Лямбда-исчисление

С. Л. Кузнецов

Математический институт им. В.А. Стеклова РАН
sk@mi-ras.ru https://homepage.mi-ras.ru/~sk/

Казанский университет · лекция 1 · 5 ноября 2024 г.

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.
- λ-исчисление — система записи и преобразования λ-термов — является основой парадигмы **функционального программирования.**

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.
- λ-исчисление — система записи и преобразования λ-термов — является основой парадигмы **функционального программирования.**
  - В этой парадигме программа является не набором команд, а сложным функциональным выражением, вычисляемым последовательностью упрощений (редукций).

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.
- λ-исчисление — система записи и преобразования λ-термов — является основой парадигмы **функционального программирования.**
  - В этой парадигме программа является не набором команд, а сложным функциональным выражением, вычисляемым последовательностью упрощений (редукций).
- Мы будем пользоваться маленькими латинскими буквами для записи переменных, а заглавными — для составных выражений (термов).

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.
- λ-исчисление — система записи и преобразования λ-термов — является основой парадигмы **функционального программирования.**
  - В этой парадигме программа является не набором команд, а сложным функциональным выражением, вычисляемым последовательностью упрощений (редукций).
- Мы будем пользоваться маленькими латинскими буквами для записи переменных, а заглавными — для составных выражений (термов).
- Если выражение F задаёт функцию, а M — её аргумент, то выражение FM — это **применение** F к M.

λ-выражения

- λ-выражения, или **λ-термы** используются для абстрактной записи определения и применения функций.
- λ-исчисление — система записи и преобразования λ-термов — является основой парадигмы **функционального программирования.**
  - В этой парадигме программа является не набором команд, а сложным функциональным выражением, вычисляемым последовательностью упрощений (редукций).
- Мы будем пользоваться маленькими латинскими буквами для записи переменных, а заглавными — для составных выражений (термов).
- Если выражение F задаёт функцию, а M — её аргумент, то выражение FM — это **применение** F к M.
  - В математике обычно используется запись $F(M)$, однако для таких функций, как sin или ln, скобки не используются: $\sin x$, $\ln y^2$.

λ-выражения

- Немного более сложной для понимания является **λ-абстракция** $\lambda x.M$, задающая функцию из x в соответствующие значения M.

λ-выражения

- Немного более сложной для понимания является **λ-абстракция** $\lambda x.M$, задающая функцию из x в соответствующие значения M.
- Например, если $M = x^2 + 1$, то $\lambda x.M$ — это функция (квадратичная) $f : \ x \mapsto x^2 + 1$.

λ-выражения

- Немного более сложной для понимания является **λ-абстракция** $\lambda x.M$, задающая функцию из x в соответствующие значения M.
- Например, если $M = x^2 + 1$, то $\lambda x.M$ — это функция (квадратичная) $f : \ x \mapsto x^2 + 1$.
- Обозначение с буквой λ сперва выглядит странным, однако оказывается довольно удобным.

λ-выражения

- Немного более сложной для понимания является **λ-абстракция** $\lambda x.M$, задающая функцию из x в соответствующие значения M.
- Например, если $M = x^2 + 1$, то $\lambda x.M$ — это функция (квадратичная) $f : \ x \mapsto x^2 + 1$.
- Обозначение с буквой λ сперва выглядит странным, однако оказывается довольно удобным.
- Во-первых, задаваемые таким образом функции безымянны, и нам не нужно вводить новую букву (f) для каждой функции.

λ-выражения

- Немного более сложной для понимания является **λ-абстракция** $\lambda x.M$, задающая функцию из x в соответствующие значения M.
- Например, если $M = x^2 + 1$, то $\lambda x.M$ — это функция (квадратичная) $f : \ x \mapsto x^2 + 1$.
- Обозначение с буквой λ сперва выглядит странным, однако оказывается довольно удобным.
- Во-первых, задаваемые таким образом функции безымянны, и нам не нужно вводить новую букву (f) для каждой функции.
- Во-вторых, с помощью лямбд можно разграничивать аргумент от параметров. Например, $\lambda x.(x^2 + px + q)$ задаёт квадратичную функцию с параметрами p и q, а $\lambda p.(x^2 + px + q)$ — линейную, с параметрами x и q.

λ-выражения

- Переменная x может и не входить в выражение M. Тогда $\lambda x.M$ задаёт функцию-константу. (Примеры: $\lambda x.3$ или $\lambda y.(x^2 + px + q)$.)

λ-выражения

- Переменная x может и не входить в выражение M. Тогда $\lambda x.M$ задаёт функцию-константу. (Примеры: $\lambda x.3$ или $\lambda y.(x^2 + px + q)$.)
- **Каррирование** — трюк, позволяющий задавать функции нескольких переменных.

λ-выражения

- Переменная x может и не входить в выражение M. Тогда $\lambda x.M$ задаёт функцию-константу. (Примеры: $\lambda x.3$ или $\lambda y.(x^2 + px + q)$.)
- **Каррирование** — трюк, позволяющий задавать функции нескольких переменных.
- Вместо $F(M, N)$ пишем $(FM)N$, иначе говоря, $(F(M))(N)$.

λ-выражения

- Переменная x может и не входить в выражение M. Тогда $\lambda x.M$ задаёт функцию-константу. (Примеры: $\lambda x.3$ или $\lambda y.(x^2 + px + q)$.)
- **Каррирование** — трюк, позволяющий задавать функции нескольких переменных.
- Вместо $F(M, N)$ пишем $(FM)N$, иначе говоря, $(F(M))(N)$.
- Здесь F принимает свой первый аргумент M и возвращает **функцию**, которая, в свою очередь, принимает второй аргумент N.

λ-выражения

- Переменная x может и не входить в выражение M. Тогда $\lambda x.M$ задаёт функцию-константу. (Примеры: $\lambda x.3$ или $\lambda y.(x^2 + px + q)$.)
- **Каррирование** — трюк, позволяющий задавать функции нескольких переменных.
- Вместо $F(M, N)$ пишем $(FM)N$, иначе говоря, $(F(M))(N)$.
- Здесь F принимает свой первый аргумент M и возвращает **функцию**, которая, в свою очередь, принимает второй аргумент N.
- λ-абстракция для нескольких переменных задаётся последовательностью лямбд: $\lambda x.\lambda y.(x^2 + y^2)$, или коротко $\lambda xy.(x^2 + y^2)$.

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.
- На этой лекции мы рассматриваем λ-исчисление **без типов** — любой терм можно применить к любому другому как функцию.

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.
- На этой лекции мы рассматриваем λ-исчисление **без типов** — любой терм можно применить к любому другому как функцию.
  - Таким образом, например, наряду с выражением $\sin \pi$ появляется также странное “π sin” (т.е. $\pi(\sin)$).

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.
- На этой лекции мы рассматриваем λ-исчисление **без типов** — любой терм можно применить к любому другому как функцию.
  - Таким образом, например, наряду с выражением $\sin \pi$ появляется также странное "π sin" (т.е. $\pi(\sin)$).
- В λ-исчислении с типами (следующая лекция) применение ограничено: чтобы применить F к M, эти термы должны иметь типы, соответственно, вида $\tau \to \sigma$ и τ; тогда (FM) получает тип σ.

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.
- На этой лекции мы рассматриваем λ-исчисление **без типов** — любой терм можно применить к любому другому как функцию.
  - Таким образом, например, наряду с выражением $\sin \pi$ появляется также странное "π sin" (т.е. $\pi(\sin)$).
- В λ-исчислении с типами (следующая лекция) применение ограничено: чтобы применить F к M, эти термы должны иметь типы, соответственно, вида $\tau \to \sigma$ и τ; тогда (FM) получает тип σ.
- Типизованное λ-исчисление имеет более простую семантику и в некотором смысле лучшие свойства.

λ-выражения

- λ-абстрагировать можно и функцию, например, $\lambda f g.\lambda x.f(gx)$ задаёт абстрактный оператор композиции.
- На этой лекции мы рассматриваем λ-исчисление **без типов** — любой терм можно применить к любому другому как функцию.
  - Таким образом, например, наряду с выражением $\sin \pi$ появляется также странное "π sin" (т.е. $\pi(\sin)$).
- В λ-исчислении с типами (следующая лекция) применение ограничено: чтобы применить F к M, эти термы должны иметь типы, соответственно, вида $\tau \to \sigma$ и τ; тогда (FM) получает тип σ.
- Типизованное λ-исчисление имеет более простую семантику и в некотором смысле лучшие свойства.
- Однако вычислительные свойства удобнее начать рассматривать в бестиповом случае («контроль типов мешает программировать»).

«Чистые» λ-термы

- В наших примерах, таких как $\lambda x.(x^2 + p \cdot x + q)$, помимо применения и λ-абстракции, использовались также другие («базовые») операции, например, арифметические действия.

«Чистые» λ-термы

- В наших примерах, таких как $\lambda x.(x^2 + p \cdot x + q)$, помимо применения и λ-абстракции, использовались также другие («базовые») операции, например, арифметические действия.
- В языке «чистого» λ-исчисления никаких других операций нет.

