

OBLICZENIA I WNIOSKOWANIE W SYSTEMIE COQ

Małgorzata Biernacka

Instytut Informatyki UWr

Wykład 2
05.03.2013

PEWNE POJĘCIA PODSTAWOWE

- ▶ dowód formalny
- ▶ logika konstruktywna
- ▶ izomorfizm Curry'ego-Howarda, czyli odpowiedniość między systemem wnioskowania a systemem typów (typy-formuły, termy-dowody, obliczenie-normalizacja)
- ▶ dowodzenie – znajdowanie (jakiegokolwiek) termu-dowodu danego typu (zwykle za pomocą taktyk, ale niekoniecznie)
- ▶ programowanie – konstruowanie (konkretnego) termu-programu (zwykle “wprost”, ale niekoniecznie)

SYSTEM COQ

- ▶ implementuje (i rozszerza) rachunek konstrukcji z definicjami indukcyjnymi: Calculus of Constructions, Th. Coquand 1984, Calculus of Inductive Constructions, Ch. Paulin 1991 (w skrócie CIC lub pCIC)
- ▶ implementacja w języku OCaml
- ▶ narzędzia:
 - coqc - kompilator ($.v \rightarrow .vo$)
 - coqtop - interpreter
 - coqide - interfejs graficzny
 - coqchk - narzędzie do weryfikacji dowodów
 - coqdoc, coqdep, coq_makefile etc.
- ▶ Coq jest intensywnie rozbudowywany i rozszerzany

BEZPIECZEŃSTWO I POPRAWNOŚĆ SYSTEMU

- ▶ poprawność implementacji
 - jądro Coq – typechecker dla pCIC zaimplementowany niezależnie od reszty systemu, w stylu czysto funkcyjnym
 - Coq spełnia kryterium de Bruijna: każdy dowód jest weryfikowalny w minimalnym podsystemie – jądrze Coq
 - poprawność względna – w pewnym momencie zakładamy poprawność: implementacji jądra Coq, kompilatora OCaml i poprawność działania komputera
- ▶ niesprzeczność systemu logicznego
 - pCIC jest niesprzeczny
 - aksjomaty użytkownika mogą powodować, że system staje się sprzeczny
 - bezpieczne aksjomaty (EM, PI, AC, FE)

JĘZYK GALLINA

- ▶ język specyfikacji Coq
- ▶ obejmuje termy rachunku konstrukcji pCIC
- ▶ wszystkie termy są typowane
- ▶ “termy” i “typy” należą do tej samej kategorii syntaktycznej termów
- ▶ w języku Gallina zapisujemy programy, specyfikacje, dowody, formuły
- ▶ system typów $\leftrightarrow$ system wnioskowania (izomorfizm C-H)

SORTY: SET, PROP I TYPE

- ▶ sorty są termami Galliny
- ▶ każdy term ma typ
- ▶ każdy typ należy do pewnego sortu
- ▶ **Set** jest sortem “obliczeniowym”, uniwersum typów danych/specyfikacji programów (obiekty sortu Set są typami programów)
- ▶ **Prop** jest sortem “logicznym”, uniwersum formuł logicznych (obiekty sortu Prop są typami dowodów)
- ▶ specyfikacje i formuły logiczne są termami
- ▶ **Type** jest sortem Set i Prop (naprawdę hierarchia $Type_i$)

PODSTAWOWE KONSTRUKCJE JĘZYKA GALLINA

identyfikatory zmiennych i stałych	x
sorty	Set, Prop, Type
produkt zależny	<code>forall $x : t, u$</code>
λ – abstrakcja	<code>fun $x : t \Rightarrow u$</code>
aplikacja	<code>$t\ u$</code>
konstrukcja let	<code>let $x := t$ in u</code>
definicje rekurencyjne	<code>fix..., cofix...</code>
dopasowanie wzorca	<code>match...with...end</code>

PODSTAWOWE KONSTRUKCJE JĘZYKA GALLINA, C.D.

