Primjene Coq alata za dokazivanje u matematici i računarstvu

Logika prvog reda s induktivnim definicijama

Miho Hren

Mentori: Vedran Čačić, Marko Doko + Ante Đerek
Fakultet Elektrotehnike i Računarstva

2023./2024.

Što je Coq?

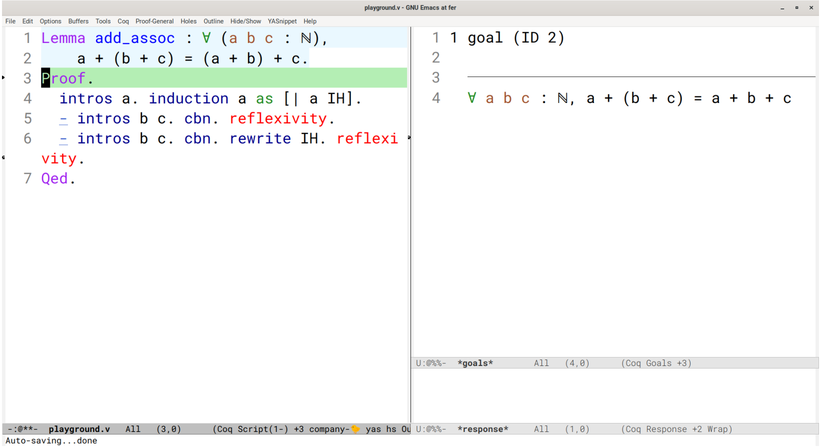
- programski jezik
- **interaktivni dokazivač teorema** — stroj provjerava dokaz
- temeljen na teoriji tipova

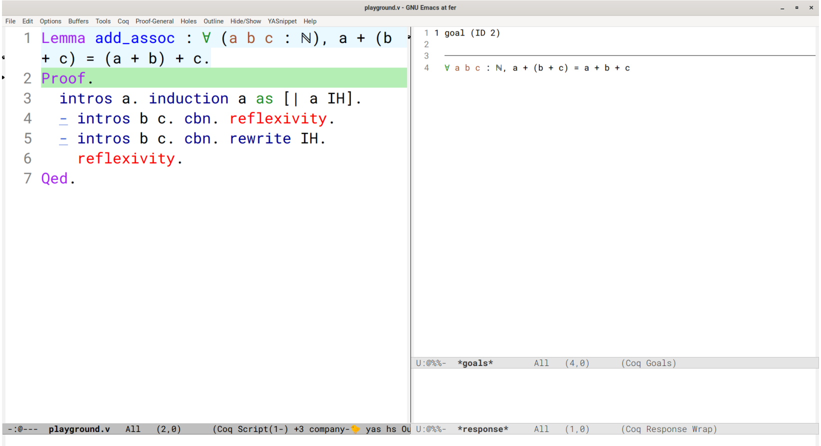
$$\begin{aligned} \textit{propozicija} &= \textit{tip} \\ \textit{dokaz} &= \textit{program} \end{aligned}$$

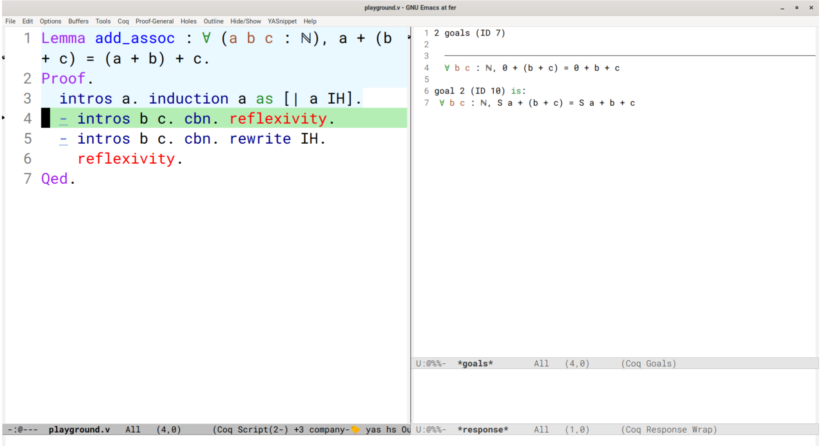
- u matematici: **teorem o četiri boje**, temelji matematike, *obrazovanje*
- u računarstvu: **CompCert**, VeLLVM, CertiKOS, certificirano programiranje

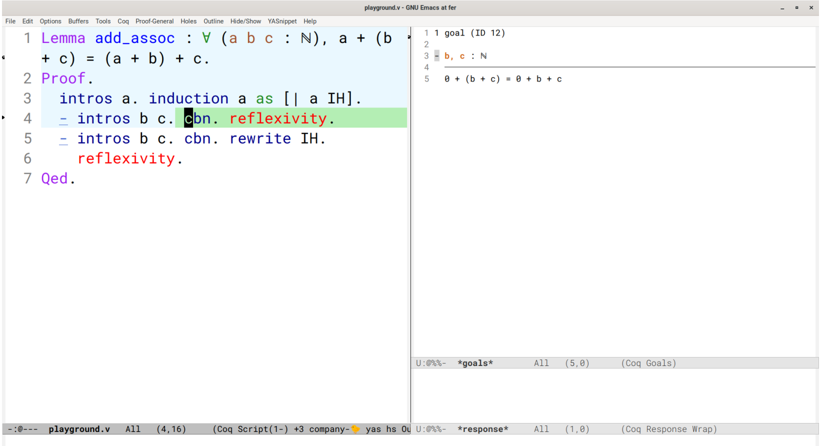
Zašto?