«Чистые» λ-термы

- В наших примерах, таких как $\lambda x.(x^2 + p \cdot x + q)$, помимо применения и λ-абстракции, использовались также другие («базовые») операции, например, арифметические действия.
- В языке «чистого» λ-исчисления никаких других операций нет.
- Тем не менее, такой язык оказывается достаточно выразительным — в нём можно запрограммировать **произвольную вычислимую функцию.**

«Чистые» λ-термы

- В наших примерах, таких как $\lambda x.(x^2 + p \cdot x + q)$, помимо применения и λ-абстракции, использовались также другие («базовые») операции, например, арифметические действия.
- В языке «чистого» λ-исчисления никаких других операций нет.
- Тем не менее, такой язык оказывается достаточно выразительным — в нём можно запрограммировать **произвольную вычислимую функцию.**
  - Аналогия: в теории множеств можно стартовать с языка $(=, \in)$ без базовых объектов, и строить всё из пустого множества: $0 = \varnothing, 1 = \{\varnothing\}, 2 = \{\varnothing, \{\varnothing\}\}, 3 = \{0, 1, 2\}, \ldots$

«Чистые» λ-термы

Пусть $\mathrm{Var} = \{x, y, z, \ldots\}$ — счётное множество **предметных переменных.**

Определение

Множество λ-термов Λ определяется как наименьшее по включению множество со следующими свойствами:

1. $\mathrm{Var} \subseteq \Lambda$;
2. если $M, N \in \Lambda$, то $(MN) \in \Lambda$;
3. если $M \in \Lambda$ и $x \in \mathrm{Var}$, то $\lambda x.M \in \Lambda$.

Связывание переменных

- Операция λ-абстракции **связывает** переменные, так же, как кванторы в логике предикатов.
- Остальные переменные считаются **свободными.**

Связывание переменных

- Операция λ-абстракции **связывает** переменные, так же, как кванторы в логике предикатов.
- Остальные переменные считаются **свободными.**

Определение

Множество $\mathrm{FVar}(M)$ свободных переменных λ-терма M определяется рекурсивно:

1. $\mathrm{FVar}(x) = \{x\}$, если $x \in \mathrm{Var}$;
2. $\mathrm{FVar}((MN)) = \mathrm{FVar}(M) \cup \mathrm{FVar}(N)$;
3. $\mathrm{FVar}(\lambda x.M) = \mathrm{FVar}(M) - \{x\}$.

Связывание переменных

- Операция λ-абстракции **связывает** переменные, так же, как кванторы в логике предикатов.
- Остальные переменные считаются **свободными.**

Определение

Множество $\mathrm{FVar}(M)$ свободных переменных λ-терма M определяется рекурсивно:

1. $\mathrm{FVar}(x) = \{x\}$, если $x \in \mathrm{Var}$;
2. $\mathrm{FVar}((MN)) = \mathrm{FVar}(M) \cup \mathrm{FVar}(N)$;
3. $\mathrm{FVar}(\lambda x.M) = \mathrm{FVar}(M) - \{x\}$.

- Одна и та же переменная может входить в терм и свободным, и связанным образом.

Подстановка

- Вместо свободных переменных можно **подставлять** сложные λ-термы: при подстановке $M[x := K]$ все **свободные** вхождения x в M заменяются на K.

Подстановка

- Вместо свободных переменных можно **подставлять** сложные λ-термы: при подстановке $M[x := K]$ все **свободные** вхождения x в M заменяются на K.
- При этом нужно следить за другими лямбдами: если переменная y, бывшая свободной в K, попадёт под лямбду в M, то такая подстановка некорректна.

Подстановка

- Вместо свободных переменных можно **подставлять** сложные λ-термы: при подстановке $M[x := K]$ все **свободные** вхождения x в M заменяются на K.
- При этом нужно следить за другими лямбдами: если переменная y, бывшая свободной в K, попадёт под лямбду в M, то такая подстановка некорректна.
- Пример некорректной подстановки: $(\lambda y.(x + y))[x := y^2] = \lambda y.(y^2 + y)$.

Подстановка

Определение

Корректная (разрешённая) подстановка $M[x := K]$ определяется рекурсивно:

1. $x[x := K] = K$;
2. $y[x := K] = y$, если $y \in \mathrm{Var} - \{x\}$;
3. $(MN)[x := K] = (M[x := K]\, N[x := K])$;
4. $(\lambda x.M)[x := K] = \lambda x.M$;
5. при $y \neq x$, $(\lambda y.M)[x := K] = \lambda y.M[x := K]$, если $y \notin \mathrm{FVar}(K)$ (иначе подстановка некорректна).

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**
- $\lambda y.M =_\alpha \lambda z.M[y := z]$

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**
- $\lambda y.M =_\alpha \lambda z.M[y := z]$
- α-преобразование всегда корректно, если z — новая переменная.

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**
- $\lambda y.M =_\alpha \lambda z.M[y := z]$
- α-преобразование всегда корректно, если z — новая переменная.
- α-преобразование можно применить к подтерму вида $\lambda y.M$ внутри более сложного терма.

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**
- $\lambda y.M =_\alpha \lambda z.M[y := z]$
- α-преобразование всегда корректно, если z — новая переменная.
- α-преобразование можно применить к подтерму вида $\lambda y.M$ внутри более сложного терма.
- В нашем примере подстановка исправляется так: $(\lambda y.(x + y))[x := y^2] =_\alpha (\lambda z.(x + z))[x := y^2] = \lambda z.(y^2 + z)$.

α-преобразование

- Некорректную подстановку можно исправить переименованием связанной переменной, или **α-преобразованием.**
- $\lambda y.M =_\alpha \lambda z.M[y := z]$
- α-преобразование всегда корректно, если z — новая переменная.
- α-преобразование можно применить к подтерму вида $\lambda y.M$ внутри более сложного терма.
- В нашем примере подстановка исправляется так: $(\lambda y.(x + y))[x := y^2] =_\alpha (\lambda z.(x + z))[x := y^2] = \lambda z.(y^2 + z)$.
- В дальнейшем α-равные термы рассматриваются как совпадающие.

Кодирование натуральных чисел

- Чтобы реализовать вычислимые функции в λ-исчислении, зададим кодирование натуральных чисел так называемыми **нумералами Чёрча.**

Кодирование натуральных чисел

- Чтобы реализовать вычислимые функции в λ-исчислении, зададим кодирование натуральных чисел так называемыми **нумералами Чёрча.**
- Числу n сопоставляется следующий λ-терм:

$$\underline{n} = \lambda f.\lambda x.\, \underbrace{f(f(\ldots(f}_{n \text{ раз}}\, x)\ldots))$$

Кодирование натуральных чисел

- Чтобы реализовать вычислимые функции в λ-исчислении, зададим кодирование натуральных чисел так называемыми **нумералами Чёрча.**
- Числу n сопоставляется следующий λ-терм:

$$\underline{n} = \lambda f.\lambda x.\underbrace{f(f(\ldots(f}_{n \text{ раз}} x)\ldots))$$

  - В частности, $\underline{0} = \lambda f.\lambda x.x$ и $\underline{1} = \lambda f.\lambda x.(f x)$.

Кодирование натуральных чисел

- Чтобы реализовать вычислимые функции в λ-исчислении, зададим кодирование натуральных чисел так называемыми **нумералами Чёрча.**
- Числу n сопоставляется следующий λ-терм:

$$\underline{n} = \lambda f.\lambda x.\, \underbrace{f(f(\ldots(f}_{n \text{ раз}}\, x)\ldots))$$

  - В частности, $\underline{0} = \lambda f.\lambda x.x$ и $\underline{1} = \lambda f.\lambda x.(f x)$.
- Неформально, нумерал Чёрча — это «итератор», преобразующий функцию f в её n-ю итерацию f^n.

Вычисления как редукции

- Нумерал Чёрча удобен для кодирования арифметических операций и рекурсивных функций.

Вычисления как редукции

- Нумерал Чёрча удобен для кодирования арифметических операций и рекурсивных функций.
- Например, операция прибавления единицы задаётся так:

$$S = \lambda n.\lambda f.\lambda x.f((nf)x),$$

а сложения — так:

$$+ = \lambda mn.\lambda f.\lambda x.(mf)((nf)x).$$

Вычисления как редукции

- Нумерал Чёрча удобен для кодирования арифметических операций и рекурсивных функций.
- Например, операция прибавления единицы задаётся так:

$$S = \lambda n.\lambda f.\lambda x.f((nf)x),$$

а сложения — так:

$$+ = \lambda mn.\lambda f.\lambda x.(mf)((nf)x).$$

- Однако пока что не хватает правил, задающих собственно **вычисление,** для приведения, например, $S\underline{6}$ к $\underline{7}$, а $\underline{2} + \underline{2} = (+\,\underline{2})\,\underline{2}$ к $\underline{4}$.

