Доказательство леммы Хигмана (для двух букв) формализованное на Агде

С.А.Романенко

июль 2017

Вложение Хигмана

Слово v *вкладывается* в слово w, если можно получить v из w, вычеркнув некоторые буквы в w.

> Обозначение: `v ⊴ w`.

Мнемоника: w "больше", чем v, ибо содежит больше "мяса".

```
a :: b :: a :: c :: b ⊴ c :: a :: b :: b :: a :: b :: c :: b :: b :: c
```

```
  a b   a   c   b
c a b b a b c b b c
```

Очевидные свойства:

- `[] ⊴ w`
- `v ⊴ w → v ⊴ a :: w`
- `v ⊴ w → a :: v ⊴ a :: w`

Лемма Хигмана

"Хорошие" последовательности

Любая бесконечная последовательность слов $(w_i)_{0 \leq i < \omega}$ называется "хорошей", если она содержит такие два слова w_i и w_j, что $i < j$ и $w_i \trianglelefteq w_j$.

Лемма

Если алфавит ***конечен***, то любая бесконечная последовательность слов $(w_i)_{0 \leq i < \omega}$ является "хорошей".

- G. Higman. 1952. **Ordering by divisibility in abstract algebras.** Proceedings of the London Mathematical Society, 3(2):326–336. DOI=10.1112/plms/s3-2.1.326

Классическое доказательство для 2-х букв

- C. Nash-Williams. 1963. **On well-quasi-ordering finite trees.** Proceedings of the Cambridge Philosophical Society 59(4), 833–835. DOI=10.1017/S0305004100003844

Схема доказательства (от противного):

- Пусть множество плохих последовательностей не пусто.
  - Упорядочим его лексикографически (по длине слов).
  - Пусть `ws` - минимальная плохая последовательность.
- Изготовим из `ws` её "урезанную" версию `vs` (вычеркивая буквы из слов и выбрасывая некоторые слова).
- `vs` - меньше, чем `ws`, значит - она хорошая. Докажем, что хорошесть `vs` влечет хорошесть `ws`.
- `ws` - хорошая. Но она - плохая. Противоречие!

Выбор "минимальной" последовательности

Начинаем с пустой последовательности. Далее - наращиваем последовательность итеративно.

Пусть уже выбраны k слов w_0 , ... , w_{k-1}. Выбираем следующее слово w_k, удовлетворяющее двум требованиям.

1. w_0 , ... , w_{k-1}, w_k является началом хотя бы одной плохой последовательности.
2. Если слово v получается отбрасыванием от w_k одной или нескольких первых букв, то все последовательности, которые начинаются с w_0 , ... , w_{k-1}, v - хорошие.

Затем, используя "аксиому зависимого выбора", делаем вывод, что существует бесконечная последовательность $(w_i)_{0 \leq i < \omega}$ (которую мы и называем "минимальной").

"Подрезание" последовательностей

- Если w содержит $w_i = []$, то оно вложится в w_{i+1}. QED
- В противном случае, $w_i = c_i :: v_i$.
- Рассмотрим c_i. Тогда хотя бы одна из букв в c_i будет повторяться бесконечное число раз! Предположим, что это буква a. (Нетривиальное место! Вспомним про ультрафильтры.)
- Выполняем "урезание по a".
  - Слова вида $a :: w$ заменим на w.
  - Слова вида $b :: w$ удалим.
  - Для начальных $b :: w$ (до первого $a :: w$) - исключение, их оставляем без изменения.

Хорошо в урезанном ⟶ хорошо в полном

`T a vs ws → Good vs → Good ws`

Вложения 3-х типов: $y \trianglelefteq y$, $y \trianglelefteq x$, $x \trianglelefteq x$.

ws	vs	v ⊴ v'	w ⊴ w'
$b :: y_1$	$b :: y_1$	$b :: y_1 \trianglelefteq b :: y_2$	$b :: y_1 \trianglelefteq b :: y_2$
$b :: y_2$	$b :: y_2$	$b :: y_2 \trianglelefteq x_1$	$b :: y_2 \trianglelefteq a :: x_1$
$a :: x_1$	x_1		
$b :: y_3$			
$a :: x_2$	x_2	$x_2 \trianglelefteq x_3$	$a :: x_2 \trianglelefteq a :: x_3$
$b :: y_4$			
$a :: x_3$	x_3		
…	…	…	…

Избавление от неконструктивности (1)

Какие проблемы?