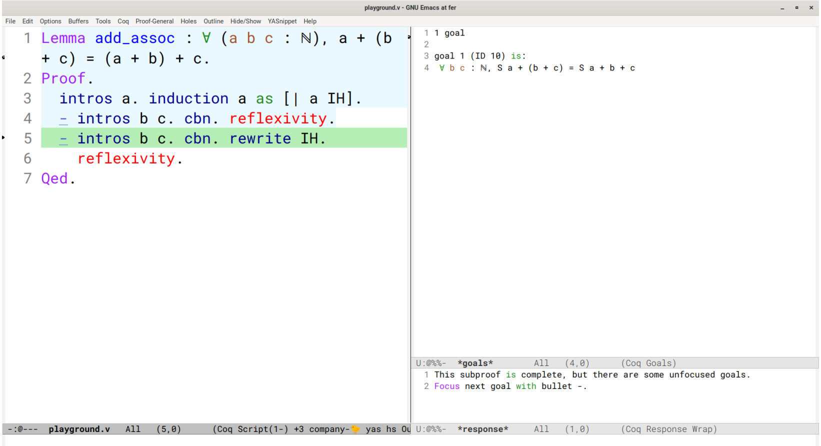
Kritična infrastruktura!

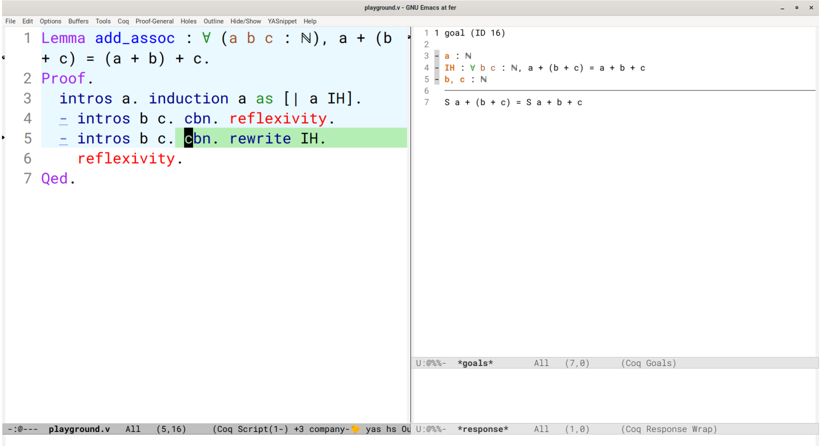

playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
1 Lemma add_assoc : ∀ (a b c : ℕ),
2 a + (b + c) = (a + b) + c.
3 Proof.
4 intros a. induction a as [| a IH].
5 - intros b c. cbn. reflexivity.
6 - intros b c. cbn. rewrite IH. reflexi
vity.
7 Qed.
1 1 goal (ID 2)
2
3
4 ∀ a b c : ℕ, a + (b + c) = a + b + c
U:@%%- *goals* All (4,0) (Coq Goals +3)
-:@**- playground.v All (3,0) (Coq Script(1-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response +2 Wrap)
Auto-saving...done

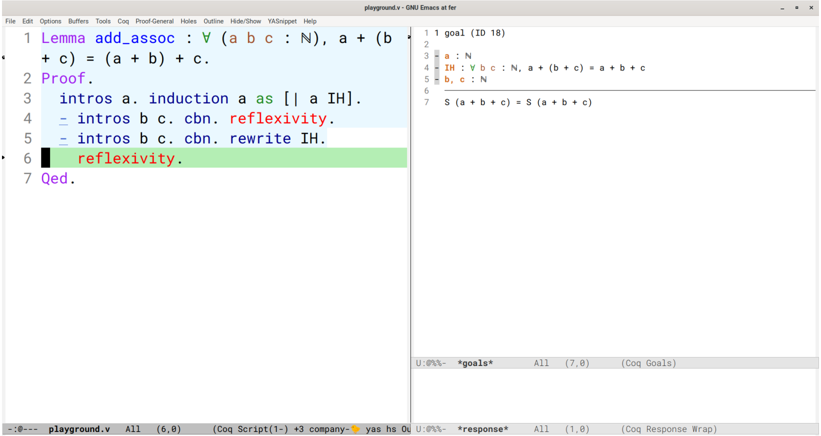

playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
Proof.
intros a. induction a as [| a IH].
- intros b c. cbn. reflexivity.
- intros b c. cbn. rewrite IH.
reflexivity.
Qed.
1 goal (ID 2)
∀ a b c : ℕ, a + (b + c) = a + b + c
U:@%%- *goals* All (4,0) (Coq Goals)
-:@--- playground.v All (2,0) (Coq Script(1-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response Wrap)