Вычисления как редукции

- Нумерал Чёрча удобен для кодирования арифметических операций и рекурсивных функций.
- Например, операция прибавления единицы задаётся так:

$$S = \lambda n.\lambda f.\lambda x.f((nf)x),$$

а сложения — так:

$$+ = \lambda mn.\lambda f.\lambda x.(mf)((nf)x).$$

- Однако пока что не хватает правил, задающих собственно **вычисление,** для приведения, например, $S\underline{6}$ к $\underline{7}$, а $\underline{2} + \underline{2} = (+\,\underline{2})\,\underline{2}$ к $\underline{4}$.
- Такие правила называются редукциями, и в λ-исчислении основное из них — **β-редукция.**

β-редукция

- β-редукция:

$$(\lambda x.M)N \to_\beta M[x := N].$$

β-редукция

- β-редукция:

$$(\lambda x.M)N \to_\beta M[x := N].$$

  - При этом подстановка должна быть корректна, иначе сначала нужно применить α-преобразование.

β-редукция

- β-редукция:

$$(\lambda x.M)N \to_\beta M[x := N].$$

  - При этом подстановка должна быть корректна, иначе сначала нужно применить α-преобразование.
- β-редукция может быть применена к любому подтерму вида $(\lambda x.M)N$ (называемому **редексом**), заменяя его на **редукт** $M[x := N]$.

β-редукция

- β-редукция:
$$(\lambda x.M)N \to_\beta M[x := N].$$
  - При этом подстановка должна быть корректна, иначе сначала нужно применить α-преобразование.
- β-редукция может быть применена к любому подтерму вида $(\lambda x.M)N$ (называемому **редексом**), заменяя его на **редукт** $M[x := N]$.
- Последовательное применение редукций, обозначаемое $U \twoheadrightarrow_\beta V$, и является процессом вычисления в λ-исчислении: $U = W_1 \to_\beta W_2 \to_\beta \ldots \to_\beta W_k = V$.

β-редукция

- β-редукция:
$$(\lambda x.M)N \to_\beta M[x := N].$$
  - При этом подстановка должна быть корректна, иначе сначала нужно применить α-преобразование.
- β-редукция может быть применена к любому подтерму вида $(\lambda x.M)N$ (называемому **редексом**), заменяя его на **редукт** $M[x := N]$.
- Последовательное применение редукций, обозначаемое $U \twoheadrightarrow_\beta V$, и является процессом вычисления в λ-исчислении: $U = W_1 \to_\beta W_2 \to_\beta \ldots \to_\beta W_k = V$.
- В частности, вычисление функции $f : \mathbb{N} \to \mathbb{N}$, заданной термом F, осуществляется так: $F\,\underline{n} \twoheadrightarrow_\beta \underline{f(n)}$.

Свойства β-редукции

- Такое определение процесса вычислений оставляет два вопроса.

Свойства β-редукции

- Такое определение процесса вычислений оставляет два вопроса.
- Во-первых, внутри терма может быть несколько разных редексов, т.е. процесс редукции **недетерминирован.** В частности, не может ли так получиться, что разные последовательности редукций терма $F\,\underline{n}$ приведут к разным результатам вида $\underline{m}$?

Свойства β-редукции

- Такое определение процесса вычислений оставляет два вопроса.
- Во-первых, внутри терма может быть несколько разных редексов, т.е. процесс редукции **недетерминирован.** В частности, не может ли так получиться, что разные последовательности редукций терма $F\,\underline{n}$ приведут к разным результатам вида $\underline{m}$?
- Во-вторых, возможна ли **бесконечная** последовательность редукций (бесконечное вычисление)?

Свойства β-редукции

- Такое определение процесса вычислений оставляет два вопроса.
- Во-первых, внутри терма может быть несколько разных редексов, т.е. процесс редукции **недетерминирован.** В частности, не может ли так получиться, что разные последовательности редукций терма $F\,\underline{n}$ приведут к разным результатам вида $\underline{m}$?
- Во-вторых, возможна ли **бесконечная** последовательность редукций (бесконечное вычисление)?
  - Здесь термин «редукция» («упрощение») вводит в заблуждение: на самом деле, терм $M[x := N]$ зачастую намного длиннее исходного терма $(\lambda x.M)N$.

Свойства β-редукции

Определение

Терм называется **нормальной формой** ($V \in \mathrm{NF}$) если в нём нет редексов (т.е. невозможно применить β-редукцию).

Определение

Приведение терма U к нормальной форме: $U \twoheadrightarrow_\beta V \in \mathrm{NF}$.

Свойства β-редукции

Определение

Терм называется **нормальной формой** ($V \in \mathrm{NF}$) если в нём нет редексов (т.е. невозможно применить β-редукцию).

Определение

Приведение терма U к нормальной форме: $U \twoheadrightarrow_\beta V \in \mathrm{NF}$.

— Можно ли терм привести к двум α-разным нормальным формам? — **Нет.**

Свойства β-редукции

Определение

Терм называется **нормальной формой** ($V \in \mathrm{NF}$) если в нём нет редексов (т.е. невозможно применить β-редукцию).

Определение

Приведение терма U к нормальной форме: $U \twoheadrightarrow_\beta V \in \mathrm{NF}$.

— Можно ли терм привести к двум α-разным нормальным формам? — **Нет.**

— Существуют ли термы, не приводимые к нормальной форме? — **Да.**

Пример: терм $\Omega = (\lambda x.(xx))(\lambda x.(xx))$, для которого $\Omega \to_\beta \Omega$.

Свойство Чёрча – Россера

Единственность нормальной формы следует из **конфлюэнтности,** или **свойства Чёрча – Россера (CR):**

Теорема

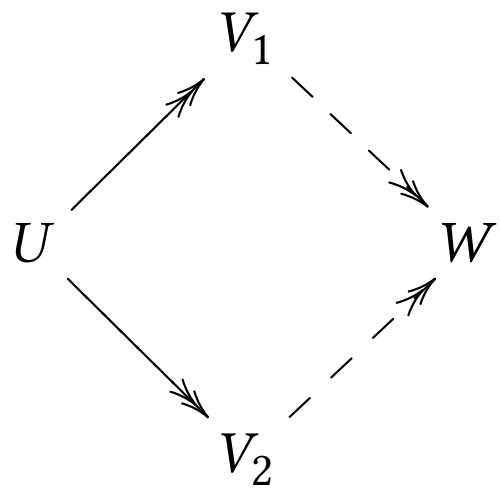
Если $U \twoheadrightarrow_\beta V_1$ *и* $U \twoheadrightarrow_\beta V_2$, *то существует такой* W, *что* $V_1 \twoheadrightarrow_\beta W$ *и* $V_2 \twoheadrightarrow_\beta W$.

Свойство Чёрча – Россера

Единственность нормальной формы следует из **конфлюэнтности,** или **свойства Чёрча – Россера (CR):**

Теорема

Если $U \twoheadrightarrow_\beta V_1$ *и* $U \twoheadrightarrow_\beta V_2$, *то существует такой* W, *что* $V_1 \twoheadrightarrow_\beta W$ *и* $V_2 \twoheadrightarrow_\beta W$.

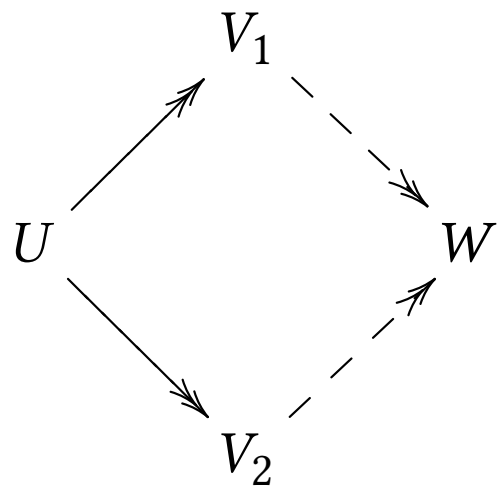


Свойство Чёрча – Россера

Единственность нормальной формы следует из **конфлюэнтности,** или **свойства Чёрча – Россера (CR):**

Теорема

Если $U \twoheadrightarrow_\beta V_1$ *и* $U \twoheadrightarrow_\beta V_2$, *то существует такой* W, *что* $V_1 \twoheadrightarrow_\beta W$ *и* $V_2 \twoheadrightarrow_\beta W$.