- ▶ konstrukcje $\text{forall } x : t, u$, $\text{fun } x : t \Rightarrow u$ wiążą zmienną x w termie u
- ▶ pojęcie zmiennej związanej, zmiennej wolnej, α -równoważności i podstawienia jak w rachunku lambda
- ▶ $\text{forall } x : t, u$ reprezentuje kwantyfikator ogólny albo produkt zależny
- ▶ jeśli zmienna x nie występuje w termie u , produkt $\text{forall } x : t, u$ zapisujemy jako $t \rightarrow u$
- ▶ $t \rightarrow u$ oznacza albo typ funkcyjny albo logiczną implikację

REGUŁY TYPOWANIA DLA SORTÓW

$$\frac{WF(E ; \Gamma)}{E ; \Gamma \vdash \text{Set} : \text{Type}_i}$$

$$\frac{WF(E ; \Gamma)}{E ; \Gamma \vdash \text{Prop} : \text{Type}_i}$$

$$\frac{WF(E ; \Gamma) \quad i < j}{E ; \Gamma \vdash \text{Type}_i : \text{Type}_j}$$

REGUŁY TYPOWANIA DLA IDENTYFIKATORÓW

$$\frac{WF(E ; \Gamma) \quad x : T \in \Gamma \text{ lub } x := t : T \in \Gamma}{E ; \Gamma \vdash x : T}$$

$$\frac{WF(E ; \Gamma) \quad c : T \in E \text{ lub } c := t : T \in E}{E ; \Gamma \vdash c : T}$$

REGUŁY TYPOWANIA DLA PRODUKTU ZALEŻNEGO

$$\frac{WF(E ; \Gamma) \vdash T : s \quad s \in \{\text{Set}, \text{Prop}, \text{Type}\} \quad E ; \Gamma, x : T \vdash U : \text{Prop}}{E ; \Gamma \vdash \text{forall } x : T, U : \text{Prop}}$$

$$\frac{WF(E ; \Gamma) \vdash T : s \quad s \in \{\text{Set}, \text{Prop}\} \quad E ; \Gamma, x : T \vdash U : \text{Set}}{E ; \Gamma \vdash \text{forall } x : T, U : \text{Set}}$$

$$\frac{WF(E ; \Gamma) \vdash T : \text{Type}_i \quad i \leq k \quad E ; \Gamma, x : T \vdash U : \text{Type}_j \quad j \leq k}{E ; \Gamma \vdash \text{forall } x : T, U : \text{Type}_k}$$

- ▶ Prop jest niepredykatywny
- ▶ Set jest predykatywny

REGUŁY TYPOWANIA DLA ABSTRAKCJI, APLIKACJI I DEFINICJI LOKALNEJ

$$\frac{E; \Gamma \vdash \text{forall } x: T, U: s \quad s \in S \quad E; \Gamma, x: T \vdash t: U}{E; \Gamma \vdash \text{fun } x: T \Rightarrow t: \text{forall } x: T, U}$$

$$\frac{E; \Gamma \vdash t: \text{forall } x: U, T \quad E; \Gamma \vdash u: U}{E; \Gamma \vdash t u: T[x \mapsto u]}$$

$$\frac{E; \Gamma \vdash t: T \quad E; \Gamma, x := t: T \vdash u: U}{E; \Gamma \vdash \text{let } x := t \text{ in } u: U[x \mapsto T]}$$

JĘZYK KOMEND VERNACULAR

- ▶ deklaracje lokalne i globalne
- ▶ definicje lokalne i globalne
- ▶ definicje indukcyjne
- ▶ twierdzenia, lematy
- ▶ dowody
- ▶ polecenia pomocnicze

DEKLARACJE I DEFINICJE GLOBALNE (Set)

- ▶ globalna deklaracja

```
Parameter x : bool.
```

```
> x is assumed
```

- ▶ globalna definicja

```
Definition square := fun x => x*x.
```

```
> square is defined
```

```
Definition cube (x:nat) : nat := x*x*x.
```

```
> cube is defined
```

