Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский
Академический университет Российской академии наук»
Центр высшего образования

Кафедра математических и информационных технологий

Жаворонков Эдгар Андреевич

Реализация сравнения различных представлений λ-термов

Магистерская диссертация

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Омельченко А. В.

Научный руководитель:
преп. Исаев В. И.

Рецензент:
ООО "ИнтеллиДжей Лабс", программист Березун Д. А.

Санкт-Петербург
2017

Оглавление

Введение

Мотивация Верификация программного обеспечения используется в важнейших отраслях индустрии разработки программного обеспечения и включает в себя формальные рассуждения о программах и их свойствах. Так как языки программирования используют связывание имен, то в программах на них зачастую возникают проблема коллизий в именах переменных, функций и других конструкций.

Языки программирования или средства для разработки пытаются сами решить эту проблему. Например анализатор кода в среде разработке может подсказать о том, что такое имя уже занято и программисту следует придумать новое. Некоторые языки разрешают имена с помощью системы модулей(или пространств имен, как например в языке C++).

Более того, иногда возникает желание рассматривать конструкции в языке программирования с точностью до имен параметров. Это полезно, например, при поиске фрагментов кода, которые производят одинаковые вычисления или подчиняются некоторому общему шаблону. Пример таких фрагментов – всевозможные циклы и обходы контейнеров.

Соответственно, для того, чтобы доказывать какие-либо свойства программ необходима формальная система, позволяющая решать проблему коллизий имен и формально описывающая отношение так называемой α-эквивалентности – эквивалентности в поведении конструкций с разными именами формальных параметров. Пример такой системы – это λ-исчисление, лежащее в основе функциональных языков программирования. В работе делается попытка формализовать различные представления этой системы и понять, с каким из них наиболее удобно работать.

План работы В первой главе будет дан анализ предметной области, краткое описание λ-исчисления. Мы обозначим цель работы и задачи, решение которых необходимо для её достижения. Кроме того, будут рассмотрены уже существующие решения и описаны их отличия от представленного в работе.

Во второй главе будут описаны три представления λ-термов. Для каждого из представлений мы определим характерные свойства и покажем, почему они верны. Кроме того, мы покажем, что эти представления эквивалентны между собой.

В третьей главе предполагается описание деталей реализации. Мы опишем язык, с помощью которого предлагается формализовать рассмотренные во второй главе представления а так же тонкие моменты, с которыми пришлось столкнуться в ходе выполнения работы.

1. Описание предметной области

1.1. λ-исчисление

Историческая справка Лямбда-исчисление – это формальная система, придуманная в 30-ых годах прошлого века Алонзо Черчём [3] с целью анализа и формализации понятия вычислимости. В 60-ых годах Питером Ландином была опубликована работа [9], в которой выдвигалась идея о том, что λ-исчисление может использоваться для моделирования различных выражений в языках программирования того времени, что в дальнейшем привело к развитию языков в стиле **ML**. С тех пор идеи λ-исчисления широко используются в мире функционального программирования.

Неформальное описание λ-термов Мы формально определим λ-термы во второй главе, здесь же мы просто скажем, что термы λ-исчисления рекурсивно конструируются из переменных с помощью всего двух операций – применения функции к аргументу и создания анонимной функции. Наличие каких-либо констант здесь не предполагается. Несмотря на кажущуюся простоту, λ-исчисление является очень мощной формальной системой, в частности, Шейнфинкелем и Карри в работах [13, 5] введен в рассмотрение базис из двух термов(комбинаторов) $S = \lambda fgx.fx(gx)$ и $K = \lambda xy.x$, который примечателен тем, что обладает полнотой по Тьюрингу.

Отметим, что лямбда-термы допускают не единственный способ записи. В работе мы неоднократно будем употреблять термин «представление термов». Под ним мы будем понимать способ, которым термы кодируются на каком-либо языке. Мотивация к появлению различных представлений λ-термов состоит в том, что являясь каким-никаким, но языком программирования лямбда-исчисление само столкнулось с проблемой коллизии имен переменных. Различные представления термов по-разному решают эту проблему. Как следствие, какие-то представления удобнее для восприятия человеком и неформальных рассуждений «на бумаге». Какие-то представления удобны для компьютерной реализации и используются как внутреннее представление в функциональных языках программирования или системах автоматического доказательства теорем – [11].

Вариации и расширения Изначально, в λ-исчислении не вводилось никаких правил типизации, однако в дальнейшем появилось множество типизированных вариаций. Барендрегтом в [1] описан так называемый λ-куб, который наглядно классифицирует восемь различных систем типизации лямбда-исчисления.

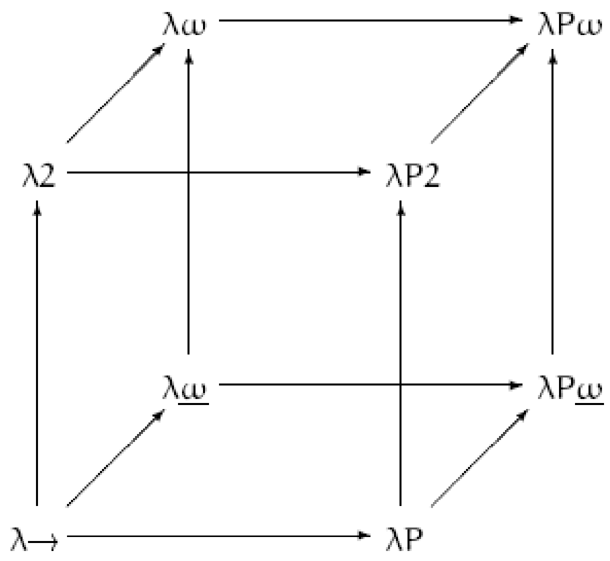

Рис. 1: Лямбда-куб

База куба – просто типизированное λ-исчисление($\lambda\rightarrow$), в котором термы могут зависеть только от термов. Три оси соответствуют расширениям, комбинации которых позволяют получить остальные системы типов:

1. Термы, которые зависят от типов – система $\lambda 2$ или **System F**
2. Типы, которые зависят от типов – система $\lambda\underline{\omega}$(операторы над типами)
3. Типы, которые зависят от термов – система λP(зависимые типы)

Соответствие Карри-Говарда Затронув системы типизации лямбда-исчисления, нельзя не отметить так называемое соответствие Карри-Говарда [7], которое устанавливает прямую связь между логикой и теорией типов. Логической связке соответствует конструкция в теории типов, а логическому утверждению – тип. Доказательству того факта, что утверждение истинно, соответствует тогда доказательство того факта, что соответствующий этому утверждению тип населен. Иначе говоря, мы можем предъявить λ-терм соответствующего типа, чтобы доказать исходное утверждение. Для наглядности некоторые соответствия сведены в таблицу:

Высказывание A:	Тип A:
Истинно	Населен
Тождественная истина	$\top$(единичный тип)
Тождественная ложь	$\bot$(пустой тип без обитателей)
$\neg A$(отрицание)	$A \to \bot$
$A \land B$(конъюнкция)	$A \times B$(тип-произведение)
$A \lor B$(дизъюнкция)	$A \coprod B$(тип-сумма)
$A \to B$(импликация)	$A \to B$(тип функций из A в B)
$\exists x.P(x)$	$\Sigma(x : A)(Pa)$(тип зависимых пар)
$\forall x.P(x)$	$\Pi(x : A) \to (Pa)$(тип зависимых функций)

Таблица 1: Соответствия высказываний в логике и конструкций в теории типов

Чем больше логических связок мы хотим использовать, тем более мощные теории типов нам придется использовать, чтобы доказывать эти утверждения. Так, например, если нам потребуется доказывать формулы пропозициональной логики, то мы можем обойтись просто типизированным λ-исчислением. Если нам понадобятся кванторы в формулах, то на помощь приходят теории с зависимыми типами. Благодаря соответствию Карри-Говарда процесс доказательства утверждений становится похож на программирование, следовательно, задача проверки корректности доказательства сводится к задаче проверки типов.

Существует две известные теории с зависимыми типами – исчисление конструкций(Calculus Of Constructions), представленное Тьерри Коканом в [4] и интуиционистская теория типов Мартин-Лёфа(Martin-Löf Type Theory), описанная в [10]. Их расширения лежат в основе таких систем интерактивного доказательства теорем как **Coq** и **Agda** соответственно.

1.2. Постановка задачи

Целью работы является реализация сравнения между собой различных представлений λ-термов. Для достижения этой цели необходимо решить следующие задачи:

1. Описать типы данных, кодирующие термы в соответствующем представлении
2. Сформулировать свойства термов и операции над ними:
   (a) Определить отношение α-эквивалентности на множестве именованных термов
   (b) Для каждого представления определить операцию подстановки терма
3. Для каждого представления доказать сформулированные в предыдущем пункте свойства термов

4. Установить соответствие между описанными в первом пункте представлениями

TODO

1.3. Существующие решения

Стоит отметить, что задача формализации λ-исчисления довольно популярна, в связи с чем существует довольно много её решений. Один из примеров – [12], в котором, в частности, формализовано чистое λ-исчисление.

Однако для этого, и аналогичных ему решений характерен тот факт, что авторы рассматривают лишь одно представление λ-термов, упуская из внимания остальные и, соответственно, не устанавливая между ними никакого соответствия.

Кроме того, наше решение будет использовать другую теорию типов! Как следствие, мы будем писать на другом языке, на котором еще никто не формализовывал λ-исчисление. Мы дадим описание этого языка в третьей главе, когда будем рассказывать об особенностях реализации.

2. Представления λ-термов

В этой главе мы опишем три представления λ-термов: именованное, неименованное(через индексы Де Брауна) и монадическое. Мы формально опишем, как определяются термы в каждом из представлений, определим характерные свойства для каждого представления и покажем, почему они верны. Кроме того, мы установим соответствие между всеми представлениями.

2.1. Именованное представление термов

Термы λ-исчисления(λ-термы) в именованном представлении конструируются из переменных путем применения друг к другу или создания анонимных функций.

Формально, пусть $\mathcal{V} = \{x, y, z, \dots\}$ – счетное множество переменных. Договоримся обозначать переменные прописными буквами, а произвольные термы – заглавными. Тогда множество λ-термов Λ определяется индуктивно, согласно следующим правилам:

$$\frac{v \in \mathcal{V}}{v \in \Lambda}$$

$$\frac{M \in \Lambda \qquad N \in \Lambda}{MN \in \Lambda}$$

$$\frac{M \in \Lambda \qquad v \in \mathcal{V}}{\lambda v.M \in \Lambda}$$

Нотация аппликации MN обозначает применение функции M ко входу N. Заметим, что так как здесь не вводится никаких правил типизации, то ничто не мешает нам применить терм к самому себе(i.e FF). Нотация абстракции $\lambda x.M$, в свою очередь, обозначает создание анонимной функции от переменной x, которая сопоставляет конкретному значению x выражение $M[x]$. Здесь заметим, что терм M вовсе не обязан содержать в себе переменную x, в таком случае мы считаем абстракцию $\lambda x.M$ константной функцией.