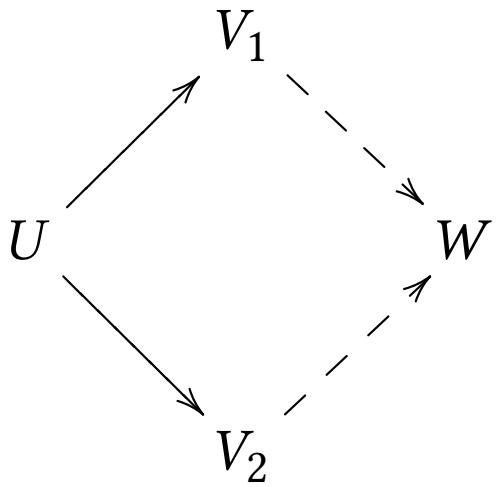
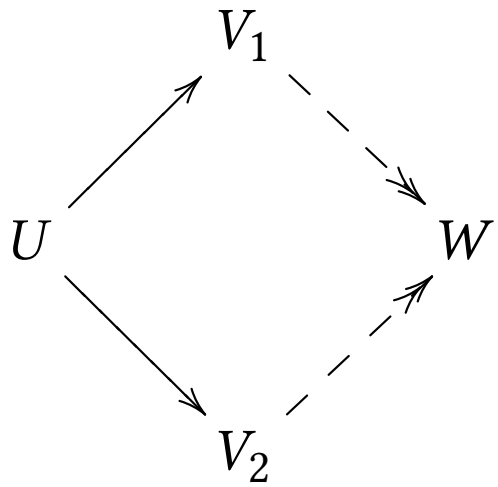
Действительно, если $V_1, V_2 \in \mathrm{NF}$, то имеем $V_1 =_\alpha W =_\alpha V_2$ (поскольку дальнейшие β-редукции невозможны).

Свойство Чёрча – Россера

Наивные попытки доказать свойство CR индукцией по длине последовательностей редукций не срабатывают.

Свойство Чёрча – Россера

Наивные попытки доказать свойство CR индукцией по длине последовательностей редукций не срабатывают.

свойство ромба (D)	слабое свойство Чёрча – Россера (WCR)
	
неверно	**недостаточно для CR**

«Пакетные» редукции

Определим новое отношение $\to_\ell$ следующим рекурсивным образом:

1. $M \to_\ell M$;
2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

«Пакетные» редукции

Определим новое отношение $\to_\ell$ следующим рекурсивным образом:

1. $M \to_\ell M$;
2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

Существенное отличие здесь в пп. 2 и 3: на одном шаге $\to_\ell$ β-редукции могут осуществляться одновременно в M_1 и N_1 («пакет редукций»).

«Пакетные» редукции

Определим новое отношение $\to_\ell$ следующим рекурсивным образом:

1. $M \to_\ell M$;
2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

Существенное отличие здесь в пп. 2 и 3: на одном шаге $\to_\ell$ β-редукции могут осуществляться одновременно в M_1 и N_1 («пакет редукций»).

$$\to_\beta \subseteq \to_\ell \subseteq \twoheadrightarrow_\beta$$

«Пакетные» редукции

Определим новое отношение $\to_\ell$ следующим рекурсивным образом:

1. $M \to_\ell M$;
2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

Существенное отличие здесь в пп. 2 и 3: на одном шаге $\to_\ell$ β-редукции могут осуществляться одновременно в M_1 и N_1 («пакет редукций»).

$$\to_\beta \subseteq \to_\ell \subseteq \twoheadrightarrow_\beta$$

$$U \to_\beta V \Rightarrow U \to_\ell V \Rightarrow U \twoheadrightarrow_\beta V$$

«Пакетные» редукции

Определим новое отношение $\to_\ell$ следующим рекурсивным образом:

1. $M \to_\ell M$;
2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

Существенное отличие здесь в пп. 2 и 3: на одном шаге $\to_\ell$ β-редукции могут осуществляться одновременно в M_1 и N_1 («пакет редукций»).

$$\to_\beta \subseteq \to_\ell \subseteq \twoheadrightarrow_\beta$$

$$U \to_\beta V \Rightarrow U \to_\ell V \Rightarrow U \twoheadrightarrow_\beta V$$

$$U \twoheadrightarrow_\beta V \iff U \twoheadrightarrow_\ell V$$

«Пакетная» редукция

- В отличие от $\to_\beta$, отношение $\to_\ell$ обладает свойством D: если $U \to_\ell V_1$ и $U \to_\ell V_2$, то найдётся такой W, что $V_1 \to_\ell W$ и $V_2 \to_\ell W$.

«Пакетная» редукция

- В отличие от $\to_\beta$, отношение $\to_\ell$ обладает свойством D: если $U \to_\ell V_1$ и $U \to_\ell V_2$, то найдётся такой W, что $V_1 \to_\ell W$ и $V_2 \to_\ell W$.
- Доказательство — разбор случаев с использованием леммы о подстановке: если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $M_1[x := N_1] \to_\ell M_2[x := N_2]$.

«Пакетная» редукция

- В отличие от $\to_\beta$, отношение $\to_\ell$ обладает свойством D: если $U \to_\ell V_1$ и $U \to_\ell V_2$, то найдётся такой W, что $V_1 \to_\ell W$ и $V_2 \to_\ell W$.
- Доказательство — разбор случаев с использованием леммы о подстановке: если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $M_1[x := N_1] \to_\ell M_2[x := N_2]$.
- Теперь «достроением прямоугольника» доказывается CR для $\to_\ell$, а значит и для $\to_\beta$.

«Пакетная» редукция

- В отличие от $\to_\beta$, отношение $\to_\ell$ обладает свойством D: если $U \to_\ell V_1$ и $U \to_\ell V_2$, то найдётся такой W, что $V_1 \to_\ell W$ и $V_2 \to_\ell W$.
- Доказательство — разбор случаев с использованием леммы о подстановке: если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $M_1[x := N_1] \to_\ell M_2[x := N_2]$.
- Теперь «достроением прямоугольника» доказывается CR для $\to_\ell$, а значит и для $\to_\beta$.
- Свойство Чёрча – Россера доказано.

Сильно и слабо нормализуемые термы

Определение

Терм U **слабо нормализуем** ($U \in \mathrm{WN}$), если $U \twoheadrightarrow_\beta V$ для некоторого $V \in \mathrm{NF}$.

Определение

Терм U **сильно нормализуем** ($U \in \mathrm{SN}$), если **любая** последовательность редукций из U заканчивается (за конечное число шагов) нормальной формой.

Сильно и слабо нормализуемые термы

Определение

Терм U **слабо нормализуем** ($U \in \mathrm{WN}$), если $U \twoheadrightarrow_\beta V$ для некоторого $V \in \mathrm{NF}$.

Определение

Терм U **сильно нормализуем** ($U \in \mathrm{SN}$), если **любая** последовательность редукций из U заканчивается (за конечное число шагов) нормальной формой.

- Ясно, что $\mathrm{SN} \subseteq \mathrm{WN}$.

Сильно и слабо нормализуемые термы

Определение

Терм U **слабо нормализуем** ($U \in \mathrm{WN}$), если $U \twoheadrightarrow_\beta V$ для некоторого $V \in \mathrm{NF}$.

Определение

Терм U **сильно нормализуем** ($U \in \mathrm{SN}$), если **любая** последовательность редукций из U заканчивается (за конечное число шагов) нормальной формой.

- Ясно, что $\mathrm{SN} \subseteq \mathrm{WN}$.
- Примеры:

$\lambda x.x \in \mathrm{NF}$; $((\lambda x.x)\, y) \in \mathrm{SN}$; $\Omega \notin \mathrm{WN}$; $((\lambda x.y)\, \Omega) \in \mathrm{WN} - \mathrm{SN}$.

Представимость вычислимых функций

Определение

Терм F представляет частичную функцию $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$, если для любого набора $n_1, \ldots, n_k$ имеем:

1. $F\,\underline{n_1}\ldots\underline{n_k} \twoheadrightarrow_\beta \underline{f(n_1, \ldots, n_k)}$, если $f(n_1, \ldots, n_k)$ определено;
2. $F\,\underline{n_1}\ldots\underline{n_k} \notin \mathrm{WN}$ иначе.

Представимость вычислимых функций

Определение

Терм F представляет частичную функцию $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$, если для любого набора $n_1, \ldots, n_k$ имеем:

1. $F\,\underline{n_1}\ldots\underline{n_k} \twoheadrightarrow_\beta \underline{f(n_1, \ldots, n_k)}$, если $f(n_1, \ldots, n_k)$ определено;
2. $F\,\underline{n_1}\ldots\underline{n_k} \notin \mathrm{WN}$ иначе.

Отметим, что в п. 1 нет требования $F\,\underline{n_1}\ldots\underline{n_k} \in \mathrm{SN}$, таким образом, для вычисления нужно выбрать правильную **стратегию** редукций. (Об этом поговорим чуть позже.)

Примитивно-рекурсивные функции

- Для начала научимся кодировать более простой подкласс вычислимых функций — **примитивно-рекурсивные функции** (ПРФ).

