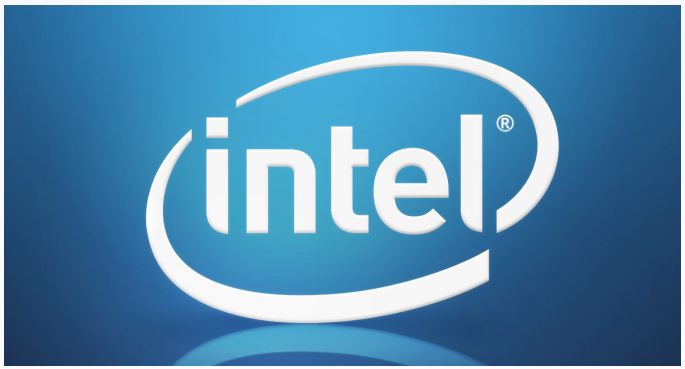
Constructing Trusted Code Base V

Aleksy Schubert & Jacek Chrząszcz



Coq - inductive definitions - Introduction tactics

- apply c_i
- constructor i
- constructor
- left, right = constructor 1, constructor 2
  (if there are only 2 constructors)
- split = constructor 1 = constructor
  (if there are only 1 constructor)
- exists t = split with t
  (if there are only 1 constructor)

Destruction tactics

- `destruct`
- `simple destruct`
- `case` (basic)

- `intros (H1,H2)` — "conjunction"
- `intros [H1|H2]` — "disjunction"

Induction tactics

- `induction`
- `simple induction`
- `elim` (basic)
- `elimtype`

For mutually inductive types automatic lemmas are good for nothing.
To generate good ones use:

`Scheme` $\mathtt{id}_1$ `:= Induction for` I_i `Sort` s_1
`with ...`
$\mathtt{id}_n$ `:= Induction for` I_n `Sort` s_n

or non-dependent version:

`Scheme` $\mathtt{id}_1$ `:= Minimality for` I_i `Sort` s_1
`with ...`
$\mathtt{id}_n$ `:= Minimality for` I_n `Sort` s_n

Equality inductive type

- Leibniz equality `eq`
- direct proving: `reflexivity` or `split` or `constructor` or `apply eq_refl`
- proving: `symmetry`, `transitivity`
- use: `rewrite`, `rewrite <-`, `subst`, `replace`

Inductive types and equality

- `0=1 -> ?` `discriminate`
- `S x = S y -> x = y ?` `injection`
- either one: `simplify_eq`

Advanced destruction

How this could happen ?

- `inversion`
  (conclusions from (Even n) and discriminate and injection)
- `inversion_clear`
  (as above, but does some cleaning)
- `dependent inversion`
  (if the destructed name occurs in the “goal”)

Taktyki konwersji

- `unfold` *Id* lub `unfold` *Id* `at` num_1 ... num_n lub `unfold` "*symb*"
  rozwija (niektóre) wystąpienia *Id* lub *symb* i $\beta\iota$ normalizuje
- `fold` *term*
  zwija formę $\beta\iota\zeta$ normalną *termu*
- `simpl` lub `simpl` *Id* lub `simpl` *term*
  rozwija definicje o ile da sie 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 [Id`$_1$ `Id`$_2$`] iota zeta`
  lub `lazy beta delta -[Id`$_1$ `Id`$_2$`]`

Taktyki konwersji (c.d.)

- `change` $term$ lub
  `change` $term_1$ `with` $term_2$ lub
  `change` $term_1$ `at` num_1 `...` num_i `with` $term_2$ lub
  `change ... in` $ident$
  zmienia cel/hipotezę na konwertowalny
- `pattern` $term$ lub
  `pattern` $term$ `at` num_1 `...` num_n lub
  `pattern` $term$ `at -` num_1 `...` num_n
  zastępuje cel postaci $\phi(term)$ w `((fun x:`A `=>` ϕ`(x))` $term$`)`.

Taktyki automatyczne

`auto` lub `auto` n lub `auto with` $baza_1 ... baza_n$ lub `auto with *`
Korzysta z „bazy" lematów probując 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 ... 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 intuinicjonistycznych; 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` — experymentalne 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