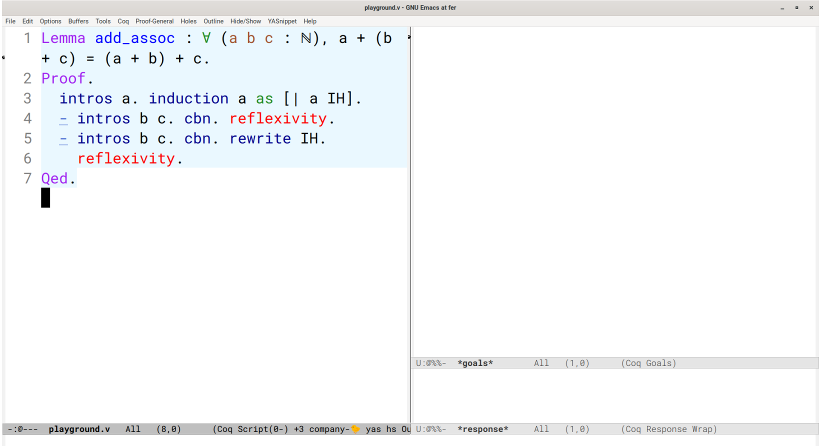

playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
Proof.
intros a. induction a as [| a IH].
- intros b c. cbn. reflexivity.
- intros b c. cbn. rewrite IH.
reflexivity.
Qed.
-:@--- playground.v All (4,0) (Coq Script(2-) +3 company- yas hs Ou
2 goals (ID 7)
∀ b c : ℕ, 0 + (b + c) = 0 + b + c
goal 2 (ID 10) is:
∀ b c : ℕ, S a + (b + c) = S a + b + c
U:@%%- *goals* All (4,0) (Coq Goals)
U:@%%- *response* All (1,0) (Coq Response Wrap)


playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
1 Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
2 Proof.
3 intros a. induction a as [| a IH].
4 - intros b c. cbn. reflexivity.
5 - intros b c. cbn. rewrite IH.
6 reflexivity.
7 Qed.
1 1 goal (ID 12)
2
3 b, c : ℕ
4
5 0 + (b + c) = 0 + b + c
U:@%%- *goals* All (5,0) (Coq Goals)
-:@--- playground.v All (4,16) (Coq Script(1-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response Wrap)

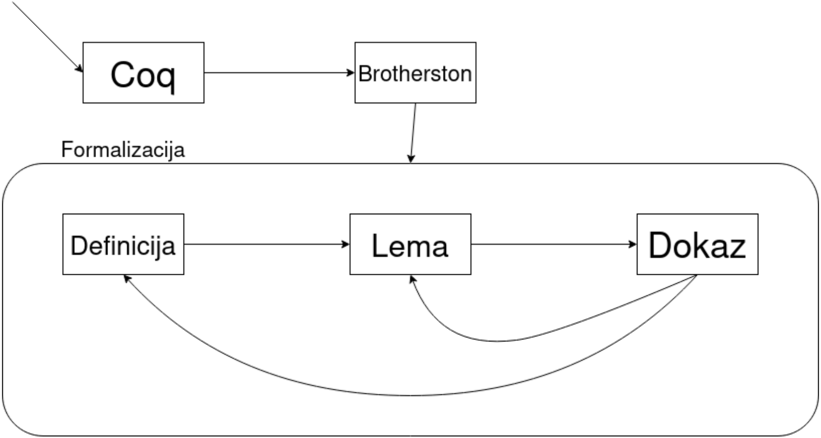

playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
Proof.
intros a. induction a as [| a IH].
- intros b c. cbn. reflexivity.
- intros b c. cbn. rewrite IH.
reflexivity.
Qed.
1 goal
goal 1 (ID 10) is:
∀ b c : ℕ, S a + (b + c) = S a + b + c
U:@%%- *goals* All (4,0) (Coq Goals)
This subproof is complete, but there are some unfocused goals.
Focus next goal with bullet -.
-:@--- playground.v All (5,0) (Coq Script(1-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response Wrap)


playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
1 Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
2 Proof.
3 intros a. induction a as [| a IH].
4 - intros b c. cbn. reflexivity.
5 - intros b c. cbn. rewrite IH.
6 reflexivity.
7 Qed.
-:@--- playground.v All (5,16) (Coq Script(1-) +3 company- yas hs Ou
1 1 goal (ID 16)
2
3 a : ℕ
4 IH : ∀ b c : ℕ, a + (b + c) = a + b + c
5 b, c : ℕ
6
7 S a + (b + c) = S a + b + c
U:@%%- *goals* All (7,0) (Coq Goals)
U:@%%- *response* All (1,0) (Coq Response Wrap)


playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
Proof.
intros a. induction a as [| a IH].
- intros b c. cbn. reflexivity.
- intros b c. cbn. rewrite IH.
reflexivity.
Qed.
1 goal (ID 18)
a : ℕ
IH : ∀ b c : ℕ, a + (b + c) = a + b + c
b, c : ℕ
S (a + b + c) = S (a + b + c)
U:@%%- *goals* All (7,0) (Coq Goals)
-:@--- playground.v All (6,0) (Coq Script(1-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response Wrap)


playground.v - GNU Emacs at fer
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
Lemma add_assoc : ∀ (a b c : ℕ), a + (b + c) = (a + b) + c.
Proof.
intros a. induction a as [| a IH].
- intros b c. cbn. reflexivity.
- intros b c. cbn. rewrite IH.
reflexivity.
Qed.
U:@%%- *goals* All (1,0) (Coq Goals)
-:@--- playground.v All (8,0) (Coq Script(0-) +3 company- yas hs Ou
U:@%%- *response* All (1,0) (Coq Response Wrap)

Proces



Sintaksa: signatura

Definition 2.1.1 (First-order language with inductive predicates). A *(first-order) language with inductive predicates* Σ is a set of symbols including:

- denumerably many constant symbols $c_1, c_2, \ldots$;
- denumerably many function symbols $f_1, f_2, \ldots$, each with associated arity $k > 0$;
- denumerably many *ordinary* predicate symbols $Q_1, Q_2, \ldots$, each with associated arity $k \geq 0$;
- finitely many *inductive* predicate symbols $P_1, \ldots, P_n$, each with associated arity $k \geq 0$.

```
Structure signature := {
    FuncS : Set;
    fun_ar : FuncS -> nat;
    PredS : Set;
    pred_ar : PredS -> nat;
    IndPredS : Set;
    indpred_ar : IndPredS -> nat;
  }.
```

Primjer: Peanova signatura

$$\sigma_{PA} = \{\{o^0, s^1, +^2, \cdot^2\}, \{=^2\}, \{Nat^1, Even^1, Odd^1\}\}$$

Sintaksa: termi

Definition 2.1.2 (Terms)**.** The set of *terms* of a first-order language Σ, $Terms(\Sigma)$, is the smallest set of expressions of Σ closed under the following rules:

1. any variable $x \in \mathcal{V}$ is a term;
2. any constant symbol $c \in \Sigma$ is a term;
3. if $f \in \Sigma$ is a function symbol of arity k, and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term.

```
Inductive term  : Set :=
| var_term : var -> term
| TFunc : forall (f : FuncS Σ),
    vec term (fun_ar f) -> term.
```

Primjeri terma

$$s(o), s(s(x)), f(x, v, w, g(z, q))$$

```
Example PA_one (* s(o) *): term Σ__PA :=
  TFunc PA_succ [TFunc PA_zero []].
```

Sintaksa: formula

Definition 2.1.6 (Formulas). Given a first-order language Σ, the set of Σ-formulas of $\mathrm{FOL}_{\mathrm{ID}}$ is the smallest set of expressions closed under the following rules:

1. if $t_1, \ldots, t_k$ are terms of Σ, and Q is a predicate symbol in Σ of arity k, then $Q(t_1, \ldots, t_k)$ is a formula;
2. if t and u are terms of Σ then $t = u$ is a formula;
3. if F is a formula then so is $\neg F$;
4. if F_1 and F_2 are formulas then so are $F_1 \wedge F_2$, $F_1 \vee F_2$ and $F_1 \rightarrow F_2$;
5. if F is a formula and $x \in \mathcal{V}$ is a variable, then $\exists x F$ and $\forall x F$ are formulas.

```
Inductive formula : Set :=
| FPred (P : PredS Σ)
    : vec (term Σ) (pred_ar P) -> formula
| FIndPred (P : IndPredS Σ)
    : vec (term Σ) (indpred_ar P) -> formula
| FNeg : formula -> formula
| FImp : formula -> formula -> formula
| FAll : formula -> formula.
```

Sintaksa: formula

Primjer formule

$$\forall x, \mathit{Nat}(x) \rightarrow \mathit{Even}(x) \vee \mathit{Odd}(x)$$

```
Definition every_nat_is_even_or_odd
  : formula Σ__PA :=
  let x := var_term 0 in
  FAll
    (FImp
       (FIndPred PA_Nat [x])
       (FOr
          (FIndPred PA_Even [x])
          (FIndPred PA_Odd  [x]))).
```

Sintaksa: produkcije

Definition 2.2.1 (Inductive definition set)**.** An *inductive definition set* Φ for a language Σ is a finite set of *productions*, which are rules of the form:

$$\frac{Q_1\mathbf{u_1}(\mathbf{x}) \dots Q_h\mathbf{u_h}(\mathbf{x}) \quad P_{j_1}\mathbf{t_1}(\mathbf{x}) \dots P_{j_m}\mathbf{t_m}(\mathbf{x})}{P_i\mathbf{t}(\mathbf{x})} \qquad j_1,\dots,j_m,i \in \{1,\dots,n\}$$

i.e., $Q_1,\dots,Q_h$ are ordinary predicates and $P_{j_1},\dots,P_{j_m},P_i$ are inductive predicates of Σ.

$$\frac{Q_1 \mathrm{u}_1 \dots Q_n \mathrm{u}_n \qquad P_1 \mathrm{v}_1 \dots P_m \mathrm{v}_m}{P \mathrm{t}}$$

```
Record production :=
  mkProd {
      preds
        : list {P: PredS Σ & vec (term Σ) (pred_ar P)};
      indpreds
        : list {P: IndPredS Σ & vec (term Σ) (indpred_ar P)};
      indcons : IndPredS Σ;
      indargs : vec (term Σ) (indpred_ar indcons);
    }.
```

Biti prirodan broj.

$$\frac{}{Nat(o)} \qquad \frac{Nat(x)}{Nat(s(x))}$$

Biti paran, odnosno neparan broj.

$$\frac{}{Even(o)} \qquad \frac{Odd(x)}{Even(s(x))} \qquad \frac{Even(x)}{Odd(s(x))}$$

Semantika: struktura, okolina

```
Structure structure := {
    domain :> Set;
    interpF (f : FuncS Σ)
        : vec domain (fun_ar f) -> domain;
    interpP (P : PredS Σ)
        : vec domain (pred_ar P) -> Prop;
    interpIP (P : IndPredS Σ)
        : vec domain (indpred_ar P) -> Prop;
  }.

Definition env := var -> M.
```