Некоторые примеры термов:

$$\lambda x.x$$

$$\lambda xy.x$$

$$(\lambda x.f(xx))(\lambda x.f(xx))$$

Переменная x после абстракции $\lambda x.M$ называется *связанной*. Соответственно, до абстракции она была *свободной*. Формально, множества $FV(T)$ свободных и $BV(T)$

связанных переменных терма T определяются индуктивно следующим образом:

$$\begin{aligned}
FV(x) &= \{x\} \\
FV(MN) &= FV(M) \cup FV(N) \\
FV(\lambda x.M) &= FV(M) \setminus \{x\}
\end{aligned}$$

$$\begin{aligned}
BV(x) &= \emptyset \\
BV(MN) &= BV(M) \cup BV(N) \\
BV(\lambda x.M) &= \{x\} \cup BV(M)
\end{aligned}$$

Применение абстракции к некоторому аргументу $(\lambda x.M)N$ – это *подстановка* $M[x \mapsto N]$ терма N вместо *свободных* вхождений переменной x в терме M. Формально, правила подстановки:

$$\begin{aligned}
x[x \mapsto N] &= N \\
y[x \mapsto N] &= y, (x \neq y) \\
(TS)[x \mapsto N] &= T[x \mapsto N]S[x \mapsto N] \\
(\lambda x.T)[x \mapsto N] &= \lambda x.T \\
(\lambda y.T)[x \mapsto N] &= \lambda y.T[x \mapsto N], (y \notin FV(N), x \neq y)
\end{aligned}$$

Рассмотрим, что произойдет, если в последнем правиле условие $y \notin FV(N)$ не выполняется:

$$(\lambda y.x)[x \mapsto y] = \lambda y.y$$

Получилось, что в результате подстановки мы превратили константную функцию $\lambda y.x$ в тождественную. Такая ситуация называется проблемой захвата переменной, когда при подстановке в λ-абстракцию переменные подставляемого терма захватываются абстракцией.

Эту проблему можно решить если принять так называемое соглашение Барендрегта о том, что имена связанных переменных всегда выбирать так, чтобы они отличались от имен свободных. В примере выше, например, мы можем переименовать связанную переменную y в свежую z и поведении абстракции $\lambda z.x$ не изменится. Тогда подстановку можно использовать без каких-либо оговорок о свободных и связанных переменных. Пример выше превратится в:

$$(\lambda z.x)[x \mapsto y] = \lambda z.y$$

Как мы уже установили выше, мы можем переименовывать связанные переменные в абстракциях и их поведение при применении к аргументам не изменится. Более того, имена связанных переменных не играют для нас никакой роли. Поэтому, как правило, λ-термы и рассматривают с точностью до имен параметров абстракций. Формально, на множестве именованных термов Λ можно задать отношение $=_\alpha \in \Lambda \times \Lambda$, которое называется α-эквивалентностью и определяется как минимальное отношение конгруэнтности, порожденное следующими правилами:

$$\frac{x \in \mathcal{V}}{x =_\alpha x}$$

$$\frac{M =_\alpha M' \qquad N =_\alpha N'}{MN =_\alpha M'N'}$$

$$\frac{M[x \mapsto y] =_\alpha M' \qquad y \notin FV(M)}{\lambda x.M =_\alpha \lambda y.M'}$$

Наконец, сформулируем лемму о подстановке:

***Утверждение* 1.** *Для любых $T, M, N \in \Lambda; x, y \in \mathcal{V}$, если $x \neq y$ и $x \notin FV(M)$, то верно $T[x \mapsto N][y \mapsto M] = T[y \mapsto M][x \mapsto N[y \mapsto M]]$*

Доказательство. Индукция по терму T. База индукции – случай, когда T является переменной. Рассмотрим три случая:

1. $T \equiv x$. Левая часть – $x[x \mapsto N][y \mapsto M] = N[y \mapsto M]$. Правая часть – $x[y \mapsto M][x \mapsto N[y \mapsto M]] = N[y \mapsto M]$.

2. $T \equiv y$. Левая часть – $y[x \mapsto N][y \mapsto M] = M$. Правая часть – $y[y \mapsto M][x \mapsto N[y \mapsto M]] = M[x \mapsto N[y \mapsto M]]$. Так как $x \notin FV(M)$, то $M[x \mapsto N[y \mapsto M]] = M$

3. $T \equiv z \neq x, y$. Обе части редуцируются к z.

Случай аппликации тривиален, рассмотрим случай абстракции $\lambda z.T$. Рассмотрим возможные случаи:

1. $z \equiv x$. Левая часть – $(\lambda x.T)[x \mapsto N][y \mapsto M] = (\lambda x.T)[y \mapsto M] \overset{\mathrm{x \notin FV(M)}}{=} \lambda x.T[y \mapsto M]$. Правая часть – $(\lambda x.T)[y \mapsto M][x \mapsto N[y \mapsto M]] \overset{\mathrm{x \notin FV(M)}}{=} (\lambda x.T[y \mapsto M])[x \mapsto N[y \mapsto M]] = \lambda x.T[y \mapsto M]$.

2. $z \equiv y$. Пусть $y \in FV(N)$ и $y \in FV(M)$. По соглашению Барендрегта, нам нужно переименовать y в свежую переменную y', такую что $y' \notin FV(N)$ и $y' \notin FV(M)$. Это эквивалентно тому, что мы можем осуществить подстановку $\lambda y'.t[y \mapsto y']$. Вычислим левую часть:

$$(\lambda y'.T[y \mapsto y'])[x \mapsto N][y \mapsto M] =$$
$$\lambda y'.T[y \mapsto y'][x \mapsto N][y \mapsto M] \overset{\mathrm{IH}}{=}$$
$$\lambda y'.T[y \mapsto y'][y \mapsto M][x \mapsto N[y \mapsto M]]$$

Правая часть:

$$(\lambda y'.T[y \mapsto y'])[y \mapsto M][x \mapsto N[y \mapsto M]] =$$
$$\lambda y'.T[y \mapsto y'][y \mapsto M][x \mapsto N[y \mapsto M]]$$

Заметим здесь, что так как $y' \notin FV(N)$ и $y' \notin FV(M)$, то, очевидно, что $y' \notin FV(N[y \mapsto M])$, поэтому в последнем переходе обе подстановки «проваливаются» в абстракцию.