- Доказательство от противного подразумевает использование закона исключенного третьего.
  А откуда-ж его взять?
- Нужно выбрать букву, которая повторяется бесконечное число раз. А для этого надо иметь всю бесконечную последовательность целиком. Как? "Аксиома выбора - наше всё"? Ну-ну!
- И вообще, как конструктивно работать с актуальными бесконечными последовательностями? А нужно это?

Избавление от неконструктивности (2)

Убираем актуально бесконечные последовательности

Хорошесть - это "открытое свойство": если оно выполнено для всей бесконечной последовательности `ws`, то существует начальный отрезок `ws` из свойств которого это следует.

Убираем "отрицалово" - мыслим позитивно

Зачем доказывать, что "неверно, что существует последовательность, которая не является хорошей"?
Лучше докажем, что при наращивании любой конечной последовательности она обязательно становится хорошей.

Убираем преждевременный выбор

Раз мы не знаем, какая из букв повторится бесконечное число раз, будем наблюдать параллельно за двумя буквами.

Ссылки

- J.-C. Raoult. 1988. **Proving open properties by induction.** Inf. Process. Lett. 29, 1 (September 1988), 19-23. DOI=0020-0190(88)90126-3
- T. Coquand, D. Fridlender. 1993. **A proof of Higman’s lemma by structural induction.** Unpublished draft (November 1993), available at http://www.math.chalmers.se/~frito/Papers/open.ps.gz
- S. Berghofer. 2004. **A constructive proof of Higman's lemma in Isabelle.** In *Types for Proofs and Programs, TYPES'04*. LNCS, 3085: 66-82. Springer Verlag, 2004. DOI=10.1007/978-3-540-24849-1_5

Формализация на Агде

Доказательство на Агде - в репозитории

> https://github.com/sergei-romanenko/agda-Higman-lemma

в папке `Berghofer`.

Получено переписыванием доказательства Бергхофера с Изабели на Агду.

> Итак - приступаем!

Буквы, слова, последовательности

```
data Letter : Set where
  l0 l1 : Letter

Word = List Letter
```

Некоторые свойства множества из двух букв

```
data _<>_ : (a b : Letter) → Set where
  l0<>l1 : l0 <> l1
  l1<>l0 : l1 <> l0

<>-sym : ∀ {a b} → a <> b → b <> a
≡⊎<> : ∀ a b → a ≡ b ⊎ a <> b
dirichlet2 : ∀ {a b} → a <> b → ∀ c → c ≡ a ⊎ c ≡ b
```

Доказательства - разбором случаев, **без индукции**! 😄

Вложение Хигмана

```
data _⊴_ : (v w : Word) → Set where
  ⊴-[]   : [] ⊴ []
  ⊴-drop : ∀ {v w a} → v ⊴ w → v ⊴ a ∷ w
  ⊴-keep : ∀ {v w a} → v ⊴ w → a ∷ v ⊴ a ∷ w
```

Теорема: `[]` вкладывается в любое слово `w`

```
[]⊴ : ∀ w → [] ⊴ w
[]⊴ [] = ⊴-[]
[]⊴ (a ∷ w) = ⊴-drop ([]⊴ w)
```

Первое доказательство **по индукции**! 💥✨👏

Конечные последовательности

Конечные последовательности можно было бы представить списками слов: `List Word`. Но это - не очень хорошо с "педагогической" и психологической точек зрения.
Поэтому, определяем последовательности так:

```
data Seq : Set where
  ε : Seq
  _#_ : Seq → Word → Seq
```

В некоторых случаях, требуется рассматривать непустые последовательности.

```
data ε≠ : Seq → Set where
  ε≠# : ∀ {ws w} → ε≠ (ws # w)
```

Хорошесть (конечной) последовательности

`ws ∋◃ v` - некоторое слово из `ws` вкладывается в `v`.

```
data _∋◃_ : (ws : Seq) (v : Word) → Set where
  here  : ∀ {w ws v} (w◃v : w ◃ v) → (ws # w) ∋◃ v
  there : ∀ {w ws v} (ws∋◃v : ws ∋◃ v) → (ws # w) ∋◃ v
```

`Good (ws # w)`:

- либо вложение чего-то из `ws` в `w`,
- либо вложение внутри `ws`.

```
data Good : (ws : Seq) → Set where
  here  : ∀ {ws w} (ws∋◃w : ws ∋◃ w) → Good (ws # w)
  there : ∀ {ws w} (good-ws : Good ws) → Good (ws # w)
```

"Подойдите к барьеру! И вам станет хорошо..."

