Иерархии универсумов

Иерархия универсумов

- Обобщённая типовая система:

Название сорта	Примеры сортов	Примеры конструкций, имеющих сорт
Тип	α, $\alpha \to \beta$, $\star \to \alpha$	$3 : \texttt{int}, \texttt{id} : \forall \alpha . \alpha \to \alpha$
Род (kind)	$\star$, $\star \to \star$, $\alpha \to \star$	$\texttt{list} : \star \to \star$
Сорт	$\square$	$\star \to \star : \square$

  Также, сорт функции классифицируется по сорту её результата.
- В Аренде сорт (тип) характеризуется двумя параметрами:

  ```
  \h-Type m
  ```

  что также записывается как

  ```
  \Type m h
  ```

  Здесь *h* — гомотопический уровень, *m* — предикативный уровень.

Prop и Set в Coq

- Вселенная Prop — типы чистых доказательств
- Вселенная Set — типы значений
- Type — совокупность Prop и Set
- Ожидаем, что Prop сами исчезнут из кода конструктивных доказательств при оптимизации.
- Однако, разделение на вселенные произвольное. Скажем, есть несколько доказательств того, что 15 — составное:
  $((3,5), 3 \cdot 5 = 15) : \exists(x,y).x \cdot y = 15$ и $((5,3), 5 \cdot 3 = 15) : \exists(x,y).x \cdot y = 15$
  Не всегда доказательство единственно.
- Тип-сумма есть в двух вариантах:

Prop	Set
Дизъюнкция: `a \/ b`	Тип-сумма: `{a} + {b}`

Prop и Set в алгоритме Эвклида

- Равенство — это Prop:

```
Inductive eq (A:Type) (x:A) : A -> Prop := eq_refl : x = x :>A
```

- А типы «a делится с остатком на b», «$n \leqslant m$ или $n > m$» — это Set:

```
Inductive diveucl a b : Set :=
  divex : forall q r, b > r -> a = q * b + r -> diveucl a b.
Definition le_gt_dec n m : {n <= m} + {n > m}. ... Defined.
```

- Конструктивное доказательство алгоритма Эвклида — это смесь Prop и Set:

```
Lemma eucl_dev : forall n, n > 0 -> forall m:nat, diveucl m n.
Proof.
  intros b H a; pattern a in |- *; apply gt_wf_rec; intros n H0.
  elim (le_gt_dec b n).
  intro lebn. elim (H0 (n - b)); auto with arith.
  intros q r g e.
  apply divex with (S q) r; simpl in |- *; auto with arith.
  elim plus_assoc. elim e; auto with arith.
  intros gtbn.
  apply divex with 0 n; simpl in |- *; auto with arith.
Qed.
```

Извлечение кода из доказательства

Напомним определения:

```
Inductive diveucl a b : Set :=
  divex : forall q r, b > r -> a = q * b + r -> diveucl a b.

Lemma eucl_dev : forall n, n > 0 -> forall m:nat, diveucl m n.
```

Извлечённый код содержит только относящиеся к Set термы:

```
type diveucl = Divex of nat * nat

let rec eucl_dev n m =
  let s = le_gt_dec n m in
  (match s with
   | Left ->
     let d = let y = sub m n in eucl_dev n y in
     let Divex (q, r) = d in Divex ((S q), r)
   | Right -> Divex (O, m))
```

Гомотопическая иерархия универсумов

HoTT предлагает естественный смысл универсумам.

Определение

Тип τ — это n-тип, если тип равенств $\tau = \tau$ есть $(n-1)$-тип. Универсум Prop содержит (-1)-типы, все такие типы имеют ровно одно значение.

Название	Уровень	
Prop	-1	Значения единственны
Set	0	Путь между значениями единственен

Гомотопическая иерархия в Аренде

- Компилятор пытается угадать уровень.
- Компилятору можно подсказать.
- Уровень типа можно принудительно изменить.

Пример подсказки: разрешимость

Разрешимое высказывание: такое, для которого существует алгоритм, возвращающий yes — если высказывание доказуемо, и no — если доказуемо отрицание.

```
\data Dec (P : \Prop) | yes P | no (P -> Empty)
```

Выглядит как индуктивный тип с двумя вариантами (`Dec P : \Set`). Однако, если $\vdash x : \texttt{yes}\ P$ и $\vdash y : \texttt{no}\ (P \to \bot)$, то $\vdash (\dots) : \bot$. Потому:

```
\data Dec (P : \Prop) | yes P | no (P -> Empty)
  \where
    \use \level isProp {P : \Prop} (d1 d2 : Dec P) : d1 = d2
      | yes x1, yes x2 => path (\lam i => yes (Path.inProp x1 x2 @ i))
      | yes x1, no e2 => \case e2 x1 \with {}
      | no e1, yes x2 => \case e1 x2 \with {}
      | no e1, no e2 => path (\lam i => no
           (\lam x => (\case e1 x \return e1 x = e2 x \with {}) @ i)
        )
```