3. $z \equiv y$. Пусть $y \in FV(N)$, но теперь уже $y \notin FV(M)$. Аналогично предыдущему пункту, мы переименовываем y в свежую переменную y', такую что $y' \notin FV(N)$ и $y' \notin FV(M)$, осуществляя подстановку $\lambda y'.t[y \mapsto y']$. Вычислим левую часть:

$$(\lambda y'.T[y \mapsto y'])[x \mapsto N][y \mapsto M] =$$
$$\lambda y'.T[y \mapsto y'][x \mapsto N][y \mapsto M] \overset{\text{IH}}{=}$$
$$\lambda y'.T[y \mapsto y'][y \mapsto M][x \mapsto N[y \mapsto M]]$$

А в этом случае, заметим, что так как мы сначала заменили **все** свободные вхождения y в T на y', а потом на место y подставили M, то вторая подстановка ничего не делает, поэтому левая часть окончательно равна $\lambda y'.T[y \mapsto y'][x \mapsto N[y \mapsto M]]$

Правая часть тогда редуцируется в:

$$(\lambda y.T)[y \mapsto M][x \mapsto N[y \mapsto M]] =$$
$$(\lambda y.T)[x \mapsto N[y \mapsto M]] =$$
$$\lambda y.T[x \mapsto N[y \mapsto M]]$$

В последнем шаге вычисления, отметим, что так как $y \in FV(N)$ и $y \notin FV(M)$, то $y \notin FV(N[y \mapsto M])$, поэтому подстановка снова заносится под абстракцию. Переименуем и здесь y в y', тогда получим:

$$\lambda y.T[x \mapsto N[y \mapsto M]] =_\alpha$$
$$\lambda y'.T[x \mapsto N[y \mapsto M]][y \mapsto y'] \overset{\text{IH}}{=}$$
$$\lambda y'.T[y \mapsto y'][x \mapsto N[y \mapsto M][y \mapsto y']]$$

Из тех же соображений, что и выше, подстановка $N[y \mapsto M][y \mapsto y']$ – это то же самое, что и $N[y \mapsto M]$, поэтому правая часть окончательно вычисляется в $\lambda y'.T[y \mapsto y'][x \mapsto N[y \mapsto M]]$.

4. $z \neq x, y$. Доказательство аналогично предыдущим двум пунктам. ∎

2.2. Неименованное представление термов

Как уже мы уже видели в предыдущем разделе, имена формальных параметров λ-абстракций не важны и, в целом, мы можем не обращать на них внимания. Более того, мы можем вообще отказаться от именованных переменных! Широко известен альтернативный способ записи термов через так называемые индексы Де Брауна(De Bruijn) – [6]. В нем вместо имен переменных используются числовые индексы, показывающие сколько лямбд назад была захвачена переменная. Например комбинатор $S = \lambda fgx.fx(gx)$, записанный в таком представлении будет иметь следующий вид: $\lambda(\lambda(\lambda 31(21)))$.

Существует и альтернативный способ такого представления. Множество всех термов разбивается на так называемые «уровни»(levels) и вместо него рассматриваются множества Λ_n, где n – длина контекста, в котором определен терм. О контексте в котором определен терм, можно думать, как о простом списке свободных переменных терма. Индуктивно, они определяются следующим образом:

$$\frac{0 \leqslant i < n}{v_{n,i} \in \Lambda_n}$$

$$\frac{T_1 \in \Lambda_n \qquad T_2 \in \Lambda_n}{T_1 T_2 \in \Lambda_n}$$

$$\frac{T \in \Lambda_{n+1}}{\lambda T \in \Lambda_n}$$

В случае переменной индекс i обозначает позицию переменной в контексте. Договоримся отсчитывать ее с конца контекста. Комбинатор S, например, в таком представлении будет выглядеть вот так: $\lambda(\lambda(\lambda v_{3,2} v_{3,0}(v_{3,1} v_{3,0})))$.

Такое представление термов удобно потому что α-эквивалентность сводится к самому обычному равенству и, как следствие, пропадает проблема коллизии имен переменных.

Определим операцию подстановки для таких термов. Мы определим ее в более общем случае – вместо какой-то одной переменной мы будем осуществлять подстановку во **все** переменные терма. Пусть $T \in \Lambda_n$, и $S_0, \dots S_{n-1} \in \Lambda_k$. Тогда $subst(T, S_{n-1}, \dots, S_0) \in \Lambda_k$ определяется следующим образом:

$$subst(v_{n,i}, S_{n-1}, \dots, S_0) = S_i$$

$$subst(T, v_{n,n-1}, \dots, v_{n,0}) = T$$

$$subst(T_1 T_2, S_{n-1}, \dots, S_0) =$$

$$subst(T_1, S_{n-1}, \dots, S_0)\ subst(T_2, S_{n-1}, \dots, S_0)$$

$$subst(\lambda T, S_{n-1}, \dots, S_0) = \lambda(subst(T, w(S_{n-1}), \dots, w(S_0), v_{n+1,0})$$

Операция $w(T)$ работает следующим образом. Пусть терм $T \in \Lambda_n$, тогда терм $w(T) \in \Lambda_{n+1}$ и определен как:

$$w(v_{n,i}) = v_{n+1,i+1}$$
$$w(T_1T_2) = w(T_1)\ w(T_2)$$
$$w(\lambda T) = \lambda(w(T))$$

Сформулируем и докажем вспомогательную лемму, которая пригодится нам далее:

***Лемма* 1.** *Пусть* $T \in \Lambda_n$, *a* $S_{n-1}, \dots, S_0 \in \Lambda_m$. *Тогда* $subst(w(T), w(S_{n-1}), \dots, w(S_0), v_{m+1,0}) = w(subst(T, S_{n-1}, \dots S_0))$

Доказательство. Индукция по структуре терма T.