- Если счастье наступило - на том и конец.
- Если пока не наступило - наступит позже.
- Можно шагать вправо, можно - влево, а хана (счастье) всё равно будет... (По принципу "витязь на распутье".)

```
mutual

  data Bar : Seq → Set where
    now   : ∀ {ws} (n : Good ws) → Bar ws
    later : ∀ {ws} (l : Later ws) → Bar ws

  Later : Seq → Set
  Later ws = ∀ w → Bar (ws # w)
```

Другими словами, "как последовательность ни продолжай", а всё равно к счастью придешь...

Урезывание `ws` до `vs` по `a`

Добавление `a` к словам в `ws`

```
_::∈_ : (a : Letter) (ws : Seq) → Seq
a ::∈ ε = ε
a ::∈ (ws # w) = (a ::∈ ws) # (a :: w)
```

`vs` результат урезания `ws` по `a`

```
data T (a : Letter) : (vs ws : Seq) → Set where
  init : ∀ {w ws b} (a<>b : a <> b) →
           T a ((b ::∈ ws) # w) ((b ::∈ ws) # (a :: w))
  keep : ∀ {w vs ws} →
           T a vs ws → T a (vs # w) (ws # (a :: w))
  drop : ∀ {w vs ws b} (a<>b : a <> b) →
           T a vs ws → T a vs (ws # (b :: w))
```

Полезные свойства `T` и `Good`

Пустое слово, вложится в любое следующее. (И будет хорошо.)

```
bar-#[] : (ws : Seq) → Bar (ws # [])
bar-#[] ws = later (λ w → now (here (here ([]⊴ w))))
```

"Подъём на букву" кашу не портит.

```
good∷ : ∀ {a ws} → Good ws → Good (a ∷∈ ws)
```

"Коронное свойство": хороша урезанная - хороша и полная.

```
t-good : ∀ {a vs ws} → T a vs ws → Good vs → Good ws
```

Частный случай урезания для непустых последовательностей.

```
t-ε≠ : ∀ a ws → ε≠ ws → T a ws (a ∷∈ ws)
```

Сборка полного счастья из урезанного (1)

Урежем `zs` по буквам `a` `b`, получив `xs` `ys`. Если их продолжения всегда ведет к счастью, то к счастью ведут и все продолжения `zs`.

```
tt-bb : ∀ {zs a b xs ys} → a <> b → T a xs zs → T b ys zs →
          Bar xs → Bar ys → Bar zs
```

`T a xs zs → Good xs → Good zs`

Иначе - счастье в `xs` наступит позже. Смотрим на `ys`.

```
tt-bb a<>b ta tb (now nx) bar-ys =
  now (t-good ta nx)
tt-bb a<>b ta tb (later lx) bar-ys =
  tt-lb a<>b ta tb lx bar-ys
```

Сборка полного счастья из урезанного (2)

```
tt-lb : ∀ {zs a b xs ys} → a <> b → T a xs zs → T b ys zs →
          Later xs → Bar ys → Bar zs
```

`T a ys zs → Good ys → Good zs`

Иначе - счастье в `ys` наступит позже. Нужно изучить возможные продолжения `zs`.

```
tt-lb a<>b ta tb lx (now ny) =
  now (t-good tb ny)
tt-lb a<>b ta tb lx (later ly) =
  later (tt-ll a<>b ta tb lx ly)
```

Сборка полного счастья из урезанного (3)

```
tt-ll : ∀ {zs a b xs ys} → a <> b → T a xs zs → T b ys zs →
          Later xs → Later ys → Later zs
```

Вспоминаем, что `Later zs` - это `∀ w → Bar (zs # w)`.

Изучаем следующее слово в продолжении `zs`.