Пропозициональное усечение

При необходимости тип можно обернуть в структуру с нужным гомотопическим уровнем.

```
\data TruncP (A : \Type)
  | inP A
  | truncP (a a' : TruncP A) : a = a'
  \where {
    \use \level levelProp {A : \Type} (a a' : TruncP A) : a = a'
       => path (truncP a a')
  }
```

Или явно её задать с нужным уровнем (и нужным равенством):

```
\truncated \data Quotient {A : \Type} (R : A -> A -> \Type) : \Set
  | in~ A
  | ~-equiv (x y : A) (R x y) (i : I) \elim i {
    | left => in~ x
    | right => in~ y
  }
```

Логические конструкции и гомотопический уровень

Конструкции, не повышающие гомотопический уровень: конъюнкция, импликация, квантор всеобщности.
Это свойство можно сформулировать и доказать:

```
\func isProp (A : \Type) => \Pi (a a' : A) -> a = a'
\func and (a b : \Prop) : isProp (\Sigma a b) => \lam p q => {?}
```

Дизъюнкция в Аренде

Повышает гомотопический уровень, не проп:

```
\data \fixr 2 Or (A B : \Type)
  | inl A
  | inr B
  \where {
    \func levelProp {A B : \Prop} (e : A -> B -> Empty) (x y : Or A B)
        : x = y \elim x, y
      | inl a, inl a' => pmap inl (Path.inProp a a')
      | inl a, inr b => absurd (e a b)
      | inr b, inl a => absurd (e a b)
      | inr b, inr b' => pmap inr (Path.inProp b b')
  }
```

И есть усечённый до пропа вариант:

```
\truncated \data \infixr 2 || (A B : \Type) : \Prop
  | byLeft A
  | byRight B
```

Квантор существования в Аренде

Квантор существования также увеличивает уровень. Не-проп вариант:

```
\func divides-set (a b : Nat) => \Sigma (p : Nat) (a = b Nat.* p)
```

Усечённый (проп) вариант:

```
\func divides' (a b : Nat) => TruncP (\Sigma (p : Nat) (a = b Nat.* p))
```

```
-- Встроенная в Аренд мета:
\func divides (a b : Nat) => Exists (p : Nat) (a = b Nat.* p)
```

Вместо `Exists` можно писать ∃.

Каков тип типа?

Напомним из обобщённой типовой системы:

$$\texttt{Nat} : \star \qquad \star : \square \qquad \square : ?$$

Такие сложные конструкции могут быть содержательны:

```
\func Church => \Pi (a : \Type) -> ((a -> a) -> a -> a)
\func zero : Church => \lam a (f : a -> a) (x : a) => x

\func inc : Church -> Church => \lam n
    => \lam (a : \Type) (f : a -> a) (x : a) => n a f (f x)

\func add : Church -> Church -> Church => \lam m n => m Church inc n
```

Парадокс Жирара

Однако, попытки ввести тип всех типов легко могут привести к противоречию.

Определение (система U^-)

U^- — это обобщённая типовая система с тремя сортами: $\{\star, \Box, \triangle\}$, двумя аксиомами:

$$\vdash \star : \Box \qquad \vdash \Box : \triangle$$

и следующие частными правилами вывода:

$$\{\langle\star,\star\rangle, \langle\Box,\star\rangle, \langle\Box,\Box\rangle, \langle\triangle,\Box\rangle\}$$

То есть, это $\lambda\omega$ с $\langle\triangle,\Box\rangle$.

Теорема (парадокс Жирара)

В типовой системе U^- тип $\Pi p^\star.p$ обитаем.

Предикативная иерархия

Определение

Атомарные типы имеют предикативный уровень 0. Составной тип имеет тип с предикативным уровнем строго выше уровней его составных частей.

$$\varphi(\backslash\mathit{Type}\ m_1, \ldots, \backslash\mathit{Type}\ m_n) : \backslash\mathit{Type}\ \max(m_1, \ldots, m_n) + 1$$

Пример

$(\star)$ *можно понимать как аналог* $\backslash\mathit{Type}\ 0$. *Тогда* $(\star \to \star) \to \star : \backslash\mathit{Type}\ 1$.
Далее, $\backslash\mathit{Type}\ 1 : \backslash\mathit{Type}\ 2$ *и т.п.*

```
\func star-star-star : (\Type 0 -> \Type 0) -> \Type 0 => {?}
\func id {t : \Type 1} (x : t) => x
\func x : (\Type 0 -> \Type 0) -> \Type 0 =>
          id {(\Type 0 -> \Type 0) -> \Type 0} star-star-star
```