1. База индукции – $v_{n,i}$. Левая часть равна, по определению подстановки:

$$subst(v_{n+1,i+1}, w(S_{n-1}), \dots, w(S_0), v_{m+1,0}) = w(S_i)$$

   Правая часть:

$$w(subst(v_{n,i}, S_{n-1}, \dots, S_0)) = w(S_i)$$

2. Случай аппликации снова тривиален. Рассмотрим случай абстракции λT. Левая часть вычислится в:

$$subst(w(\lambda T), w(S_{n-1}), \dots, w(S_0), v_{m+1,0}) =$$
$$subst(\lambda w(T), w(S_{n-1}), \dots, w(S_0), v_{m+1,0}) =$$
$$\lambda subst(w(T), w(w(S_{n-1})), \dots, w(w(S_0)), v_{m+2,1}, v_{m+2,0}) \overset{\text{IH}}{=}$$
$$\lambda w(subst(T, w(S_{n-1}), \dots, w(S_0), v_{m+1,0}))$$

   Правая часть вычисляется в:

$$w(subst(\lambda T, S_{n-1}, \dots, S_0)) =$$
$$w(\lambda subst(T, w(S_{n-1}), \dots, w(S_0), v_{m+1,0})) = \lambda w(subst(T, w(S_{n-1}), \dots, w(S_0), v_{m+1,0}))$$

■

Аналогично именованному представлению, сформулируем лемму о подстановке:

***Утверждение* 2.** *Пусть* $T \in \Lambda_n; T_{n-1}, \dots T_0 \in \Lambda_m, S_{m-1}, \dots S_0 \in \Lambda_k$, *тогда верно* $subst(subst(T, T_{n-1}, \dots T_0), S_{m-1}, \dots S_0) = subst(T, T'_{n-1} \dots, T'_0)$, *где* $T'_i = subst(T_i, S_{m-1}, \dots S_0)$.

Заметим еще, что в таком представлении термов нет необходимости в сторонних условиях, как в лемме о подстановке для именованных термов.

Доказательство. Это утверждение точно так же доказывается индукцией по структуре терма T. База индукции – случай, когда терм представляет собой переменную $v_{n,i}$. Тогда левая часть вычисляется в $subst(T_i, S_{m-1}, \dots S_0)$, ровно как и правая.

Случай аппликации снова тривиален, рассмотрим случай абстракции λT. Вычислим левую часть:

$$subst(subst(\lambda T, T_{n-1}, \dots, T_0), S_{m-1}, \dots, S_0) =$$
$$subst(\lambda subst(T, w(T_{n-1}), \dots w(T_0), v_{m+1,0}), S_{m-1}, \dots, S_0) =$$
$$\lambda(subst(subst(T, w(T_{n-1}), \dots w(T_0), v_{m+1,0}), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}) \overset{\text{IH}}{=}$$
$$\lambda(subst(T, subst(w(T_{n-1}), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}), \dots,$$
$$subst(w(T_0), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}), subst(v_{m+1,0}, w(S_{m-1}), \dots, w(S_0), v_{k+1,0}))) =$$
$$\lambda(subst(T, subst(w(T_{n-1}), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}), \dots,$$
$$subst(w(T_0), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}), v_{k+1,0}))$$

Вычислим теперь правую часть:

$$subst(\lambda T, subst(T_{n-1}, S_{m-1}, \dots, S_0), \dots, subst(T_0, S_{m-1}, \dots, S_0)) =$$
$$\lambda(subst(T, w(subst(T_{n-1}, S_{m-1}, \dots, S_0)), \dots, w(subst(T_0, S_{m-1}, \dots, S_0)), v_{k+1,0}))$$

По лемме 1 $subst(w(T_i), w(S_{m-1}), \dots, w(S_0), v_{k+1,0}) = w(subst(T_i, S_{m-1}, \dots, S_0))$ для всех $i = \overline{0, n-1}$, следовательно наше утверждение верно. ■

2.3. Монадическое представление термов

Существует еще один способ записи λ-термов, описанный в [2]. Библиотека **Bound** [8] для языка **Haskell**, например, использует именно монадическое представление.

Основная идея в том, что именованное представление для термов можно обобщить и свободные переменные брать из произвольного множества V. Тогда множество термов Λ_V определяется индуктивно по следующим правилам:

$$\frac{v \in V}{v \in \Lambda_V}$$

$$\frac{M \in \Lambda_V \qquad N \in \Lambda_V}{MN \in \Lambda_V}$$

$$\frac{M \in \Lambda_{V \coprod \{*\}}}{\lambda M \in \Lambda_V}$$

Здесь $\{*\}$ – это произвольное одноэлементное множество, а $\coprod$ – операция размеченного объединения множеств. По определению $A \coprod B$ состоит из элементов $inl(a)$ и $inr(b)$, где $a \in A$ и $b \in B$. Так как для абстракции нам нужно иметь на одну свободную переменную больше, то ее можно получить взяв размеченное объединение с произвольным одноэлементным множеством.

Пусть у нас есть функция $f : V \to W$, тогда мы можем задать функцию F_f из Λ_V в Λ_W рекурсией по терму $T \in \Lambda_V$:

1. $v \mapsto f(v)$
2. $M\ N \mapsto F_f(M)\ F_f(N)$
3. $\lambda M \mapsto \lambda F_{f'(f)}(M)$. Заметим, что просто так отобразить терм M с помощью функции f мы не можем, так как ее домен не совпадает со множеством, которым параметризован тип терма M. Поэтому мы построим по f функцию $f'(f) : V \coprod \{*\} \to W \coprod \{*\}$. Устроена она будет следующим образом:
   (a) $f'(f)(inl(x)) = inl(f(x))$
   (b) $f'(f)(inr(*)) = inr(*)$

Знакомый с языком программирования **Haskell** или теорией категорий читатель узнает, что мы задали структуру функтора. Интуитивно, действие этого функтора – это переименование переменных. Покажем, что это действительно функтор, именно, что он уважает тождественное отображение и композицию отображений.