Примитивно-рекурсивные функции

- Для начала научимся кодировать более простой подкласс вычислимых функций — **примитивно-рекурсивные функции** (ПРФ).
- Примитивная рекурсия разрешает, при вычислении $f(n)$ ($n > 0$), использовать предыдущее значение $f(n-1)$:

$$\begin{cases} f(0, \vec{m}) = g(\vec{m}) \\ f(n+1, \vec{m}) = h(f(n, \vec{m}), n, \vec{m}) \end{cases}$$

Примитивно-рекурсивные функции

- Для начала научимся кодировать более простой подкласс вычислимых функций — **примитивно-рекурсивные функции** (ПРФ).
- Примитивная рекурсия разрешает, при вычислении $f(n)$ ($n > 0$), использовать предыдущее значение $f(n-1)$:

$$\begin{cases} f(0, \vec{m}) = g(\vec{m}) \\ f(n+1, \vec{m}) = h(f(n, \vec{m}), n, \vec{m}) \end{cases}$$

- Кроме того, класс ПРФ содержит базовые функции (Z — константа 0, $S(n) = n + 1$, $I_k^n(x_1, \ldots, x_n) = x_k$) и замкнут относительно композиции.

Примитивно-рекурсивные функции

- Для начала научимся кодировать более простой подкласс вычислимых функций — **примитивно-рекурсивные функции** (ПРФ).
- Примитивная рекурсия разрешает, при вычислении $f(n)$ ($n > 0$), использовать предыдущее значение $f(n-1)$:

$$\begin{cases} f(0, \vec{m}) = g(\vec{m}) \\ f(n+1, \vec{m}) = h(f(n, \vec{m}), n, \vec{m}) \end{cases}$$

- Кроме того, класс ПРФ содержит базовые функции (Z — константа 0, $S(n) = n + 1$, $I_k^n(x_1, \ldots, x_n) = x_k$) и замкнут относительно композиции.
- Мы уже проверили, что базовые функции и композиция представимы в λ-исчислении.

Примитивно-рекурсивные функции

- Для кодирования примитивной рекурсии будем с помощью итерирования задавать **график** функции f — последовательно строить **пары** $\langle n, f(n, \vec{m})\rangle$.

Примитивно-рекурсивные функции

- Для кодирования примитивной рекурсии будем с помощью итерирования задавать **график** функции f — последовательно строить **пары** $\langle n, f(n, \vec{m})\rangle$.
- Для простоты начнём с функции без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

Примитивно-рекурсивные функции

- Для кодирования примитивной рекурсии будем с помощью итерирования задавать **график** функции f — последовательно строить **пары** $\langle n, f(n, \vec{m})\rangle$.
- Для простоты начнём с функции без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

- Пример — факториал $f(n) = n! = 1 \cdot \ldots \cdot n$.

$$\begin{cases} f(0) = 1 \\ f(n+1) = f(n) \cdot (n+1) \end{cases}$$

Примитивно-рекурсивные функции

- Для кодирования примитивной рекурсии будем с помощью итерирования задавать **график** функции f — последовательно строить **пары** $\langle n, f(n, \vec{m})\rangle$.
- Для простоты начнём с функции без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

- Пример — факториал $f(n) = n! = 1 \cdot \ldots \cdot n$.

$$\begin{cases} f(0) = 1 \\ f(n+1) = f(n) \cdot (n+1) \end{cases}$$

- Нам понадобится **кодирование пар.**

Кодирование пар

- Пара $\langle M, N \rangle$ кодируется (по Чёрчу) следующим образом: $\lambda z.((zM)N)$.

Кодирование пар

- Пара $\langle M, N \rangle$ кодируется (по Чёрчу) следующим образом: $\lambda z.((zM)N)$.
- Такое кодирование позволяет извлекать элементы пары с помощью **проекторов:**

$$P_1 = \lambda w.(w(\lambda xy.x)) \qquad P_1 \langle M, N \rangle \twoheadrightarrow_\beta M$$
$$P_2 = \lambda w.(w(\lambda xy.y)) \qquad P_2 \langle M, N \rangle \twoheadrightarrow_\beta N$$

Кодирование пар

- Пара $\langle M, N\rangle$ кодируется (по Чёрчу) следующим образом: $\lambda z.((zM)N)$.
- Такое кодирование позволяет извлекать элементы пары с помощью **проекторов:**

$$P_1 = \lambda w.(w(\lambda xy.x)) \qquad P_1\,\langle M, N\rangle \twoheadrightarrow_\beta M$$
$$P_2 = \lambda w.(w(\lambda xy.y)) \qquad P_2\,\langle M, N\rangle \twoheadrightarrow_\beta N$$

- Сама операция взятия пары также может быть абстрагирована: $\langle\cdot,\cdot\rangle = \lambda xy.\langle x, y\rangle = \lambda xyz.((zx)y)$, так что $(\langle\cdot,\cdot\rangle M)N \twoheadrightarrow_\beta \langle M, N\rangle$.

Примитивная рекурсия

- Пусть функция f задана примитивной рекурсией без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

Примитивная рекурсия

- Пусть функция f задана примитивной рекурсией без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

- Здесь g — константа, а h — ранее заданная функция, т.о., уже найдены соответствующие термы G и H.

Примитивная рекурсия

- Пусть функция f задана примитивной рекурсией без параметров:
$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$
- Здесь g — константа, а h — ранее заданная функция, т.о., уже найдены соответствующие термы G и H.
- Переход от пары $\langle n, f(n) \rangle$ к паре $\langle n+1, f(n+1) \rangle$ задаётся всегда одной и той же функцией $\langle n, m \rangle \mapsto \langle n+1, h(m, n) \rangle$.

Примитивная рекурсия

- Пусть функция f задана примитивной рекурсией без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

- Здесь g — константа, а h — ранее заданная функция, т.о., уже найдены соответствующие термы G и H.
- Переход от пары $\langle n, f(n)\rangle$ к паре $\langle n+1, f(n+1)\rangle$ задаётся всегда одной и той же функцией $\langle n, m\rangle \mapsto \langle n+1, h(m,n)\rangle$.
- Эту **функцию перехода** мы можем задать λ-термом:

$$\mathrm{Step} = \lambda w.\langle S(P_1 w), (H(P_2 w))(P_1 w)\rangle.$$

Примитивная рекурсия

- Пусть функция f задана примитивной рекурсией без параметров:

$$\begin{cases} f(0) = g \\ f(n+1) = h(f(n), n) \end{cases}$$

- Здесь g — константа, а h — ранее заданная функция, т.о., уже найдены соответствующие термы G и H.
- Переход от пары $\langle n, f(n)\rangle$ к паре $\langle n+1, f(n+1)\rangle$ задаётся всегда одной и той же функцией $\langle n, m\rangle \mapsto \langle n+1, h(m,n)\rangle$.
- Эту **функцию перехода** мы можем задать λ-термом:

$$\mathrm{Step} = \lambda w.\langle S(P_1 w), (H(P_2 w))(P_1 w)\rangle.$$

  - Здесь $S = \lambda n.\lambda f.\lambda x.f((nf)x)$ задаёт операцию «+1».

Примитивная рекурсия

- Функция перехода итерируется аргументом n:

$$\text{Step} = \lambda w.\langle S(P_1 w), (H(P_2 w))(P_1 w)\rangle$$

$$F = \lambda n.P_2((n\ \text{Step})\, G_0),\ \text{где}\ G_0 = \langle \underline{0}, G\rangle$$

Примитивная рекурсия

- Функция перехода итерируется аргументом n:

$$\text{Step} = \lambda w.\langle S(P_1 w), (H(P_2 w))(P_1 w)\rangle$$
$$F = \lambda n.P_2((n\ \text{Step})\, G_0),\ \text{где}\ G_0 = \langle \underline{0}, G\rangle$$

- Например, для факториала имеем

$$\text{Step} = \lambda w.\langle S(P_1 w), (P_2 w) \cdot (S(P_1 w))\rangle \text{ и } G_0 = \langle \underline{0}, \underline{1}\rangle$$

(здесь $U \cdot V$ означает $(\times U)\, V$, где $\times$ — ранее рекурсивно определённая функция умножения).

Примитивная рекурсия

- Функция перехода итерируется аргументом n:

$$\text{Step} = \lambda w.\langle S(P_1 w), (H(P_2 w))(P_1 w)\rangle$$
$$F = \lambda n.P_2((n\ \text{Step})\, G_0),\ \text{где}\ G_0 = \langle \underline{0}, G\rangle$$

- Например, для факториала имеем

$$\text{Step} = \lambda w.\langle S(P_1 w), (P_2 w)\cdot(S(P_1 w))\rangle \text{ и } G_0 = \langle \underline{0}, \underline{1}\rangle$$

(здесь $U \cdot V$ означает $(\times U)\, V$, где $\times$ — ранее рекурсивно определённая функция умножения).