DEKLARACJE I DEFINICJE LOKALNE (Set)

- ▶ używane wewnątrz sekcji (mechanizm bloków)
- ▶ lokalna deklaracja

```
Section Trojmian.
```

```
Variables a b c d: nat.
```

```
Definition wartosc x := a*x*x + b*x + c.
```

```
End Trojmian.
```

```
Print wartosc.
```

```
> wartosc = fun a b c x : nat => a * x * x + b * x + c  
      : nat -> nat -> nat -> nat -> nat
```

- ▶ lokalna definicja

```
Let s := a+b.
```

```
Definition kw_sumy := s*s.
```

DEKLARACJE I DEFINICJE LOKALNE (Prop)

- ▶ lokalna deklaracja

```
Hypothesis h : False. (* odpowiednik Variable *)
```

DEKLARACJE I DEFINICJE GLOBALNE (Prop)

- ▶ globalna deklaracja

```
Axiom h:False.      (* odpowiednik Parameter *)
```

- ▶ globalna definicja (Theorem, Lemma, Corollary, Fact, Remark)

```
Theorem nat_gt_0:  
forall n:nat, n>=0.  
Proof.  
...  
Qed.
```

DOWODZENIE

- ▶ możemy definiować formuły logiczne i predykaty indukcyjne
- ▶ możemy dowodzić formuł
- ▶ formułą jest każdy term typu Prop
- ▶ dowód formuły A to term, którego typem jest A (C-H izo.)
- ▶ możemy dowód napisać wprost, lub skonstruować go interaktywnie za pomocą taktyk
- ▶ problem szukania dowodu to problem szukania termu o danym typie

DOWODZENIE

- ▶ **cel** – para złożona z (lokalnego) kontekstu typowania Γ oraz pewnego dobrze uformowanego typu T w tym kontekście (w pewnym środowisku E)
- ▶ **dowód** celu (Γ, T) – term t taki, że $E ; \Gamma \vdash t : T$
- ▶ taktyki realizują *backward reasoning*
- ▶ **taktyka** – komenda, która zastosowana do bieżącego celu g produkuje ciąg nowych celów $g_1, \dots, g_n$ na podstawie reguł typowania (bottom-up)
- ▶ taktyka zawiera przepis na skonstruowanie termu-dowodu dla bieżącego celu g mając termy-dowody dla celów $g_1, \dots, g_n$
- ▶ każdą komendę kończymy pojedynczą kropką
- ▶ początek dowodu dobrze jest zacząć komendą Proof
- ▶ koniec dowodu zapisujemy komendą Qed lub Save (powoduje sprawdzenie typu i rozszerzenie środowiska)
- ▶ nazwa twierdzenia to identyfikator termu-dowodu

RODZAJE TAKTYK

- ▶ taktyki atomowe (bazowe)
- ▶ kombinacje taktyk bazowych
- ▶ taktyki implementujące heurystyki i procedury decyzyjne (procedury języka taktyk)

TAKTYKI BAZOWE

- ▶ odpowiadają poszczególnym regułom typowania w pCIC
- ▶ taktyki odpowiadające regułom Var, Lam/Let, App:
 - `assumption, exact id` – aksjomat (identyfikator o danym typie jest już w lokalnym kontekście lub w środowisku i chcemy go wykorzystać)
 - `intro, intros, intro id, intros id1, ..., idn` – wprowadzenie hipotezy lub hipotez (implikacja, produkt zależny, `let`)
 - `apply id` – aplikacja hipotezy lub twierdzenia zdefiniowanego w bieżącym środowisku
 - `assert id:form, cut form` – odwrócenie `modus ponens`
- ▶ `intro, assumption, apply` są zupełne dla minimalnej logiki intuicjonistycznej
- ▶ taktyki “strukturalne”:
 - `clear id` – usunięcie hipotezy `id`
 - `move id after id` – permutacja hipotez