***Утверждение* 3.** *Для любого $T \in \Lambda_V$ верно, что $F_{id_V}(T) = T$*

Доказательство. Индукция по терму T. База тривиальна, равно как и случай аппликации, покажем, что утверждение верно и для случая лямбды. Вспомогательная функция $f'(f)$ устроена следующим образом:

1. $f'(id_V)(inl(x)) = inl(x)$
2. $f'(id_V)(inr(*)) = inr(x)$

Следовательно, оно является тождеством на $V \coprod \{*\}$. По идукционной гипотезе получаем, что случай для лямбды тоже верен. ■

***Утверждение* 4.** *Для любого $T \in \Lambda_V$ и $f : V \to W$, $g : W \to X$ верно, что $F_{g\circ f}(T) = F_g(T) \circ F_f(T)$*

Доказательство. Снова индукция по терму T. Случай переменной снова тривиален, случай аппликации напрямую следуюет из индукционной гипотезы, рассмотрим случай абстракци. Посмотрим, во что вычислится левая часть:

$$F_{g\circ f}(\lambda M) = \lambda F_{f'(g\circ f)}(M)$$

где

$$f'(g \circ f)(inl(v)) = inl(g(f(v)))$$

$$f'(g \circ f)(inr(*)) = inr(*)$$

Правая, в свою очередь:

$$F_g(F_f(\lambda M)) = F_g(\lambda F_{f'(f)}(M)) =$$
$$\lambda F_{g'(g)}(F_{f'(f)}(M))$$

Вспомогательная функция $f'(f) : V \coprod \{*\} \to W \coprod \{*\}$ устроена здесь следующим образом:

1. $f'(f)(inl(v)) = inl(f(v))$
2. $f'(f)(inr(*)) = inr(*)$

Вспомогательная функция $g'(g) : W \coprod \{*\} \to X \coprod \{*\}$ устроена здесь следующим образом:

1. $g'(g)(inl(w)) = g'(g)(inl(f(v))) = inl(g(f(v)))$
2. $g'(g)(inr(*)) = inr(*)$

Несложно увидеть, что эти две функции из левой и правой частей ведут себя одинаково, следовательно и для случая лямбды утверждение верно. ■

Мы пойдем еще дальше и зададим структуру монады. Существует много способов определить её, мы воспользуемся тем, который принят в языке программирования **Haskell**. Именно, мы определим две операции: монадическую единицу `return` $: V \to \Lambda_V$ и монадическое связывание `bind` $: \Lambda_V \to (V \to \Lambda_W) \to \Lambda_W$. Кроме этого, мы покажем, что они удовлетворяют монадическим законам.

Монадическая единица устроена очень просто – это переменная. Связывание же, принимает на вход так называемую стрелку Клейсли [14] $k : V \to \Lambda_W$ и определяется индуктивно по структуре терма T:

1. $v \mapsto k(v)$
2. $M\ N \mapsto (\mathtt{bind}(M, k))\ (\mathtt{bind}(N, k))$
3. $\lambda M \mapsto \lambda(\mathtt{bind}(M, k'(k)))$, где $k'(k) : V \coprod \{*\} \to \Lambda_{W \coprod \{*\}}$ и определяется следующим образом:
   (a) $k'(k)(inl(v)) = F_{inl}(k(v))$
   (b) $k'(k)(inr(*)) = \mathtt{return}(inr(*))$

Сформулируем и докажем теперь свойства этих двух операций.

Утверждение 5. *Для любого $T \in \Lambda_V$ верно $\mathtt{bind}(T, \mathtt{return}) = T$*

Доказательство. Индукция по структуре терма T:

1. База индукции, $T = v$. Имеем, что $\texttt{bind}(v, \texttt{return}) = \texttt{return}(v) = v$.

2. Случай аппликации напрямую следует из предположения индукции.

3. Пусть теперь $T = \lambda M$. Имеем, что $\texttt{bind}(\lambda M, \texttt{return}) = \lambda(\texttt{bind}(M, k'(\texttt{return})))$. Вспомогательное отображение $k'(return)$ устроено следующим образом:

   (a) $k'(\texttt{return})(inl(v)) = F_{inl}(\texttt{return}(v)) = \texttt{return}(inl(v))$

   (b) $k'(\texttt{return})(inr(*)) = \texttt{return}(inr(*))$

Заметим, что оно ведет себя так же, как и $\texttt{return}$ на $V \coprod \{*\}$, следовательно по индукционной гипотезе получаем исходное утверждение. ■

Утверждение 6. *Для любого $v \in V$ и $k : V \to \Lambda_V$ верно $\texttt{bind}(\texttt{return}(v), k) = k(v)$*

Доказательство. Утверждение тривиально следует из определения $\texttt{bind}$ и того факта, что $\texttt{return}$ – это переменная. ■

Прежде, чем формулировать последний монадный закон, сформулируем и докажем несколько технических лемм, которые помогут нам в его доказательстве.

Лемма 2. *Для любых $T \in \Lambda_V$, $f : V \to W$ и $k : W \to \Lambda_U$ верно $\texttt{bind}(F_f(T), k) = \texttt{bind}(T, k \circ f)$.*

Доказательство. Индукция по структуре терма T с тривиальными случаями переменной и аппликации. Рассмотрим случай абстракции λM. Нам нужно показать, что $\lambda\texttt{bind}(F_{f'(f)}(M), k'(k)) = \lambda\texttt{bind}(M, k'(k \circ f))$. По индукционной гипотезе мы знаем, что $\texttt{bind}(F_{f'(f)}(M), k'(k)) = \texttt{bind}(M, k'(k) \circ f'(f))$, Заметим теперь, что $k'(k \circ f)$ и $k'(k) \circ f'(f)$ ведут себя одинаково на всех входах, следовательно это утверждение доказано. ■

Лемма 3. *Для любых $T \in \Lambda_V$, $f : W \to U$ и $k : V \to \Lambda_W$ верно $F_f(\texttt{bind}(T, k)) = \texttt{bind}(T, (x \mapsto F_f(k(x))))$.*

Доказательство. Снова индукция по структуре терма T. Случай переменной и аппликации снова тривиален, поэтому рассмотрим случай абстракции λM.