- Для вычисления $f(3) = 3!$ строится следующая последовательность редукций:

$$F\,\underline{3} = \big(\lambda n.P_2((n\ \text{Step})G_0)\big)\,\underline{3} \to_\beta P_2((\underline{3}\ \text{Step})G_0) \twoheadrightarrow_\beta$$
$$\twoheadrightarrow_\beta P_2(\text{Step}(\text{Step}(\text{Step}\ \langle \underline{0}, \underline{1}\rangle))) \twoheadrightarrow_\beta P_2(\text{Step}(\text{Step}\ \langle \underline{1}, \underline{1}\rangle)) \twoheadrightarrow_\beta$$
$$\twoheadrightarrow_\beta P_2(\text{Step}\ \langle \underline{2}, \underline{2}\rangle) \twoheadrightarrow_\beta P_2\ \langle \underline{3}, \underline{6}\rangle \twoheadrightarrow_\beta \underline{6}.$$

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.
- Для простоты пусть параметр один:

$$\begin{cases} f(0, m) = g(m) \\ f(n+1, m) = h(f(n, m), n, m) \end{cases}$$

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.
- Для простоты пусть параметр один:

$$\begin{cases} f(0, m) = g(m) \\ f(n+1, m) = h(f(n, m), n, m) \end{cases}$$

- Для начала извлечём параметр как свободную переменную m: $\quad G' = Gm; \quad H' = \lambda xy.(((Hx)y)m).$

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.
- Для простоты пусть параметр один:

$$\begin{cases} f(0, m) = g(m) \\ f(n+1, m) = h(f(n, m), n, m) \end{cases}$$

- Для начала извлечём параметр как свободную переменную m: $G' = Gm$; $H' = \lambda xy.(((Hx)y)m)$.
  - Термы G и H были замкнутые, т.е. не содержали свободных переменных, а в G' и H' параметры стали свободными переменными.

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.
- Для простоты пусть параметр один:

$$\begin{cases} f(0, m) = g(m) \\ f(n+1, m) = h(f(n, m), n, m) \end{cases}$$

- Для начала извлечём параметр как свободную переменную m: $\quad G' = Gm; \quad H' = \lambda xy.(((Hx)y)m).$
  - Термы G и H были замкнутые, т.е. не содержали свободных переменных, а в G' и H' параметры стали свободными переменными.
- Теперь строим термы Step и F', со свободной m:

$$\text{Step} = \lambda w.\langle S(P_1 w), (H'(P_2 w))(P_1 w)\rangle; \qquad F' = \lambda n.P_1((n\,\text{Step})\,G'_0)$$

Примитивная рекурсия

- Примитивная рекурсия с параметрами кодируется схожим образом.
- Для простоты пусть параметр один:

$$\begin{cases} f(0, m) = g(m) \\ f(n+1, m) = h(f(n, m), n, m) \end{cases}$$

- Для начала извлечём параметр как свободную переменную m: $\quad G' = Gm; \quad H' = \lambda xy.(((Hx)y)m)$.
  - Термы G и H были замкнутые, т.е. не содержали свободных переменных, а в G' и H' параметры стали свободными переменными.
- Теперь строим термы Step и F', со свободной m:

$$\text{Step} = \lambda w.\langle S(P_1 w), (H'(P_2 w))(P_1 w)\rangle; \qquad F' = \lambda n.P_1((n\,\text{Step})\,G'_0)$$

- Наконец, абстрагируем m: $F = \lambda nm.(F'n)$.

О вычислении ПРФ

- Термы $F\,\underline{n}$ на самом деле оказываются сильно нормализуемыми, т.е. выбор последовательности применения редукций не имеет значения.

О вычислении ПРФ

- Термы $F\,\underline{n}$ на самом деле оказываются сильно нормализуемыми, т.е. выбор последовательности применения редукций не имеет значения.
- Сами нумералы являются нормальными формами, поэтому при достижении $\underline{f(n)}$ вычисление завершается.

О вычислении ПРФ

- Термы $F\,\underline{n}$ на самом деле оказываются сильно нормализуемыми, т.е. выбор последовательности применения редукций не имеет значения.
- Сами нумералы являются нормальными формами, поэтому при достижении $\underline{f(n)}$ вычисление завершается.
- С другой стороны, мы работаем в бестиповом λ-исчислении, поэтому контроль правильности применения не осуществляется: если применять F не к нумералу(-ам), то ничего не гарантируется.

О вычислении ПРФ

- Термы $F\,\underline{n}$ на самом деле оказываются сильно нормализуемыми, т.е. выбор последовательности применения редукций не имеет значения.
- Сами нумералы являются нормальными формами, поэтому при достижении $\underline{f(n)}$ вычисление завершается.
- С другой стороны, мы работаем в бестиповом λ-исчислении, поэтому контроль правильности применения не осуществляется: если применять F не к нумералу(-ам), то ничего не гарантируется.
- Класс ПРФ не совпадает с классом всех всюду определённых вычислимых функций, однако достаточно обширен — например, содержит все функции, вычислимые за полиномиальное время (и вообще за время, ограниченное некоторой ПРФ).

Логические операции

- Помимо натуральных чисел, кодирование в стиле Чёрча доступно и для других дискретных сущностей.

Логические операции

- Помимо натуральных чисел, кодирование в стиле Чёрча доступно и для других дискретных сущностей.
- Например, логические (булевы) константы «истина» и «ложь» кодируются следующим образом:

$$T = \lambda xy.x, \qquad F = \lambda xy.y,$$

что позволяет легко задавать условный оператор:

$$\text{if } Q \text{ then } M \text{ else } N = QMN,$$

так что if T then M else $N \twoheadrightarrow_\beta M$ и if F then M else $N \twoheadrightarrow_\beta N$.

Логические операции

- Помимо натуральных чисел, кодирование в стиле Чёрча доступно и для других дискретных сущностей.
- Например, логические (булевы) константы «истина» и «ложь» кодируются следующим образом:

$$T = \lambda xy.x, \qquad F = \lambda xy.y,$$

что позволяет легко задавать условный оператор:

$$\text{if } Q \text{ then } M \text{ else } N = QMN,$$

так что if T then M else $N \twoheadrightarrow_\beta M$ и if F then M else $N \twoheadrightarrow_\beta N$.

- Через условный оператор задаются булевы операции, например: and $= \lambda xy.(\text{if } x \text{ then } y \text{ else } F)$.

Функциональное программирование

- Таким образом, на базе чистого λ-исчисления можно выстроить функциональный язык программирования.

Функциональное программирование

- Таким образом, на базе чистого λ-исчисления можно выстроить функциональный язык программирования.
- В реальных языках, для достижения эффективности, используются машинные примитивы для арифметических, логических операций и проч.

Функция Аккермана

- Функция Аккермана – Петер является классическим примером вычислимой, всюду определённой и при этом не примитивно-рекурсивной функцией.

Функция Аккермана

- Функция Аккермана – Петер является классическим примером вычислимой, всюду определённой и при этом не примитивно-рекурсивной функцией.
- Сначала определяется функция двух аргументов:

$$\begin{cases} A(0, m) = m + 1 \\ A(n + 1, 0) = A(n, 1) \\ A(n + 1, m + 1) = A(n, A(n + 1, m)) \end{cases}$$

Функция Аккермана

- Функция Аккермана – Петер является классическим примером вычислимой, всюду определённой и при этом не примитивно-рекурсивной функцией.
- Сначала определяется функция двух аргументов:

$$\begin{cases} A(0, m) = m + 1 \\ A(n + 1, 0) = A(n, 1) \\ A(n + 1, m + 1) = A(n, A(n + 1, m)) \end{cases}$$

- Диагонализованная функция $A_d(n) = A(n, n)$ растёт быстрее любой ПРФ, и потому сама не является ПРФ.

Функция Аккермана

- Функция Аккермана – Петер является классическим примером вычислимой, всюду определённой и при этом не примитивно-рекурсивной функцией.
- Сначала определяется функция двух аргументов:

$$\begin{cases} A(0, m) = m + 1 \\ A(n + 1, 0) = A(n, 1) \\ A(n + 1, m + 1) = A(n, A(n + 1, m)) \end{cases}$$

- Диагонализованная функция $A_d(n) = A(n, n)$ растёт быстрее любой ПРФ, и потому сама не является ПРФ.
- В λ-исчислении, однако, функция A также представима, потому что можно каждый её «слой» $A_n = \lambda x.A(n, x)$, рекурсивно используя предыдущий «слой» A_{n-1} целиком.

Произвольные вычислимые функции

- Для кодирования произвольной вычислимой функции воспользуемся **теоремой Клини о нормальной форме.**

Произвольные вычислимые функции

- Для кодирования произвольной вычислимой функции воспользуемся **теоремой Клини о нормальной форме.**

Теорема

Всякая вычислимая частичная функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ *может быть представлена в виде*

$$f(n) = q(\min\{k \mid g(k,n) = 0\})$$

для некоторых ПРФ g *и* q*. При этом если* $\forall k\ g(k,n) \neq 0$*, то* $f(n)$ *не определено.*

Произвольные вычислимые функции

- Минимизация $f(n) = q(\min\{k \mid g(k,n) = 0\})$ задаётся следующей рекурсивной программой:

$$f(n) = q(h(0,n))$$
$$h(m,n) = \big(\text{if } g(m,n) = 0 \text{ then } m \text{ else } h(m+1,n)\big)$$

Произвольные вычислимые функции

- Минимизация $f(n) = q(\min\{k \mid g(k,n) = 0\})$ задаётся следующей рекурсивной программой:

$$f(n) = q(h(0,n))$$
$$h(m,n) = \big(\text{if } g(m,n) = 0 \text{ then } m \text{ else } h(m+1,n)\big)$$

  - Здесь $h(m,n) = \min\{k \geqslant m \mid g(k,n) = 0\}$.