Primjer: standardna Peanova struktura

$$M_{PA} = (\mathbb{N}, 0, S, +, \cdot, =, \mathbb{N}, \mathbb{E}, \mathbb{O})$$

Semantika: istinitost formule

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

Primjer

$$(M_{PA}, \rho) \vDash \forall x, Nat(x) \to Even(x) \vee Odd(x)$$

Semantika: aproksimacije skupa produkcija

```
Definition InterpInd :=
    forall P : IndPredS Σ, vec M (indpred_ar P) -> Prop.

Fixpoint φ_Φ_n (α : nat) : InterpInd :=
  match α with
  | 0 => fun _ _ => False
  | S α => φ_Φ (φ_Φ_n α)
  end.

(* aproksimacija beskonačne razine *)
Definition φ_Φ_ω : InterpInd :=
    fun P v => exists α, φ_Φ_n α P v.

Lemma φ_Φ_ω_least_prefixed : least prefixed φ_Φ_ω.
```

Semantika: standardni modeli

```
Definition standard_model
  (Φ: IndDefSet Σ)
  (M : structure Σ)
  : Prop :=
    forall (P : IndPredS Σ) ts,
      interpIP P ts <-> φ_Φ_ω Φ M P ts.
```

Sekvente

$$\Gamma \vdash \Delta$$

```
Definition Sat_sequent (s : sequent) : Prop :=
  let '(Γ ⊢ Δ) := s in
  forall (M : structure Σ),
      standard_model Φ M -> forall (ρ : env M),
        (forall φ, In φ Γ -> ρ ⊨ φ) ->
        exists ψ, In ψ Δ /\ ρ ⊨ ψ.
```

$$\Gamma \mid\models \Delta$$

Sistem sekvenata: „obična” pravila izvoda

$$\frac{\Gamma \cap \Delta \neq \varnothing}{\Gamma \vdash \Delta}\,(Ax) \qquad \frac{\Gamma' \vdash \Delta' \qquad \Gamma' \subseteq \Gamma \qquad \Delta' \subseteq \Delta}{\Gamma \subseteq \Delta}\,(Wk)$$

$$\frac{\Gamma \vdash \varphi, \Delta \qquad \varphi, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,(Cut) \qquad \frac{\Gamma \vdash \Delta}{\Gamma[\sigma] \vdash \Delta[\sigma]}\,(Subst)$$

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

$$\frac{\Gamma \vdash \varphi, \Delta \qquad \psi, \Gamma \vdash \Delta}{\varphi \to \psi, \Gamma \vdash \Delta}\,(ImpL) \qquad \frac{\varphi, \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}\,(ImpR)$$

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

```
Inductive LKID : sequent → ℙ :=
(* Structural rules. *)
| Ax : ∀ Γ Δ φ, In φ Γ → In φ Δ → LKID (Γ ⊢ Δ)
| Wk : ∀ Γ' Δ' Γ Δ,
    Γ' ⊆ Γ →
    Δ' ⊆ Δ →
    LKID (Γ' ⊢ Δ') →
    LKID (Γ ⊢ Δ)
| Cut : ∀ Γ Δ φ,
    LKID (Γ ⊢ φ :: Δ) →
    LKID (φ :: Γ ⊢ Δ) →
    LKID (Γ ⊢ Δ)
| Subst : ∀ Γ Δ,
    LKID (Γ ⊢ Δ) →
    ∀ σ, LKID (map (subst_formula σ) Γ ⊢ map (subst_formula σ) Δ)
(* Propositional rules. *)
| NegL : ∀ Γ Δ φ, LKID (Γ ⊢ φ :: Δ) → LKID (FNeg φ :: Γ ⊢ Δ)
| NegR : ∀ Γ Δ φ, LKID (φ :: Γ ⊢ Δ) → LKID (Γ ⊢ FNeg φ :: Δ)
| ImpL : ∀ Γ Δ φ ψ,
    LKID (Γ ⊢ φ :: Δ) → LKID (ψ :: Γ ⊢ Δ) →
    LKID (FImp φ ψ :: Γ ⊢ Δ)
| ImpR : ∀ Γ Δ φ ψ,
    LKID (φ :: Γ ⊢ ψ :: Δ) → LKID (Γ ⊢ (FImp φ ψ) :: Δ)
(* Quantifier rules. *)
| AllL : ∀ Γ Δ φ t,
    LKID (subst_formula (t .: ids) φ :: Γ ⊢ Δ) →
    LKID (FAll φ :: Γ ⊢ Δ)
| AllR : ∀ Γ Δ φ,
    LKID (shift_formulas Γ ⊢ φ :: shift_formulas Δ) →
    LKID (Γ ⊢ (FAll φ) :: Δ)
```

Sistem sekvenata: produkcijska pravila

Produkcija

$$\frac{Q_1\mathsf{u}_1 \dots Q_n\mathsf{u}_n \qquad P_1\mathsf{v}_1 \dots P_m\mathsf{v}_m}{P\mathsf{t}}$$

Pravilo

$$\frac{\Gamma \vdash Q_1\mathsf{u}_1[\sigma], \Delta \quad \dots \quad \Gamma \vdash Q_n\mathsf{u}_n[\sigma], \Delta \qquad \Gamma \vdash P_1\mathsf{v}_1[\sigma], \Delta \quad \dots \quad \Gamma \vdash P_m\mathsf{v}_m[\sigma], \Delta}{\Gamma \vdash P\mathsf{t}[\sigma], \Delta}$$