Нам нужно показать, что $\lambda F_{f'(f)}(\texttt{bind}(M, k'(k))) = \lambda\texttt{bind}(M, k'(x \mapsto F_f(k(x))))$. По индукционной гипотезе мы знаем, что $F_{f'(f)}(\texttt{bind}(M, k'(k))) = \texttt{bind}(M, (x \mapsto F_{f'(f)}(k'(k)(x))))$. Покажем теперь, что стрелки Клейсли $k'(x \mapsto F_f(k(x)))$ и $x \mapsto F_{f'(f)}(k'(k)(x))$ ведут себя одинаково на всех входах:

1. $inr(*)$. Обе стрелки вычисляются в $\texttt{return}(inr(*))$

2. $inl(v)$. Надо показать, что $F_{f'(inl)}(F_f(k(v))) = F_{f'(f)}(F_{inl}(k(v)))$. Так как $\Lambda_{\_}$ – функтор и он уважает композицию отображений имеем, что нужно показать $F_{f'(inl) \circ f}(k(v)) = F_{f'(f) \circ inl}(k(v))$. Для этого в свою очередь нужно снова показать, что два отображения $f'(inl) \circ f$ и $f'(f) \circ inl$ ведут себя одинаково на всех входах, но это очень легко понять, просто взглянув на определение f'.

■

Лемма 4. *Для любого $T \in \Lambda_V$ и $f : V \to \Lambda_W$ верно $F_{inl}(\texttt{bind}(T, f)) = \texttt{bind}(F_{inl}(t), k'(g))$.*

Доказательство. Снова индукция по структуре терма T, случай переменной и аппликации тривиален, рассмотрим случай абстракции λM.

Левая часть вычислится в:

$$\lambda F_{f'(inl)}(\texttt{bind}(M, k'(f)))$$

Правая в:

$$\lambda \texttt{bind}(F_{f'(inl)}(M), k'(k'(f)))$$

По лемме 2 имеем, что $\texttt{bind}(F_{f'(inl)}(M), k'(k'(f))) = \texttt{bind}(M, k'(k'(f)) \circ f'(inl))$. По лемме 3 имеем $F_{f'(inl)}(\texttt{bind}(M, k'(f))) = \texttt{bind}(M, (x \mapsto F_{f'(inl)}(k'(f)(x))))$. Заметим теперь, что стрелки Клейсли $k'(k'(f)) \circ f'(inl)$ и $x \mapsto F_{f'(inl)}(k'(f)(x))$ ведут себя одинаково на всех входах, тогда по симметричности и транзитивности равенства получаем доказательство требуемого утверждения. ■

Утверждение 7. *Для любого $T \in \Lambda_V$, $f : V \to \Lambda_W, g : W \to \Lambda_U$ верно $\texttt{bind}(\texttt{bind}(T, f), g) = \texttt{bind}(T, (x \mapsto \texttt{bind}(f(x), g)))$.*

Доказательство. Индукция по терму T:

1. Рассмотрим случай, когда $T = v$. Тогда левая часть вычисляется в $\texttt{bind}(f(v), g)$, ровно как и правая.

2. Случай для аппликации следует напрямую из индукционной гипотезы.

3. Рассмотрим случай абстракции λM. Посмотрим, во что вычисляется левая часть:

$$\texttt{bind}(\texttt{bind}(\lambda M, f), g) = \texttt{bind}(\lambda \texttt{bind}(M, k'(f)), g) =$$
$$\lambda \texttt{bind}(\texttt{bind}(M, k'(f)), k'(g))$$

где

$$k'(g)(inl(w)) = F_{inl}(g(w))$$
$$k'(g)(inr(*)) = \texttt{return}(inr(*))$$

и

$$k'(f)(inl(v)) = F_{inl}(f(v))$$
$$k'(f)(inr(*)) = \texttt{return}(inr(*))$$

Правая часть, в свою очередь, вычисляется в:

$$\texttt{bind}(\lambda M, (x \mapsto \texttt{bind}(f(x), g))) = \lambda \texttt{bind}(M, k'(x \mapsto \texttt{bind}(f(x), g)))$$

Чтобы воспользоваться индукционной гипотезой, необходимо показать, что $k'(x \mapsto \texttt{bind}(f(x), g)) : V \coprod \{*\} \to \Lambda_{U \coprod \{*\}}$ ведет себя так же как и $x \mapsto \texttt{bind}(k'(f)(x), k'(g))$. Для этого мы просто покажем, что они возвращают одинаковый результат на всех входах.

Рассмотрим два случая, как могут выглядеть входные данные:

(a) $inr(*)$. Обе части вычисляются в $\texttt{return}(inr(*))$.

(b) $inl(v)$. Левая часть вычисляется в:

$$F_{inl}(\texttt{bind}(f(v), g))$$

Правая:

$$\texttt{bind}(F_{inl}(f(v)), f'(g))$$

Воспользовавшись леммой 4 для терма $f(v)$ и g получаем доказательство исходного утверждения. ■

Отметим наконец, что действие функтора мы проинтерпретировали, как переименование переменных. Действие монадического связывания можно, в таком случае, проинтерпретировать как подстановку. Это утверждение не столь очевидно, но если рассмотреть сигнатуру bind и обратить внимание на то, что функцию $k : V \to \Lambda_W$ можно задать в виде списка пар (V, Λ_W), то это соответствие становится куда более явным. Монадные законы, в свою очередь, в точности описывают свойства подстановки, которые мы в явном виде задали в прошлых разделах 2.1 и 2.2.