Произвольные вычислимые функции

- Минимизация $f(n) = q(\min\{k \mid g(k,n) = 0\})$ задаётся следующей рекурсивной программой:

$$f(n) = q(h(0,n))$$
$$h(m,n) = \big(\text{if } g(m,n) = 0 \text{ then } m \text{ else } h(m+1,n)\big)$$

  - Здесь $h(m,n) = \min\{k \geqslant m \mid g(k,n) = 0\}$.
- Рекурсия здесь, однако, не примитивная — определение $h(m,n)$ ссылается на **следующее** значение $h(m,n+1)$.

Произвольные вычислимые функции

- Минимизация $f(n) = q(\min\{k \mid g(k,n) = 0\})$ задаётся следующей рекурсивной программой:

$$f(n) = q(h(0,n))$$
$$h(m,n) = \big(\text{if } g(m,n) = 0 \text{ then } m \text{ else } h(m+1,n)\big)$$

  - Здесь $h(m,n) = \min\{k \geqslant m \mid g(k,n) = 0\}$.
- Рекурсия здесь, однако, не примитивная — определение $h(m,n)$ ссылается на **следующее** значение $h(m,n+1)$.
- Такая рекурсия может не остановиться, поэтому кодирующие её термы не могут быть a priori сильно нормализуемы. В частности, её нельзя задать с помощью итератора $\underline{n}$.

Комбинатор неподвижной точки

- Произвольная рекурсия реализуется с помощью конструкции **неподвижной точки.**

Комбинатор неподвижной точки

- Произвольная рекурсия реализуется с помощью конструкции **неподвижной точки.**
- Неподвижной точкой терма F называется такой терм M, что $M =_\beta FM$.

Комбинатор неподвижной точки

- Произвольная рекурсия реализуется с помощью конструкции **неподвижной точки.**
- Неподвижной точкой терма F называется такой терм M, что $M =_\beta FM$.
- Один из способов построить неподвижную точку: $M = (\lambda x.F(xx))(\lambda x.F(xx))$.

Комбинатор неподвижной точки

- Произвольная рекурсия реализуется с помощью конструкции **неподвижной точки.**
- Неподвижной точкой терма F называется такой терм M, что $M =_\beta FM$.
- Один из способов построить неподвижную точку: $M = (\lambda x.F(xx))(\lambda x.F(xx))$.
- Исходный терм F можно абстрагировать, получив **Y-комбинатор** (комбинатор неподвижной точки): $\mathbb{Y} = \lambda f.\big((\lambda x.f(xx))(\lambda x.f(xx))\big)$.

Комбинатор неподвижной точки

- Произвольная рекурсия реализуется с помощью конструкции **неподвижной точки.**
- Неподвижной точкой терма F называется такой терм M, что $M =_\beta FM$.
- Один из способов построить неподвижную точку: $M = (\lambda x.F(xx))(\lambda x.F(xx))$.
- Исходный терм F можно абстрагировать, получив **Y-комбинатор** (комбинатор неподвижной точки): $\mathbb{Y} = \lambda f.\big((\lambda x.f(xx))(\lambda x.f(xx))\big)$.
- Отметим, что конструкция неподвижной точки принципиально не является сильно нормализуемой: $M \to_\beta FM \to_\beta F(FM) \to_\beta \ldots$, поэтому важной становится стратегия редукции.

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

- Положим $F = \lambda h.\lambda mn.\big((Gmn)(\lambda t.(h(Sm)n))\, m\big)$ и $H = (\lambda x.F(xx))(\lambda x.F(xx))$. Это неподвижная точка: $H \to_\beta FH \to_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big)$.

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

- Положим $F = \lambda h.\lambda mn.\big((Gmn)(\lambda t.(h(Sm)n))\, m\big)$ и $H = (\lambda x.F(xx))(\lambda x.F(xx))$. Это неподвижная точка: $H \to_\beta FH \to_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big)$.
- Далее, исходная функция задаётся как $\lambda x.Q(H\,\underline{0}\,x)$.

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

- Положим $F = \lambda h.\lambda mn.\big((Gmn)(\lambda t.(h(Sm)n))\, m\big)$ и $H = (\lambda x.F(xx))(\lambda x.F(xx))$. Это неподвижная точка: $H \to_\beta FH \to_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big)$.
- Далее, исходная функция задаётся как $\lambda x.Q(H\,\underline{0}\,x)$.
- При вычислении H на конкретных значениях $\underline{m}$ и $\underline{n}$ нужно не раскрывать H внутри терма, а сначала применить внешние лямбды (**«ленивое вычисление»**).

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

- Положим $F = \lambda h.\lambda mn.\big((Gmn)(\lambda t.(h(Sm)n))\, m\big)$ и $H = (\lambda x.F(xx))(\lambda x.F(xx))$. Это неподвижная точка: $H \to_\beta FH \to_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big)$.
- Далее, исходная функция задаётся как $\lambda x.Q(H\,\underline{0}\,x)$.
- При вычислении H на конкретных значениях $\underline{m}$ и $\underline{n}$ нужно не раскрывать H внутри терма, а сначала применить внешние лямбды (**«ленивое вычисление»**).
- Тогда будет вычислено $G\,\underline{m}\,\underline{n}$, и если оно равно нулю, то возвращено значение $\underline{m}$.

Рекурсия как неподвижная точка

- Мы хотим найти **решение уравнения**

$$H =_\beta \lambda mn.\big(\text{if } Gmn = 0 \text{ then } m \text{ else } H(Sm)n\big),$$

т.е.

$$H =_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big).$$

- Положим $F = \lambda h.\lambda mn.\big((Gmn)(\lambda t.(h(Sm)n))\, m\big)$ и $H = (\lambda x.F(xx))(\lambda x.F(xx))$. Это неподвижная точка: $H \to_\beta FH \to_\beta \lambda mn.\big((Gmn)(\lambda t.(H(Sm)n))\, m\big)$.
- Далее, исходная функция задаётся как $\lambda x.Q(H\,\underline{0}\,x)$.
- При вычислении H на конкретных значениях $\underline{m}$ и $\underline{n}$ нужно не раскрывать H внутри терма, а сначала применить внешние лямбды (**«ленивое вычисление»**).
- Тогда будет вычислено $G\,\underline{m}\,\underline{n}$, и если оно равно нулю, то возвращено значение $\underline{m}$.
- Иначе произойдёт редукция к $H\,\underline{m+1}\,\underline{n}$.

Нормальная стратегия редукций

- Идея «ленивого вычисления» реализуется **нормальной стратегией редукции:** каждый раз редуцировать тот редекс, чья λ стоит левее всего в терме.

Нормальная стратегия редукций

- Идея «ленивого вычисления» реализуется **нормальной стратегией редукции:** каждый раз редуцировать тот редекс, чья λ стоит левее всего в терме.

Теорема

Если $U \in \mathrm{WN}$, *то нормальная стратегия редукции приведёт* U *к нормальной форме (за конечное число шагов).*

Нормальная стратегия редукций

- Идея «ленивого вычисления» реализуется **нормальной стратегией редукции:** каждый раз редуцировать тот редекс, чья λ стоит левее всего в терме.

Теорема

Если $U \in \mathrm{WN}$, *то нормальная стратегия редукции приведёт* U *к нормальной форме (за конечное число шагов).*

- Общая идея: взять какую-то последовательность редукций и переставить их так, чтобы получилась нормальная.

Нормальная стратегия редукций

- Идея «ленивого вычисления» реализуется **нормальной стратегией редукции:** каждый раз редуцировать тот редекс, чья λ стоит левее всего в терме.

Теорема

Если $U \in \mathrm{WN}$, *то нормальная стратегия редукции приведёт* U *к нормальной форме (за конечное число шагов).*

- Общая идея: взять какую-то последовательность редукций и переставить их так, чтобы получилась нормальная.
- Для доказательства по индукции нам пригодятся «пакетные» редукции.

«Пакетные» и другие редукции

- Рекурсивное определение нормальной стратегии:
  1. $(\lambda x.P)Q \to_n P[x := Q]$;
  2. если $M_1 \to_n M_2$, то $\lambda x.M_1 \to_n \lambda x.M_2$;
  3. если $M_1 \to_n M_2$ и $M_1 \neq \lambda x.P$, то $(M_1 N) \to_n (M_2 N)$;
  4. если $N_1 \to_n N_2$ и $M \in \mathrm{NF}$, то $(MN_1) \to_n (MN_2)$.

«Пакетные» и другие редукции

- Рекурсивное определение нормальной стратегии:
  1. $(\lambda x.P)Q \to_n P[x := Q]$;
  2. если $M_1 \to_n M_2$, то $\lambda x.M_1 \to_n \lambda x.M_2$;
  3. если $M_1 \to_n M_2$ и $M_1 \neq \lambda x.P$, то $(M_1 N) \to_n (M_2 N)$;
  4. если $N_1 \to_n N_2$ и $M \in \mathrm{NF}$, то $(MN_1) \to_n (MN_2)$.
- Головная редукция:
  $((\lambda z.P)Q)\, K_1 \ldots K_m \to_h P[z := Q]\, K_1 \ldots K_m$.