Primjer

$$\frac{Odd(x)}{Even(s(x))} \qquad \frac{\Gamma \vdash Odd(x), \Delta}{\Gamma \vdash Even(s(x)), \Delta}$$

```
| Prod : ∀ Γ Δ pr σ,
    Φ pr →
    (∀ Q us,
        In (Q; us) (preds pr) →
        LKID (Γ ⊢ (FPred Q (V.map (subst_term σ) us) :: Δ))) →
    (∀ P ts,
        In (P; ts) (indpreds pr) →
        LKID (Γ ⊢ (FIndPred P (V.map (subst_term σ) ts) :: Δ))) →
    LKID ( Γ ⊢ FIndPred
                  (indcons pr)
                  (V.map (subst_term σ) (indargs pr))
                  :: Δ).
```

```
| Ind : ∀ Γ Δ
        (Pj : IndPredS Σ) (u : vec _ (indpred_ar Pj))
        (z_i : ∀ P, vec var (indpred_ar P))
        (z_i_nodup : ∀ P, VecNoDup (z_i P))
        (G_i : IndPredS Σ → formula Σ)
        (HG2 : ∀ Pi, ~mutually_dependent Pi Pj →
                    G_i Pi = FIndPred
                               Pi
                               (V.map var_term (z_i Pi))),
    let maxΓ := max_fold (map some_var_not_in_formula Γ) in
    let maxΔ := max_fold (map some_var_not_in_formula Δ) in
    let maxP := some_var_not_in_formula (FIndPred Pj u) in
    let shift_factor := max maxP (max maxΓ maxΔ) in
    let Fj := subst_formula
                (finite_subst (z_i Pj) u)
                (G_i Pj) in
    let minor_premises :=
      (∀ pr (HΦ : Φ pr) (Hdep : mutually_dependent (indcons pr) Pj),
        let Qs := shift_formulas_by
                    shift_factor
                    (FPreds_from_preds (preds pr)) in
        let Gs := map (λ '(P; args) ⇒
                         let shifted_args :=
                           V.map
                             (shift_term_by shift_factor)
                             args in
                         let σ :=
                           finite_subst
                             (z_i P)
                             (shifted_args) in
                         let G := G_i P in
                         subst_formula σ G)
                    (indpreds pr) in
        let Pi := indcons pr in
        let ty := V.map
                    (shift_term_by shift_factor)
                    (indargs pr) in
        let Fi := subst_formula
                    (finite_subst (z_i Pi) ty)
                    (G_i Pi) in
        LKID (Qs ++ Gs ++ Γ ⊢ Fi :: Δ))
    in
    minor_premises →
    LKID (Fj :: Γ ⊢ Δ) →
    LKID (FIndPred Pj u :: Γ ⊢ Δ)
```

Sistem sekvenata: pravila indukcije

Primjer

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

Primjer primjene pravila *NatInd*

$$\frac{\dfrac{\vdots}{\vdash Eo \lor Oo, Ex \lor Ox} \qquad \dfrac{\vdots}{Ey \lor Oy \vdash Esy \lor Osy, Ex \lor Ox} \qquad \dfrac{\vdots}{Ex \lor Ox \vdash Ex \lor Ox}}{Nx \vdash Ex \lor Ox}$$

Adekvatnost

Teorem

Ako je sekventa **dokaziva** u sustavu *LKID*,
onda je **istinita** na standardnim modelima.

Dokaz

Indukcijom po strukturi dokaza sekvente $\Gamma \vdash \Delta$.

- u disertaciji: šest stranica teksta
- u našoj formalizaciji: oko 450 linija

Problemi kod formalizacije dokaza:

- što je pisac htio reći?
- implicitne pretpostavke
- implicitno domensko znanje

Zaključak

Formalizacija

- sintaksa i semantika logike prvog reda s induktivnim definicijama
- dokazni sustav *LKID*
- oko **140 lema i dokaza**

Tekst

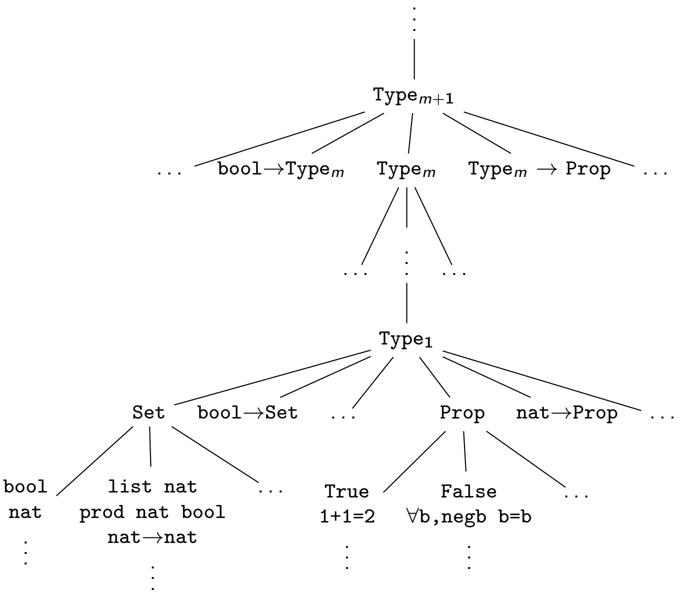
- osnovno o Coqu
- **opis formalizacije**
- ilustracija cikličkih dokaza

Što dalje?

