

Uniwersa i typy indukcyjne

19 kwietnia 2009

- Uniwersa w Coqu (przypomnienie)
- Sorty w definicjach indukcyjnych
- Dopuszczalne eliminacje
- Polimorfizm definicji indukcyjnych
- Trochę szczegółów implementacyjnych

- Sorty:

$$\begin{array}{l} Prop \\ Set \end{array} : Type_1 : Type_2 : \dots$$

Prop impredykatywny

Set, $Type_1$, $Type_2$, ... predykatywne

- Kumulatywność (v. 8.1)

$$\begin{array}{l} Prop \\ Set \end{array} \leq Type_1 \leq Type_2 \leq \dots$$

- Kumulatywność (v. 8.2)

$$Prop \leq Set \leq Type_1 \leq Type_2 \leq \dots$$

Sorty w definicjach indukcyjnych

Let E be an environment and $\Gamma, \Gamma_P, \Gamma_I, \Gamma_C$ are contexts such that Γ_I is $[I_1 : \forall \Gamma_P, A_1; \dots; I_k : \forall \Gamma_P, A_k]$ and Γ_C is $[c_1 : \forall \Gamma_P, C_1; \dots; c_n : \forall \Gamma_P, C_n]$.

$$\frac{(E[\Gamma; \Gamma_P] \vdash A_j : s'_j)_{j=1 \dots k} \quad (E[\Gamma; \Gamma_I; \Gamma_P] \vdash C_i : s_{q_i})_{i=1 \dots n}}{\mathcal{WF}(E; \text{Ind}(\Gamma)[p](\Gamma_I := \Gamma_C))[\Gamma]}$$

provided that the following side conditions hold:

- $k > 0$ and all of I_j and c_i are distinct names for $j = 1 \dots k$ and $i = 1 \dots n$,
- p is the number of parameters of $\text{Ind}(\Gamma)(\Gamma_I := \Gamma_C)$ and Γ_P is the context of parameters,
- for $j = 1 \dots k$ we have that A_j is an arity of sort s_j and $I_j \notin \Gamma \cup E$,
- for $i = 1 \dots n$ we have that C_i is a type of constructor of I_{q_i} which satisfies the positivity condition for $I_1 \dots I_k$ and $c_i \notin \Gamma \cup E$.

Dopuszczalne eliminacje

$t : I \dots$

$I : \forall \dots, s_1$

$P : \forall \dots, s_2$

`match t ... return P with ... end`

- Dla „zwykłej” definicji w sorcie *Prop* — tylko *Prop*
 $s_1 = Prop \implies s_2 = Prop$
- Dla singletonowej definicji w sorcie *Prop* — cokolwiek
- Dla definicji w sorcie *Set* lub *Type* — cokolwiek

Polimorfizm definicji indukcyjnych

Jeśli definicja jest w sortcie *Type* (i ma parametry typu *Type*),

Definition I ($P_1 : Type$) ... ($P_n : Type$) : $\forall \dots, Type := \dots$

to w typowaniu instancji „opuszcza się” sort tak nisko jak „można”.

(I $t_1 \dots t_k$) : Set

Formalnie używana jest dodatkowa reguła typowania **Ind-Family**, która chwilowo zastępuje **Ind-Const** i **App**, dając typ mniejszy w sensie \leq

Co to znaczy „można”?

- definicja z nowymi sortami musi być poprawna
- nie zmieniają się dopuszczalne eliminacje (rzadko osiągamy sort *Prop* — tylko w przypadku definicji „singletonowych”)

Trochę szczegółów implementacyjnych

- Nie chcemy na stałe przypisywać numerów (bo nie ma po co)
- Każde pojawienie się słowa `Type` w źródłach powoduje „wygenerowanie” nowej stałej uniwersowej
- Proces typowania „generuje” więzy uniwersowe

$$u_i < u_j, \quad u_i = u_j, \quad u_i \leq u_j$$

które są przechowywane w środowisku

- Jak więzy są nierozwiązywalne — zgłaszany jest błąd `Universe Inconsistency`
- W typach występują też wyrażenia uniwersowe:
 $u_i + 1, \quad \max(u_j, \dots, u_j)$
które można zobaczyć poprzez `Set Printing Universes`.
- Dla ciekawskich:

`Print Universes.` lub nawet

`Print Universes "plik.txt".` (i potem ewentualnie)