«Пакетные» и другие редукции

- Рекурсивное определение нормальной стратегии:
  1. $(\lambda x.P)Q \to_n P[x := Q]$;
  2. если $M_1 \to_n M_2$, то $\lambda x.M_1 \to_n \lambda x.M_2$;
  3. если $M_1 \to_n M_2$ и $M_1 \neq \lambda x.P$, то $(M_1 N) \to_n (M_2 N)$;
  4. если $N_1 \to_n N_2$ и $M \in \mathrm{NF}$, то $(MN_1) \to_n (MN_2)$.
- Головная редукция:
  $((\lambda z.P)Q)\, K_1 \ldots K_m \to_h P[z := Q]\, K_1 \ldots K_m$.
- Пакетная редукция:
  1. $M \to_\ell M$;
  2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
  3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
  4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.

«Пакетные» и другие редукции

- Рекурсивное определение нормальной стратегии:
  1. $(\lambda x.P)Q \to_n P[x := Q]$;
  2. если $M_1 \to_n M_2$, то $\lambda x.M_1 \to_n \lambda x.M_2$;
  3. если $M_1 \to_n M_2$ и $M_1 \neq \lambda x.P$, то $(M_1 N) \to_n (M_2 N)$;
  4. если $N_1 \to_n N_2$ и $M \in \mathrm{NF}$, то $(MN_1) \to_n (MN_2)$.
- Головная редукция:
  $((\lambda z.P)Q)\, K_1 \ldots K_m \to_h P[z := Q]\, K_1 \ldots K_m$.
- Пакетная редукция:
  1. $M \to_\ell M$;
  2. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(\lambda x.M_1)N_1 \to_\ell M_2[x := N_2]$;
  3. если $M_1 \to_\ell M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_\ell (M_2 N_2)$;
  4. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_\ell \lambda y.M_2$.
- Пакетная внутренняя редукция (противоположно головной):
  1. $x \to_{\ell i} x$, $x \in \mathrm{Var}$;
  2. если $M_1 \to_\ell M_2$, то $\lambda y.M_1 \to_{\ell i} \lambda y.M_2$;
  3. если $M_1 \to_{\ell i} M_2$ и $N_1 \to_\ell N_2$, то $(M_1 N_1) \to_{\ell i} (M_2 N_2)$.

Свойства головных и внутренних редукций

Лемма (о подстановке)

Если $M_1 \neq xK_1 \ldots K_m$, $M_1 \to_{\ell i} M_2$ *и* $N_1 \to_{\ell} N_2$, *то* $M_1[x := N_1] \to_{\ell i} M_2[x := N_2]$.

Свойства головных и внутренних редукций

Лемма (о подстановке)

Если $M_1 \neq xK_1 \ldots K_m$, $M_1 \to_{\ell i} M_2$ *и* $N_1 \to_{\ell} N_2$, *то* $M_1[x := N_1] \to_{\ell i} M_2[x := N_2]$.

Лемма (о разложении)

Если $M_1 \to_{\ell} M_2$, *то существует терм* $\hat{M}$, *такой что* $M_1 \twoheadrightarrow_h \hat{M} \to_{\ell i} M_2$.

Свойства головных и внутренних редукций

Лемма (о подстановке)

Если $M_1 \neq xK_1 \ldots K_m$, $M_1 \to_{\ell i} M_2$ *и* $N_1 \to_{\ell} N_2$, *то* $M_1[x := N_1] \to_{\ell i} M_2[x := N_2]$.

Лемма (о разложении)

Если $M_1 \to_{\ell} M_2$, *то существует терм* $\hat{M}$, *такой что* $M_1 \twoheadrightarrow_h \hat{M} \to_{\ell i} M_2$.

Лемма (о перестановке)

Если $M \to_{\ell i} N \to_h K$, *то существует терм* $\hat{N}$, *такой что* $M \twoheadrightarrow_h \hat{N} \to_{\ell i} K$.

Свойства головных и внутренних редукций

Лемма (о подстановке)

Если $M_1 \neq xK_1 \dots K_m$, $M_1 \to_{\ell i} M_2$ *и* $N_1 \to_{\ell} N_2$, *то* $M_1[x := N_1] \to_{\ell i} M_2[x := N_2]$.

Лемма (о разложении)

Если $M_1 \to_{\ell} M_2$, *то существует терм* $\hat{M}$, *такой что* $M_1 \twoheadrightarrow_h \hat{M} \to_{\ell i} M_2$.

Лемма (о перестановке)

Если $M \to_{\ell i} N \to_h K$, *то существует терм* $\hat{N}$, *такой что* $M \twoheadrightarrow_h \hat{N} \to_{\ell i} K$.

Лемма (о разложении последовательности)

Если $M_1 \twoheadrightarrow_{\ell} M_2$ (*т.е.* $M_1 \twoheadrightarrow_{\beta} M_2$), *то существует терм* $\hat{M}$, *такой что* $M_1 \twoheadrightarrow_h \hat{M} \twoheadrightarrow_{\ell i} M_2$.

Нормальная стратегия редукций

- Теперь мы готовы доказать теорему о нормальной стратегии редукций: если $M \twoheadrightarrow_\beta N \in \mathrm{NF}$, то $M \twoheadrightarrow_n N$.

Нормальная стратегия редукций

- Теперь мы готовы доказать теорему о нормальной стратегии редукций: если $M \twoheadrightarrow_\beta N \in \mathrm{NF}$, то $M \twoheadrightarrow_n N$.
- Рассуждаем индукцией по сложности N.
- Случаи:
  1. $N = \lambda y.N_1$;
  2. $N = zN_1 \ldots N_k$.

Нормальная стратегия редукций

- Теперь мы готовы доказать теорему о нормальной стратегии редукций: если $M \twoheadrightarrow_\beta N \in \mathrm{NF}$, то $M \twoheadrightarrow_n N$.
- Рассуждаем индукцией по сложности N.
- Случаи:
  1. $N = \lambda y.N_1$;
  2. $N = zN_1 \ldots N_k$.
- Имеем $M \twoheadrightarrow_h \hat{M} \twoheadrightarrow_{\ell i} N$, и поскольку редукции во второй части внутренние, $\hat{M}$ имеет ту же структуру, что и N.

Результаты об алгоритмической неразрешимости

- За счёт нормальной стратегии редукций, если некоторая (частичная) функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ представима в λ-исчислении некоторым термом F, то эта функция вычислима.

Результаты об алгоритмической неразрешимости

- За счёт нормальной стратегии редукций, если некоторая (частичная) функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ представима в λ-исчислении некоторым термом F, то эта функция вычислима.
- Действительно, если $f(n)$ определено, то $F\,\underline{n} \in \mathrm{WN}$, и нормальная стратегия его нормализует. А если $F\,\underline{n} \notin \mathrm{WF}$, то нормальная стратегия (как и любая другая) будет работать бесконечно.

Результаты об алгоритмической неразрешимости

- За счёт нормальной стратегии редукций, если некоторая (частичная) функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ представима в λ-исчислении некоторым термом F, то эта функция вычислима.
- Действительно, если $f(n)$ определено, то $F\,\underline{n} \in \mathrm{WN}$, и нормальная стратегия его нормализует. А если $F\,\underline{n} \notin \mathrm{WF}$, то нормальная стратегия (как и любая другая) будет работать бесконечно.
- Таким образом, в λ-исчислении представимы в точности все вычислимые функции.

Результаты об алгоритмической неразрешимости

- За счёт нормальной стратегии редукций, если некоторая (частичная) функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ представима в λ-исчислении некоторым термом F, то эта функция вычислима.
- Действительно, если $f(n)$ определено, то $F\,\underline{n} \in \mathrm{WN}$, и нормальная стратегия его нормализует. А если $F\,\underline{n} \notin \mathrm{WF}$, то нормальная стратегия (как и любая другая) будет работать бесконечно.
- Таким образом, в λ-исчислении представимы в точности все вычислимые функции.

Следствие

WN *алгоритмически неразрешимо.*

Результаты об алгоритмической неразрешимости

- За счёт нормальной стратегии редукций, если некоторая (частичная) функция $f : \mathbb{N} \dashrightarrow \mathbb{N}$ представима в λ-исчислении некоторым термом F, то эта функция вычислима.
- Действительно, если $f(n)$ определено, то $F\,\underline{n} \in \mathrm{WN}$, и нормальная стратегия его нормализует. А если $F\,\underline{n} \notin \mathrm{WF}$, то нормальная стратегия (как и любая другая) будет работать бесконечно.
- Таким образом, в λ-исчислении представимы в точности все вычислимые функции.

Следствие

WN *алгоритмически неразрешимо.*

- SN также неразрешимо, но доказательство сложнее.