- potpunost
- dokazni sustav $CLKID^{\omega}$
- formalno verificirani dokazivač teorema

Hvala!

Hijerarhija tipova u Coqu



Curry–Howardova korespondencija

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

Ograničenja Coqovog tipskog sustava

- uvjet pozitivnosti
- uvjet strukturalne rekurzije
- uvjet produktivnosti
- ograničenje na eliminaciju propozicije

Peanova signatura

```
Inductive Func__PA :=
| PA_zero
| PA_succ
| PA_add
| PA_mult.
Definition fun_ar__PA (s : Func__PA) : nat :=
  match s with
   | PA_zero => 0
   | PA_succ => 1
   | PA_add  => 2
   | PA_mult => 2
  end.

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

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

Peanova signatura

```
Definition Σ__PA : signature
  := {|
    FuncS := Func__PA;
    fun_ar := fun_ar__PA;
    PredS := Pred__PA;
    pred_ar := pred_ar__PA;
    IndPredS := IndPred__PA;
    indpred_ar := indpred_ar__PA;
  |}.
```

Primjer skupa produkcija

```
Definition PA_prod_N_zero : production Σ__PA.
  refine (mkProd nil nil PA_Nat _).
  refine [TFunc PA_zero []].
Defined.

Definition PA_prod_N_succ : production Σ__PA.
  refine (mkProd nil _ PA_Nat _).
  - refine (cons _ nil). exists PA_Nat; refine [var_term 0].
  - refine [TFunc PA_succ [var_term 0]].
Defined.

Definition PA_prod_E_zero : production Σ__PA.
  refine (mkProd nil nil PA_Even _).
  refine [ TFunc PA_zero []].
Defined.

Definition PA_prod_E_succ : production Σ__PA.
  refine (mkProd nil _ PA_Even _).
  - refine (cons _ nil). exists PA_Odd; refine [var_term 0].
  - refine [TFunc PA_succ [var_term 0]].
Defined.

Definition PA_prod_O_succ : production Σ__PA.
  refine (mkProd nil _ PA_Odd _).
  - refine (cons _ nil). exists PA_Even; refine [var_term 0].
  - refine [TFunc PA_succ [var_term 0]].
Defined.
```

Formalizacija standardne Peanove strukture

```
Inductive EVEN : nat -> Prop :=
| E0 : EVEN 0
| ES : forall n, ODD n -> EVEN (S n)
with ODD : nat -> Prop :=
| OS : forall n, EVEN n -> ODD (S n).

Definition M__PA : structure Σ__PA.
  refine (Build_structure nat _ _ _).
  - intros f; destruct f.
    + intros. exact 0.
    + intros n. exact (S (V.hd n)).
    + intros xy. exact (V.hd xy + V.hd (V.tl xy)).
    + intros xy. exact (V.hd xy * V.hd (V.tl xy)).
  - intros P args; destruct P.
    exact (V.hd args = V.hd (V.tl args)).
  - intros P args; destruct P.
    + exact (NAT (V.hd args)).
    + exact (EVEN (V.hd args)).
    + exact (ODD (V.hd args)).
Defined.
```

Još jedna korisna lema

```
Lemma strong_form_subst_sanity2 :
  forall (Σ : signature) (φ : formula Σ) (σ : var -> term Σ)
    (M : structure Σ) (ρ : env M),
    ρ ⊨ (subst_formula σ φ) <-> (σ >> eval ρ) ⊨ φ.
Proof.
  intros Σ φ; induction φ; intros σ M ρ; cbn; intuition.
  - erewrite <- vec_comp.
    + eauto.
    + intros u; asimpl; now rewrite eval_comp.
  - erewrite vec_comp.
    + eapply H.
    + intros u; asimpl; now rewrite eval_comp.
  - erewrite <- vec_comp.
    + eapply H.
    + intros u; asimpl; now rewrite eval_comp.
  - erewrite vec_comp.
    + eapply H.
    + intros u; asimpl; now rewrite eval_comp.
  - now apply H, IHφ.
  - now apply H, IHφ.
  - apply IHφ2; apply H; apply IHφ1; auto.
  - apply IHφ2; apply H; apply IHφ1; auto.
  - asimpl in H. specialize H with d.
    apply IHφ in H. asimpl in H. simpl in H.
    rewrite eval_shift in H. apply H.
  - rewrite IHφ. asimpl. simpl.
    rewrite eval_shift.
    apply H.
Qed.
```

Primjer aproksimiranja skupa produkcija

Aproksimiramo skup produkcija Φ_{PA} na strukturi M_{PA}.

$$\frac{}{Nat(o)} \qquad \frac{Nat(x)}{Nat(s(x))} \qquad \frac{}{Even(o)} \qquad \frac{Odd(x)}{Even(s(x))} \qquad \frac{Even(x)}{Odd(s(x))}$$

$$\begin{aligned}
(N, E, O)^0 &= (\varnothing, \varnothing, \varnothing) \\
(N, E, O)^1 &= (\{0\}, \{0\}, \varnothing) \\
(N, E, O)^2 &= (\{0, 1\}, \{0\}, \{1\}) \\
(N, E, O)^3 &= (\{0, 1, 2\}, \{0, 2\}, \{1\}) \\
(N, E, O)^4 &= (\{0, 1, 2, 3\}, \{0, 2\}, \{1, 3\}) \\
(N, E, O)^5 &= (\{0, 1, 2, 3, 4\}, \{0, 2, 4\}, \{1, 3\}) \\
(N, E, O)^6 &= \ldots \\
&\vdots \\
(N, E, O)^\omega &= (\mathbb{N}, \mathbb{E}, \mathbb{O})
\end{aligned}$$

Aproksimacije, formalno

