Obliczenia i wnioskowanie w systemie Coq

Małgorzata Biernacka

Instytut Informatyki UWr

Wykład 2
28.02.2019

System Coq

- implementuje (i rozszerza) rachunek konstrukcji z definicjami indukcyjnymi:
    - Calculus of Constructions, Th. Coquand 1984
    - Calculus of Inductive Constructions, Ch. Paulin 1991
- 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.

Bezpieczeństwo i poprawność systemu

- poprawność implementacji
  - jądro Coqa – 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 Coqa
  - poprawność względna – zależy od poprawności implementacji jądra Coqa, kompilatora OCamla, sprzętu
- niesprzeczność systemu logicznego
  - pCIC jest niesprzeczny
  - aksjomaty użytkownika mogą powodować, że system staje się sprzeczny
  - bezpieczne aksjomaty

Aksjomaty niesprzeczne z pCIC

- Excluded Middle
  $\forall$ A : Prop, A $\vee$ $\neg$ A
- Proof Irrelevance
  $\forall$ A:Prop $\forall$ p1 p2:A, p1=p2
- Functional Axiom of Choice
  $\forall$ x $\exists$ y R(x,y) $\to$ $\exists$ f $\forall$ x R(x,f(x))
- Function Extensionality
  $\forall$ f g:A$\to$ B, ($\forall$ x, f(x)=g(x)) $\to$ f=g
- Predicate Extensionality
  $\forall$ P Q:A$\to$ Prop, ($\forall$ x, P(x) $\leftrightarrow$ Q(x)) $\to$ P=Q

Język Gallina

- język specyfikacji Coqa
- 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	`forall` $x : t, u$
$\lambda -$ abstrakcja	`fun` $x : t \Rightarrow u$
aplikacja	$t\ u$
konstrukcja `let`	`let` $x := t$ `in` u
definicje rekurencyjne	`fix ...`, `cofix ...`
dopasowanie wzorca	`match ... with ... end`

Podstawowe konstrukcje języka Gallina

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

Reguły typowania dla sortów

$$\frac{WF(E\,;\,\Gamma)}{E\,;\,\Gamma \vdash \texttt{Set} : \texttt{Type}_i} \qquad \frac{WF(E\,;\,\Gamma)}{E\,;\,\Gamma \vdash \texttt{Prop} : \texttt{Type}_i}$$

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

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

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

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

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

- `Prop` jest niepredykatywny
- `Set` jest predykatywny

Reguły typowania dla abstrakcji, aplikacji i definicji lokalnej

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

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

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

Podsystem CoC dla typów prostych

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{\Gamma \vdash T \to U : s \quad s \in \{Set, Prop\} \quad \Gamma, x : T \vdash t : U}{\Gamma \vdash \texttt{fun}\ x : T \ \Rightarrow\ t : T \to U}$$

$$\frac{\Gamma \vdash t : U \to T \quad \Gamma \vdash u : U}{\Gamma \vdash t\, u : T}$$

- rachunek lambda z typami prostymi
- minimalny intuicjonistyczny rachunek zdań

Język komend Vernacular

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

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 $g_1, \ldots, g_n$ na podstawie reguł typowania (bottom-up)
- taktyka zawiera przepis na skonstruowanie termu-dowodu dla bieżącego celu g mając termy-dowody dla celów $g_1, \ldots, g_n$
- 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` $\mathtt{id}_1, \ldots, \mathtt{id}_n$ – 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

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

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

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

- taktyki: `intro`, `intros`, `intro id`, `intros` $\texttt{id}_1 \; \ldots \; \texttt{id}_n$

Intro dla reguły typowania definicji lokalnej

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

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`$_1$ `||` `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

Polecenia przydatne przy dowodach

- `Undo`, `Undo n` – cofnij ostatni krok dowodu/ostatnie `n` kroków dowodu
- `Show n` – pokaż `n`-ty cel z bieżących
- `Focus n` – wybierz `n`-ty cel z bieżących (< Coq 8.8)
- `Restart` – zacznij dowód od nowa
- `Admitted` – porzuć dowodzenie twierdzenia i zadeklaruj je jako aksjomat