Если оно пустое, то оно вложится в любое следующее слово!

```
tt-ll {zs} a<>b ta tb lx ly [] = bar-#[] zs
```

Иначе, оно имеет вид `c ∷ v`, где либо `c ≡ a`, либо `c ≡ b`.

```
tt-ll {zs} {a} {b} {xs} {ys} a<>b ta tb lx ly (c ∷ v)
  with dirichlet2 a<>b c
... | inj₁ c≡a rewrite c≡a = ?
... | inj₂ c≡b rewrite c≡b = ?
```

Сборка полного счастья из урезанного (4)

Если `c ≡ a`, то

- `zs` наращивается до `zs # (a ∷ v)`.
- `xs` наращивается до `xs # v`.
- `ys` не изменяется.

```
... | inj₁ c≡a rewrite c≡a =
  Bar (zs # (a ∷ v)) ∋
  tt-bb a<>b ta' tb' (lx v) (later ly)
  where ta' : T a (xs # v) (zs # (a ∷ v))
        ta' = keep ta
        tb' : T b ys (zs # (a ∷ v))
        tb' = drop (<>-sym a<>b) tb
```

Вызываем рекурсивно `tt-bb`. `bar-xs` - уменьшился.

Сборка полного счастья из урезанного (5)

Если `c ≡ b`, то

- `zs` наращивается до `zs # (b ∷ v)`.
- `xs` не изменяется.
- `ys` наращивается до `ys # v`.

```
... | inj₂ c≡b rewrite c≡b =
  Bar (zs # (b ∷ v)) ∋
  tt-lb a<>b ta' tb' lx (ly v)
  where ta' : T a xs (zs # (b ∷ v))
        ta' = drop a<>b ta
        tb' : T b (ys # v) (zs # (b ∷ v))
        tb' = keep tb
```

Вызываем рекурсивно `tt-lb`. `bar-xs` - не изменился,

а `bar-ys` - уменьшился.

Сборка полного счастья из урезанного (6)

Почему рекурсия "правильная"?

Структура вызовов - такая:

```
tt-bb a<>b ta tb (later lx) bar-ys =
  tt-lb a<>b ta tb lx bar-ys

tt-lb a<>b ta tb lx (later ly) =
  later (tt-ll a<>b ta tb lx ly)

tt-ll {zs} {a} {b} {xs} {ys} a<>b ta tb lx ly (c :: v) =
  ...
  tt-bb a<>b ta' tb' (lx v) (later ly)
  ...
  tt-lb a<>b ta' tb' lx (ly v)
```

Лексикографический порядок: `lx`, `ly`.

Уменьшается либо `lx`, либо `ly` (при неизменном `lx`).

"Подъем на букву" сохраняет счастье (1)

```
bar::∈ : ∀ b ws → ε≢ ws → Bar ws → Bar (b ::∈ ws)
```

Если `Good ws`, то `Good (b ::∈ ws)` (как было уже доказано).

Иначе - заглядываем на слово вперед.

```
bar::∈ b ws ε≢ws (now n) = now (good:: n)
bar::∈ b ws ε≢ws (later l) = later (later::∈ b ws ε≢ws l)
```

Пришло пустое слово - будет хорошо (как было уже доказано).

```
later::∈ : ∀ b ws → ε≢ ws → Later ws → Later (b ::∈ ws)
later::∈ b ws ε≢ws l [] = bar-#[] (b ::∈ ws)
```

"Подъем на букву" сохраняет счастье (2)

Если следующее слово не пустое, оно имеет вид `a ∷ w`.

Два случая: `a≡b` или `a<>b`.

```
later∷∈ b ws ε≢ws l (a ∷ w) with ≡⊎<> a b
... | inj₁ a≡b rewrite a≡b =
  bar∷∈-b b ws w ε≢ws l
... | inj₂ a<>b =
  bar∷∈-a b ws a w a<>b ε≢ws l
```

"Подъем на букву" сохраняет счастье (3)

Если `a ≡ b`, то `(b ::∈ ws) # (b :: w)` - это `b ::∈ (ws # w)`.

И можно применить гипотезу индукции.