```
Definition φ_pr
  (pr : production Σ)
  (interp : InterpInd)
  (ds : vec M (indpred_ar (indcons pr)))
  : Prop :=
      exists (ρ : env M),
      (forall Q us, List.In (Q; us) (preds pr) ->
               interpP Q (V.map (eval ρ) us)) /\
        (forall P ts, List.In (P; ts) (indpreds pr) ->
                 interp P (V.map (eval ρ) ts)) /\
        ds = V.map (eval ρ) (indargs pr).
```

Aproksimacije, formalno

```
Definition φ_P
  (P : IndPredS Σ)
  (interp : InterpInd)
  : vec M (indpred_ar P) -> Prop.
  refine (fun ds => _).
  refine (@ex (production Σ) (fun pr => _)).
  refine (@ex (P = indcons pr /\ Φ pr) (fun '(conj Heq HΦ) => _)).
  rewrite Heq in ds.
  exact (φ_pr pr interp ds).
Defined.

Definition φ_Φ (interp : InterpInd) : InterpInd :=
  fun P => φ_P P interp.
```

```
Lemma φ_Φ_ω_least_prefixed : least prefixed φ_Φ_ω.
Proof.
  split.
  - intros P v H.
    unfold φ_Φ, φ_P, φ_pr in H;
      destruct H as (pr & [Heq Hpr] & (ρ & Hpreds & Hindpreds & Heval)).
    unfold eq_rect in Heval; subst P; subst v.
    enough (Hsup : exists α, forall P ts,
      In (P; ts) (indpreds pr) -> φ_Φ_n α P (V.map (eval ρ) ts)).
    + destruct Hsup as [κ Hsup].
      exists (S κ), pr, (conj eq_refl Hpr), ρ; split; auto.
    + induction (indpreds pr) as [| [P' v'] indpreds' IH].
      * exists 0; inversion 1.
      * pose proof (Hindpreds P' v').
        assert (Hin: In (P'; v') ((P'; v') :: indpreds')) by now left.
        apply H in Hin as [α Hα].
        assert (IH_help : forall P ts,
          In (P; ts) indpreds' -> φ_Φ_ω P (V.map (eval ρ) ts)).
        { intros P ts Hin. apply Hindpreds. now right. }
        apply IH in IH_help as [β Hβ].
        exists (S (max α β)).
        intros P ts Hin; inversion Hin.
        -- apply φ_Φ_n_monotone with α; auto with arith.
           inversion H0; subst P.
           apply inj_pair2 in H0; now subst.
        -- apply φ_Φ_n_monotone with β; auto with arith.
  - intros interp Hprefixed P v Hω.
    destruct Hω as [α Hφ].
    enough (H: forall β, φ_Φ (φ_Φ_n β) P v -> φ_Φ interp P v).
    + now apply Hprefixed, (H α), φ_Φ_n_succ.
    + intros β; apply φ_Φ_monotone. induction β as [| β IH].
      * inversion 1.
      * cbn; unfold prefixed in Hprefixed.
        apply φ_Φ_monotone in IH. red; auto.
Qed.
```

Primjer dokaza u sustavu *LKID*

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \vdash \varphi}\,(Ax)}{\vdash \neg\varphi, \varphi}\,(NegR)}{\vdash \varphi, \neg\varphi}\,(Perm)}{\vdash \varphi \lor \neg\varphi}\,(OrR)
$$

Primjer dokaza u sustavu *LKID*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \vdash \varphi, \Delta}\,(Ax)}{\neg\varphi, \varphi \vdash \Delta}\,(NegL)}{\varphi, \neg\varphi \vdash \Delta}\,(Perm)}{\varphi \wedge \neg\varphi \vdash \Delta}\,(AndL)$$

Primjer dokaza u sustavu *LKID*

$$
\dfrac{\dfrac{\dfrac{}{Ex \vdash Ex, Essx}\,(Ax)}{Ex \vdash Osx, Essx}\,(Prod) \qquad \dfrac{\dfrac{}{Ex, Osx \vdash Osx}\,(Ax)}{Ex, Osx \vdash Essx}\,(Prod)}{\dfrac{\dfrac{Ex \vdash Essx}{\vdash Ex \to Essx}\,(ImpR)}{\vdash \forall x, Ex \to Essx}\,(AllR)}\,(Cut)
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash Eo, Oo}\,(Prod) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Nx \vdash Ex, Ox\ (\dagger)}{Ny \vdash Ey, Oy}\,(Subst)}{Ny \vdash Oy, Ey}\,(Perm)}{Ny \vdash Oy, Osy}\,(Prod)}{Ny \vdash Esy, Osy}\,(Prod)}{x = sy, Ny \vdash Ex, Ox}\,(EqL)}{}}{Nx \vdash Ex, Ox\ (\dagger)}\,(Case\ N)}{Nx \vdash Ex \vee Ox}\,(OrR)}{\vdash Nx \rightarrow Ex \vee Ox}\,(ImpR)}{\vdash \forall x, Nx \rightarrow Ex \vee Ox}\,(AllR)
$$