Ovo je zahvala.

Sadržaj

1. Uvod

2. Coq

U ovom poglavlju dajemo pogled svisoka na programski sustav 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].

2.1. Što je Coq?

Alat za dokazivanje Coq[1], punog naziva *The Coq Proof Assistant*, programski je sustav pomoću kojeg korisnici mogu dokazivati matematičke tvrdnje. **Misao:** *ne sluzi samo tome, moze biti i opceniti funkcijski programski jezik, moze sluziti za programiranje sa zavisnim tipovima* Alat se temelji na λ-računu i teoriji tipova, a prva je inačica implementirana godine 1984. [6] Ovaj rad koristi inačicu 8.18 iz rujna godine 2023.

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 `bash` i `python` ljuskama. Interaktivna ljuska (također poznata pod imenom *toplevel*) služi unosu definicija i iskazivanju lema. Skupni način rada pokreće se naredbom `coqc`, a korisniku omogućuje semantičku provjeru i prevođenje izvornih datoteka u jednostavnije formate. Kod formaliziranja i dokazivanja, korisnik će najčešće koristiti interaktivni način rada, po mogućnosti kroz neku od dostupnih razvojnih okolina. [2]

Misao: *tu negdje treba spomenuti proof mode, koji je zaseban od toplevela*

[1] `https://coq.inria.fr/`

[2] Autor rada koristio je paket *Proof General* za uređivač teksta *Emacs*. Druge često korištene okoline su *VsCoq* i *CoqIDE*.

Kao programski jezik, Coq se sastoji od više podjezika različitih namjena, od kojih spominjemo *Vernacular*, *Gallinu* i *Ltac*.

Vernacular **Misao:** *vernacular znači "govorni jezik"* je jezik naredbi kojima korisnik komunicira sa sustavom (i u interaktivnom i u skupnom načinu rada); svaka Coq skripta (datoteka s nastavkom `.v`) je niz naredbi. Neke od najčešće korištenih naredbi su `Check`, `Definition`, `Inductive`, `Fixpoint` i `Lemma`. Pomoću naredbi za tvrdnje, kao što je `Lemma`, Coq prelazi iz *toplevela* **Misao:** *treba bolji prevod* u način dokazivanja (engl.*proof mode*).

Gallina je Coqov strogo statički tipiziran specifikacijski jezik. Kako se glavnina programiranja u Coqu svodi upravo na programiranje u Gallini, posvećujemo joj idući odjeljak.