TAKTYKI DLA REGUŁY TYPOWANIA IDENTYFIKATORÓW

$$\frac{x : T \in E \cup \Gamma \quad \text{lub} \quad x := t : T \in E \cup \Gamma}{E ; \Gamma \vdash x : T}$$

- ▶ taktyki: `exact t`, `assumption`

APPLY DLA REGUŁY TYPOWANIA APLIKACJI

$$\frac{E ; \Gamma \vdash t : \text{forall } x : U, T \quad E ; \Gamma \vdash u : U}{E ; \Gamma \vdash t u : T[x \mapsto u]}$$

Działanie:

- ▶ `apply t` próbuje zunifikować cel z konkluzją termu `t` i jeśli unifikacja się powiedzie, generuje nowe podcele – odpowiadające przesłankom typu `t`

INTRO DLA REGUŁY TYPOWANIA ABSTRAKCJI

$$\frac{E; \Gamma \vdash \text{forall } x: T, U: s \quad s \in S \quad E; \Gamma, x: T \vdash t: U}{E; \Gamma \vdash \text{fun } x: T \Rightarrow t: \text{forall } x: T, U}$$

- ▶ taktyki: `intro`, `intros`, `intro id`, `intros id1 ... idn`

INTRO DLA REGUŁY TYPOWANIA DEFINICJI LOKALNEJ

$$\frac{E ; \Gamma \vdash t : T \quad E ; \Gamma, x := t : T \vdash u : U}{E ; \Gamma \vdash \text{let } x := t \text{ in } u : U[x \mapsto T]}$$

INNE TAKTYKI

- ▶ `exact term` – podanie termu `term` jako dowodu bieżącego celu
- ▶ `contradiction` – eliminacja fałszu (jeśli `False` jest w przesłankach, możemy wywnioskować wszystko)
- ▶ `reflexivity` – dowodzi celów postaci $t = u$, jeśli t i u są konwertowalne
- ▶ `auto` – heurystyka; automatycznie znajduje dowód za pomocą kombinacji taktyk `intros+assumption+apply`, działa rekurencyjnie
- ▶ `trivial` – prostsza, nierekurencyjna wersja `auto`
- ▶ `tauto` – procedura decyzyjna dla tautologii intuicjonistycznego rachunku zdań (oparta na rachunku sekwentów Dyckhoffa)
- ▶ `intuition`, `intuition tac` – korzysta z `tauto` i stosuje `tac` do dowodzenia otrzymanych celów
- ▶ `split` – wprowadzenie koniunkcji, równoważności
- ▶ `left`, `right` – wprowadzenie alternatywy
- ▶ `destruct id` – eliminacja koniunkcji/alternatywy

TAKTYKI ZŁOŻONE (*tacticals*)

Taktyki złożone można uważać za funkcje na taktykach; stanowią termy języka taktyk Ltac

- ▶ złożenie $tac_1; tac_2$ – zastosuj tac_1 do bieżącego celu, a następnie zastosuj tac_2 do **wszystkich** podcelów wygenerowanych przez tac_1
- ▶ alternatywa $tac_1 \mid\mid tac_2$ – zastosuj tac_1 ; tylko jeśli się nie uda, to zastosuj tac_2
- ▶ $idtac$, $fail$ – używane w kombinacjach taktyk: $idtac$ nie zmienia celu, $fail$ zawsze zawodzi
- ▶ $try\ tac$ – spróbuj zastosować taktykę tac ; jeśli się nie uda, to zostaw cel bez zmian

KOMENDY PRZYDATNE PRZY INTERAKTYWNYM DOWODZENIU

- ▶ `Undo`, `Undo n` – cofnij ostatni krok (taktykę) dowodu/ostanie n kroków dowodu
- ▶ `Show n` – pokaż n -ty cel z bieżących
- ▶ `Focus n` – wybierz n -ty cel z bieżących
- ▶ `Restart` – zacznij dowód od nowa
- ▶ `Admitted` – porzuć dowodzenie twierdzenia i zadeklaruj je jako aksjomat