`coq-src/dev/tools/univdot plik.txt | dot -Tps | gv -`

Cześć II. Jeszcze trochę taktyk

taktyki konwersji
automatyzacja

Taktyki konwersji

- `unfold ld` lub `unfold ld at num1 ... numn` lub `unfold "symb"`
rozwija (niektóre) wystąpienia *ld* lub *symb* i $\beta\iota$ normalizuje
- `fold term`
zwija formę $\beta\iota\zeta$ normalną *termu*
- `simpl` lub `simpl ld` lub `simpl term`
rozwija definicje o ile da się po niej wykonać ι redukcję, β normalizuje, następnie zwija fixpoint z powrotem do stałej
- `red`
rozwija stałą w głowie termu (pod produktem) i $\beta\iota\zeta$ normalizuje
- `hnf` — normalizacja, ale nie wchodzi pod produkt
- `compute` lub `cbv` lub `lazy`
oblicza formę normalną używając tej lub innej strategii
- warianty: `cbv delta [Id1 Id2]` `iota zeta`
lub `lazy beta delta -[Id1 Id2]`
- `tac in H` — wykonuje konwersję *tac* w hipotezie *H*

Taktyki konwersji (c.d.)

- `change term` lub
change $term_1$ with $term_2$ lub
change $term_1$ at $num_1 \dots num_i$ with $term_2$ lub
change ... in *ident*
zmienia cel/hipotezę na konwertowalny
- `pattern term` lub
pattern $term$ at $num_1 \dots num_n$ lub
pattern $term$ at - $num_1 \dots num_n$
zastępuje cel postaci $\phi(term)$ w $((\text{fun } x:A \Rightarrow \phi(x)) term)$.

Taktyki automatyczne

auto lub auto n lub auto with $baza_1 \dots baza_n$ lub auto with *

Korzysta z „bazy” lematów próbując robić nimi apply, czasami unfold, assumption oraz intro. Postępuje wgłąb, na głębokość 5 lub n , kierując się stałą w głowie celu oraz kosztem akcji.

- Hint Resolve Id lub Hint Resolve $Id_1 \dots Id_n : baza$
dodaje lematy do „bazy”; koszt to liczba generowanych celów
- Hint Unfold Id lub ...
dodaje informację, że daną stałą należy rozwijać; koszt to 4
- Hint Constructors I lub ...
dodaje wszystkie konstruktory typu indukcyjnego I
- Print Hint lub Print Hint id lub Print Hint *
wypisuje hinty dostępne dla stałej w głowie bieżącego celu, dla stałej id lub wszystkie.
- Create HintDb $baza$ oraz Print HintDb $baza$
tworzy / wypisuje zawartość „bazy”

Taktyki automatyczne (c.d.)

auto i spółka

- `trivial` lub `trivial with ...`
nierekurencyjna wersja `auto`, próbująca tylko akcji o koszcie 0
- `Hint Immediate Id` lub ...
jak `Hint Resolve`, ale żądamy, żeby wszystkie wygenerowane cele były rozwiązywalne przez `trivial` — do lematów, które można by stosować wiele razy
- `eauto ...` pozwala na używanie zmiennych egzystencjalnych (np. tranzytywność równości) — zwykle działa dłużej niż `auto`, czasem lepiej
- `Hint Extern n [pattern] => tac : baza`
pozwala na dołączenie taktyki `tac` o koszcie `n` do „bazy”. Taktyka ta będzie próbowana o ile cel pasuje do `pattern` lub zawsze.

UŻYWAJCIE `auto`!!!!!!!!!!!!!!

Taktyki automatyczne (c.d.)

`tauto` i `intuition`. Używając systemu sekwentów Dyckhoffa:

- `tauto` udowadnia instancje zdaniowych tautologii intuicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów
- `intuition` lub `intuition tac`
działa jak `tauto`, ale jak już nie ma co robić, próbuje `tac` lub `auto` with *

inne:

- `firstorder` — experimentalne rozszerzenie `tauto` na logikę pierwszego rzędu
- `omega` — procedura decyzyjna dla arytmetyki Presburgera. Działa na formułach bez kwantyfikatorów zawierających spójniki logiczne, równości, nierówności, różności, `+`, `-`, `*`, `pred`, `S`, `O` i stałe liczbowe dla `nat` oraz `Z`. Wymaga `Require Import Omega`.
- `ring` — dowodzi równości wielomianów
- `field` — dowodzi równości ilorazów wielomianów