Ltac je Coqov netipiziran 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 ilustrativan primjer.

```
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 programskog sustava Coq je algoritam za provjeru tipova (engl.*type checking*) implementiran u OCamlu — svaka tvrdnja koja se dokazuje izrečena je pomoću tipova. Ostatak sustava u načelu služi za knjigovodstvo i poboljšanje korisničkog iskustva. 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? **Misao:** *kažem istinite, ali u stvari mislim dokazive, no to je nespretno za napisati i diskusija oko*

toga je preopćenita

Prve inačice Coqa implementirale su račun konstrukcija, no kasnije je dodana podrška za induktivno i koinduktivno definirane tipove [7, 8]. Danas se može reći da Coq implementira polimorfni kumulativni račun induktivnih konstrukcija [9]. **Misao:** *mogu spomenuti i preteče računa konstrukcija i jezike koji ih implementiraju, npr. Lisp je implementacija λ-računa* Coq se, osim kao dokazivač teorema, može koristiti i za programiranje sa zavisnim tipovima. U toj sferi konkuriraju jezici Agda[3], Idris[4] i Lean[5]. 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).

2.2. Programiranje u Gallini

Gallina je funkcijski programski jezik ugrađen u Coq, što znači da su funkcije prvoklasni objekti — funkcije mogu biti argumenti i povratne vrijednosti drugih funkcija. Dodatno, varijable su nepromjenjive (engl.*immutable*) te se iteracija ostvaruje rekurzijom. Za uvod u funkcijsko programiranje, čitatelja upućujemo na knjigu *Programming in Haskell* [10]. Gallina je strogo statički tipiziran jezik, što znači da se svakom termu prilikom prevođenja dodjeljuje tip. U općenitom smislu, tip je kolekcija sličnih objekata. Tip nekog terma možemo provjeriti (Vernacular) naredbom `Check`.

Funkcijsko programiranje neki jednostavni primjer

Definiranje funkcija `Definition`, `Fixpoint`, `CoFixpoint`

Definiranje tipova `Inductive`, `CoInductive`

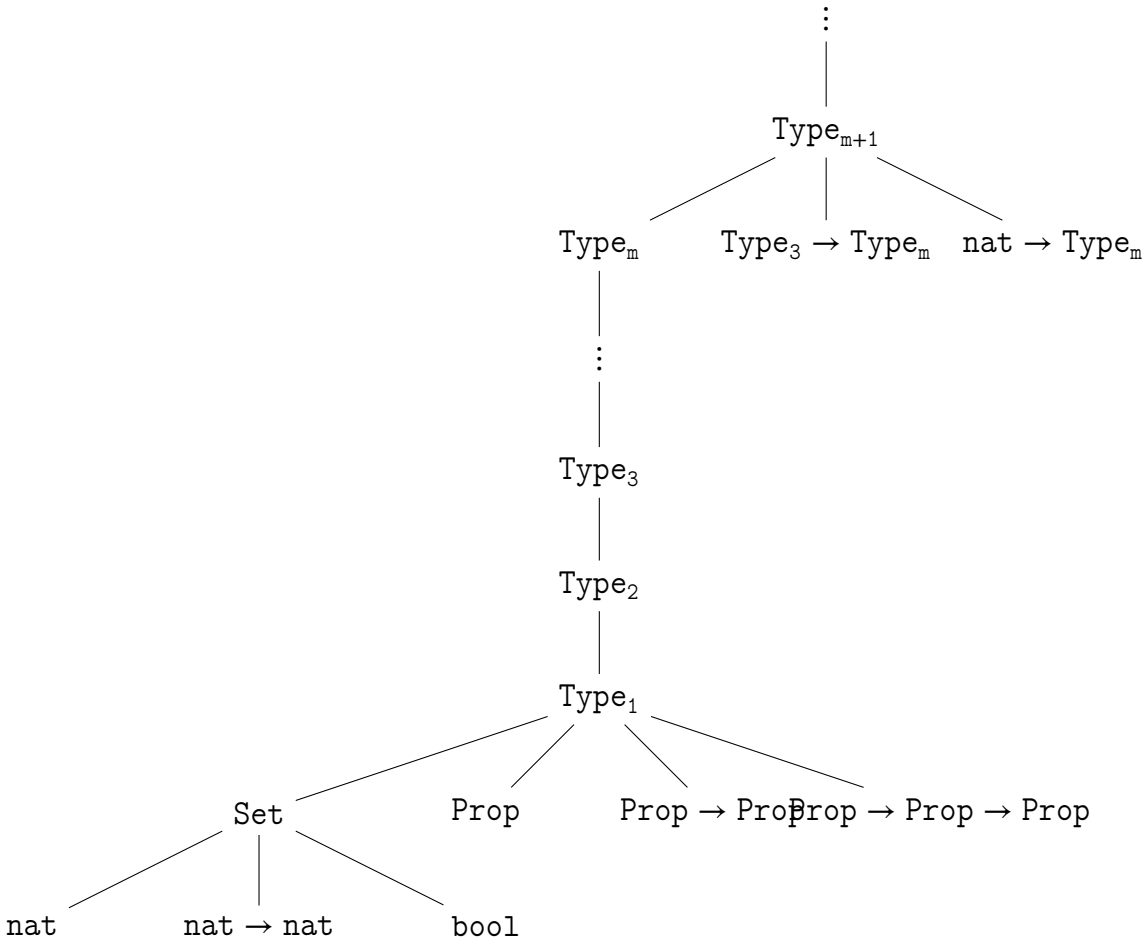
Ekstrakcija OCaml, Haskell, spomenuti da se radi na verificiranoj ekstrakciji

Misao: *kod Fixpoint, spomenuti da je Gallina odlučiv jezik, odnosno da je za svaka Gallinina funkcija odlučiva*

[3] `https://wiki.portal.chalmers.se/agda/`
[4] `https://www.idris-lang.org/`
[5] `https://lean-lang.org/`

2.3. Kumulativna hijerarhija tipova

Ukratko objasniti. Lijepa skica koja prikazuje gdje su `nat`, `nat -> nat`, `nat -> Set`, `Prop -> Prop`, i njima srodni. Razlika između `Set` i `Prop`.



Slika 2.1. Kumulativna hijerarhija tipova

2.4. Propozicije i tipovi, dokazi i programi

Ukratko objasniti što je to Curry–Howard, možda najlakše pomoću BHK interpretacije.

Primjeri dokaznih terma, recimo ručno napisan dokazni term za komutiranje univerzalnih kvantifikacija, pa neki jednostavni induktivni dokaz.

Principi indukcije kao rekurzivne funkcije. **Misao:** *Zavisni produkt se zove produkt jer je univerzalna kvantifikacija u načelu beskonačna konjunkcija.* **Misao:** *Zavisni koprodukt se zove koprodukt jer ??? Ako gledamo obične produkte, obični koprodukt je onda suma.*

Logika	Program	Logički term	Programski term
konjunkcija	produktni tip	`and`	`prod`
disjunkcija	zbrojni tip	`or`	`sum`
implikacija	funkcijski tip	`->`	`->`
univerzalna kvantifikacija	zavisni produkt	`forall x, P x`	`forall x, P x`
