}{\Phi \vdash \phi \to \psi}$$

$$\frac{\Gamma, x_1 : \tau_1, \ldots, x_k : \tau_k \vdash e : \tau}{\Gamma \vdash \texttt{[]}(\tau_1\ x_1, \ \ldots, \ \tau_k\ x_k)\{e\} : \tau_1 \times \ldots \times \tau_k \to \tau}$$

$$\frac{\Phi \vdash \phi \to \psi \quad \Phi \vdash \phi}{\Phi \vdash \psi}$$

$$\frac{\Gamma \vdash f : \tau_1 \times \ldots \times \tau_k \to \tau \quad \Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_k : \tau_k}{\Gamma \vdash f(e_1, \ \ldots, \ e_k) : \tau}$$

Sličnosti između konjunkcija i parova

$$\frac{}{\phi \wedge \psi \vdash \phi} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.\mathtt{first} : \tau_1}$$

$$\frac{}{\phi \wedge \psi \vdash \psi} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e.\mathtt{second} : \tau_2}$$

$$\frac{\Phi \vdash \phi \quad \Phi \vdash \psi}{\Phi \vdash \phi \wedge \psi} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \{e_1,\ e_2\} : \tau_1 \times \tau_2}$$

Kari–Hauard izomorfizam

- Ispostavlja se da ove sličnosti nisu slučajne – ova veza između tipova i teorema naziva se **Kari–Hauard izomorfizam** (Curry–Howard isomorphism/correspondence).
- Kako tipovima odgovaraju teoreme, tako se ispostavlja da se izrazi nekog tipa zapravo model dokaza neke teoreme.
- Dakle, dokazati teoremu u konstruktivnoj logici znači napisati izraz odgovarajućeg tipa. Važi i obratno!

Prevod

- Implikacije odgovaraju funkcijama.
- Konjunkcije odgovaraju uređenim parovima.
- Disjunkcije odgovaraju unijama.
- Dokazivanje $\forall$ tvrđenja odgovara pisanju polimorfnog izraza.
- Ovo nam maltene direktno daje program – možemo da vidimo ovo na primeru $\vdash p \wedge (q \vee r) \to (p \wedge q) \vee (p \wedge r)$, ako imamo odgovarajuće tipove `P`, `Q`, `R`.

Program koji je i dokaz

```
template<S,T> union Disj {
  Fst: S l;
  Snd: T r;
};

Disj<pair<P,Q>, pair<P,R> > f(pair<P, Disj<Q,R> > x) {
  P a = x.first;
  Disj<Q,R> b = x.second;
  match (b) {
    Fst: return Fst({a, b.l});
    Snd: return Snd({a, b.r});
  }
}
```

Uporediti sa dokazom koji smo imali ranije!

Šta sve možemo da dokažemo?

- Sve što možemo da dokažemo u konstruktivnoj logici ima odgovarajući program.
- Kako bismo, na primer, napisali program koji je dokaz za $(\forall p, q, r)((p \vee q) \vee r \to p \vee (q \vee r))$?
- A za $(\forall p, q, r)((p \to (q \to r)) \to ((p \to q) \to (p \to r)))$?
- Ispostavlja se da je $\top$ zapravo tip $(\forall \alpha)(\alpha \to \alpha)$, a $\bot$ je $(\forall \alpha)(\alpha)$. Zašto?
- Setimo se: $\neg\phi \equiv \phi \to \bot$.
- Zašto onda ne možemo da dokažemo $\neg\neg p \to p$?

Šta je sa složenijim programima i jezicima?

- Od dokaza napraviti program nije teško.
- Za silne programe je jasno šta dokazuju – dovoljno je odrediti izraza.
- Ako u programskom jeziku imamo složenije konstrukcije, onda modeluje neke druge (i dalje konstruktivne) logike.

Zavisni tipovi

- Postoje sistemi tipova (i programski jezici) koji omogućavaju da povratna vrednost funkcije ima tip koji zavisi od argumenata.
- Na primer funkcija `ntorka` uzima za argument prirodan broj n i vraća uređenu n-torku $(1, 1, 1, \ldots, 1)$, tipa $\texttt{int} \times \ldots \times \texttt{int}$ (n puta `int`).
- Ovo je veoma korisno u raznim proof assistant jezicima.

Zaključak

- Postoji duboka i jasna veza između kontruktivne logike i programa.
- Teoreme odgovaraju tipovima, dokazi tih teorema odgovaraju izrazima tih tipova.
- Zbog ovoga je posao formalizacije matematike zapravo zajednički posao programera i matematičara.