При этом `later l` заменяется на `l w`. Уменьшается!

```
bar::∈-b : ∀ b ws w → ε≠ ws →
  Later ws → Bar (b ::∈ (ws # w))
bar::∈-b b ws w ε≠ws l =
  bar::∈ b (ws # w) ε≠# (l w)
```

"Подъем на букву" сохраняет счастье (4)

Если `a <> b`, при добавлении `a :: w` к `b ::∈ ws`, гипотеза индукции не получается. Вперед идти не можем! Поэтому - сделаем шаг "вбок", добавив `w` к `b ::∈ ws`.

```
bar::∈-a : ∀ b ws a w → a <> b → ε≠ ws →
  Later ws → Bar ((b ::∈ ws) # (a :: w))
bar::∈-a b ws a w a<>b ε≠ws l =
  tt-bb a<>b t1 t2 b1 b2
  where t1 : T a ((b ::∈ ws) # w) ((b ::∈ ws) # (a :: w))
        t1 = init a<>b
        b1 : Bar ((b ::∈ ws) # w)
        b1 = later::∈ b ws ε≠ws l w
        t2 : T b ws ((b ::∈ ws) # (a :: w))
        t2 = drop (<>-sym a<>b) (t-ε≠ b ws ε≠ws)
        b2 : Bar ws
        b2 = later l
```

"Подъем на букву" сохраняет счастье (5)

Почему рекурсия "правильная"?

Структура вызовов - такая:

```
bar::∈ b ws ε≠ws (later l) =
  later (later::∈ b ws ε≠ws l)

later::∈ b ws l (a :: w) with ≡⊎<> a b =
  ...
  bar::∈ b (ws # w) ε≠# (l w)
  ...
  later::∈ b ws ε≠ws l w
```

Лексикографический порядок: `bar-ws`, `w`.

Уменьшается либо `bar-ws`, либо `w` (при неизменном `bar-ws`).

Все пути из `ε` ведут к счастью...

Если формально - то `Bar ε`.

Однако: 千里之行，始于足下 (qiān lǐ zhī xíng, shǐ yú zú xià)
"Путь в тысячу ли начинается с первого шага!"

Вот и смотрим на первое слово.
Если `[]`, то `[]` вложится в любое следующее `w`.
Если `c ∷ w`, то отбрасываем `c`, доходим до счастья -
и возвращаем `c` с помощью `bar∷∈ c`.

```
later-ε :  Later ε
later-ε [] = bar-#[] ε
later-ε (c ∷ w) = bar∷∈ c (ε # w) ε≠## (later-ε w)

bar-ε : Bar ε
bar-ε = later later-ε
```

Конец доказательства леммы Хигмана 🌟

Все пути начинаются с `ε`.

Из `Bar ε` следует, что все пути из `ε` ведут к счастью.

А если мы уже прошли часть пути `ws`, неизбежно ли счастье?

Да! Вернемся назад к `ε`, а потом повторим путь `ws`.

Классический метод (любимый математиками):

> *"Вылить из чайника воду и свести задачу к предыдущей".*

```
bar-# : ∀ w ws → Bar ws → Bar (ws # w)
bar-# w ws (now n) = now (there n)
bar-# w ws (later l) = l w

higman : ∀ ws → Bar ws
higman ε = bar-ε
higman (ws # w) = bar-# w ws (higman ws)
```

А бесконечные последовательности?

Как задать бесконечную последовательность конструктивно?

Можно? Таки да! В виде функции `f : ℕ → Word`.

И можно определить понятие **конечного** начального участка:

```
data Prefix (f : ℕ → Word) : ℕ → Seq → Set where
  zero : Prefix f zero ε
  suc  : ∀ {i xs} → Prefix f i xs →
                    Prefix f (suc i) (xs # f i)
```

Если бесконечная последовательность задана функцией `f`, можно ли отрезать от неё хороший начальный участок?

Да!

Вычисление хорошего префикса

Как всегда, обобщаем индуктивное предположение.

```
good-prefix' :
  ∀ (f : ℕ → Word) i ws → Prefix f i ws → Bar ws →
    ∃₂ λ j vs → Prefix f j vs × Good vs
good-prefix' f i ws p (now n) =
  i , ws , p , n
good-prefix' f i ws p (later l) =
  let w = f i
  in good-prefix' f (suc i) (ws # w) (suc p) (l w)
```

Подставляем `ε` в качестве `ws`.

```
good-prefix :
  ∀ (f : ℕ → Word) → ∃₂ λ i ws → (Prefix f i ws × Good ws)
good-prefix f = good-prefix' f zero ε zero bar-ε
```

Выводы

- Конечная синица в руках - лучше бесконечного журавля в небе.
- Не отрицай отрицание, мысли позитивно!
- Не определяйся без необходимости! Сиди на берегу, и жди, пока мимо проплывет...