egzistencijalna kvantifikacija	zavisni koprodukt	`ex`	`sigT`
istina	jedinični tip	`True`	`unit`
laž	prazni tip	`False`	`Empty_set`
modus ponens	poziv funkcije		
teorem	tip		
dokaz	term		
pretpostavka	varijabla		
dokazivanje	programiranje		
dokazivost	nastanjenost tipa		

Tablica 2.1. Korespondencija logike i programiranja

2.5. Ograničenja u programiranju i dokazivanju

Tu prvenstveno mislim na uvjete pozitivnosti i produktivnosti za induktivne i koinduktivne tipove, te na eliminaciju propozicija kod definiranja nečega u `Type`.

3. Logika prvog reda s induktivnim definicijama

3.1. Sintaksa

Signatura. Term. Formula.

3.2. Semantika

Struktura. Okolina. Evaluacija. Relacija ispunjivosti. Substitution sanity leme.

3.3. Standardni modeli

Produkcije. Skup induktivnih definicija. Operator φ_Φ. Aproksimanti. Standardni model.

3.4. Sistem sekvenata s induktivnim definicijama

LKID. Dopustiva pravila. Primjeri dokaza.

3.5. Adekvatnost

Lokalne adekvatnosti za pravila izvoda. Glavni teorem.

4. Ciklički dokazi

Koinduktivni tip podatka i koinduktivna propozicija. Jedan primjer su `Streamovi` i predikat `Infinite`. Jednostavniji primjer bi možda bio koinduktivni `nat` i koinduktivni `le`.

Kako bi izgledali ciklički dokazi u LKID? Ono što je tamo "repeat funkcija" je u Coqu `cofix`.

5. Zaključak

Literatura

[1] Y. Bertot i P. Castéran, *Interactive theorem proving and program development: Coq'Art: the Calculus of Inductive Constructions*. Springer Science & Business Media, 2013.

[2] B. C. Pierce, A. A. de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, i B. Yorgey, *Logical Foundations*, ser. Software Foundations, B. C. Pierce, Ur. Electronic textbook, 2023., sv. 1, version 6.5, http://softwarefoundations.cis.upenn.edu.

[3] B. C. Pierce, A. A. de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, A. Tolmach, i B. Yorgey, *Programming Language Foundations*, ser. Software Foundations, B. C. Pierce, Ur. Electronic textbook, 2024., sv. 2, version 6.5, http://softwarefoundations.cis.upenn.edu.

[4] A. W. Appel, *Verified Functional Algorithms*, ser. Software Foundations, B. C. Pierce, Ur. Electronic textbook, 2023., sv. 3, version 1.5.4, http://softwarefoundations.cis.upenn.edu.

[5] A. Chlipala, *Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant*. 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.

[7] 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*. Berlin, Heidelberg: Springer-Verlag, 1989., str. 209–228.

[8] E. Giménez, “Codifying guarded definitions with recursive schemes”, u *Types for Proofs and Programs*, P. Dybjer, B. Nordström, i J. Smith, Ur. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995., str. 39–59.

[9] M. Sozeau, S. Boulier, 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, str. 1–28, 2019.

[10] G. Hutton, *Programming in Haskell*, 2. izd. Cambridge University Press, 2016.

Sažetak

Primjene Coq alata za dokazivanje u matematici i računarstvu

Miho Hren

Unesite sažetak na hrvatskom.

Ključne riječi: prva ključna riječ; druga ključna riječ; treća ključna riječ

Abstract

Applications of the Coq Proof Assistant in mathematics and computer science

Miho Hren

Enter the abstract in English.

Keywords: the first keyword; the second keyword; the third keyword