2.4. Преобразования между представлениями

В этом разделе мы опишем преобразования между представлениями и начнем с преобразования именованных термов в неименованные. Очевидно, что для осуществления этого нам необходимо знать порядок на переменных в терме. Поэтому мы считаем, что кроме самого терма нам дают контекст.

Определение. Контекст Γ – это не содержащий дубликатов список переменных $x_1, \dots x_n$, $x_i \in \mathcal{V}$.

Определение. Терм T определен в контексте Γ тогда и только тогда, когда все свободные переменные терма T присутствуют в Γ. Этот факт традиционно обозначается как $\Gamma \vdash T$.

Итак, преобразование Φ именованных термов в неименованные принимает на вход контекст $\Gamma = x_1, \dots, x_n$, терм $T \in \Lambda$ такой что он определен в контексте Γ и возвращает неименованный терм $T' \in \Lambda_n$. Определяется оно индукцией по структуре терма T:

1. $x_1, \dots, x_n \vdash x_i \mapsto v_{n,n-i}$

2. $x_1, \dots, x_n \vdash MN \mapsto \Phi(x_1, \dots, x_n \vdash M)\ \Phi(x_1, \dots, x_n \vdash N)$

3. $x_1, \dots, x_n \vdash \lambda x.M \mapsto \lambda\Phi(x_1, \dots, x_n, x \vdash M)$

Покажем, что такое преобразование уважает отношение α-эквивалентности, введенное в разделе 2.1.

***Утверждение* 8.** *Пусть* $T_1, T_2 \in \Lambda$, $\Gamma \vdash T_1$, $\Gamma \vdash T_2$ *и* $T_1 =_\alpha T_2$. *Тогда* $\Phi(\Gamma \vdash T_1) = \Phi(\Gamma \vdash T_2)$.

Доказательство. Индукция по термам T_1 и T_2. Так как мы знаем, что они α-эквивалентны, то нам нет необходимости рассматривать всевозможные комбинации термов. Поэтому рассмотрим лишь случаи, когда термы имеют общую структуру(две переменные, две аппликации или две абстракции).

База индукции для случая двух переменных тривиальна, как и случай для двух аппликаций. Рассмотрим случай, когда $T_1 = \lambda x.M$, $T_2 = \lambda y.M'$. Так как они α-эквивалентны, то мы знаем, что $M' =_\alpha M[x \mapsto y]$ и $y \notin FV(M)$. Мы знаем, что $\Gamma \vdash \lambda x.M$ и $\Gamma \vdash \lambda y.M'$, следовательно $\Gamma, x \vdash M$ и $\Gamma, y \vdash M'$. Кроме того, так как $y \notin FV(M)$, то $\Gamma, y \vdash M[x \mapsto y]$. По индукционной гипотезе мы знаем, что $M' =_\alpha M[x \mapsto y]$, а так как они определены в одинаковых контекстах, то и $\Phi(\Gamma, y \vdash M') = \Phi(\Gamma, y \vdash M[x \mapsto y])$. Осталось заметить, что $\Phi(\Gamma, y \vdash M[x \mapsto y]) = \Phi(\Gamma, x \vdash M)$ и получить доказательство исходного утверждения. ■

Легко сконструировать преобразование в обратную сторону – из неименованных термов в именованные. Оно принимает на вход терм $T' \in \Lambda_n$ и возвращает пару из контекста Γ и терма $T \in \Lambda$, определенного в нем. Рассмотрим три случая, необходимые для того, чтобы рекурсивно задать это преобразование, назовем его Ψ.

1. В случае, когда $T' = v_{n,i}$ мы можем просто сгенерировать n именованных переменных $\Gamma = x_1, \dots x_n$ и в качестве результата вернуть пару из контекста Γ и x_{n-i}. То есть:
$$v_{n,i} \mapsto x_1, \dots x_n \vdash x_{n-i}$$

2. В случае аппликации $M\ N$ все еще проще. Так как она определена в том же контексте, что и оба аппл985 ликанта, то нам достаточно пары рекурсивных вызовов и мы можем взять любой контекст в окончательный результат. То есть:
$$M\ N \mapsto \pi_1(\Psi(M)) \vdash \pi_2(\Psi(M))\ \pi_2(\Psi(N))$$
Здесь и далее π_1 и π_2 – первая и вторая проекция для пар соответственно.

3. Для абстракции λT действуем примерно так же. Вызываемся рекурсивно от T и получаем терм, который определен в расширенном на одну переменную контексте. Так как контекст расширяется путем добавления переменно в конец, то мы точно знаем, по какой переменной абстрагироваться:

$$\lambda T \mapsto take(n, \pi_1(\Psi(T))) \vdash \lambda last(\pi_1(\Psi(T)))\pi_2(\Psi(T))$$

Здесь $take(n, xs)$ – операция, берущая первые n элементов из списка xs, а $last(xs)$ – операция, возвращающая последний элемент в списке.

Легко заметить, что эти два преобразования взаимно-обратны. Действительно, индукция по структуре терма настолько прямолинейна, что мы не будем приводить ее здесь.

Аналогичным образом легко сконструировать преобразования между неименованными и монадическими термами. Так же, довольно легко показать, что они взаимно-обратны.

3. Особенности реализации

Здесь будут описаны особенности реализации. Мы опишем язык, с помощью которого формализовывались все представления. Для каждого представления мы опишем тонкие моменты, с которыми пришлось иметь дело в ходе выполнения работы.

3.1. Описание языка Vclang

Хороший, годный язык например. Здесь будет его описание, постараемся не сильно залезать в детали. Можно сказать, что в нем по-другому устроено равенство, в отличии от агды

Заключение

Здесь будет заключение. Мы сделаем выводы, и скажем какие-то последние слова.

Список литературы

[1] Barendregt HP. Lambda calculi with types, Handbook of logic in computer science (vol. 2): background: computational structures. — 1993.

[2] Bird Richard S, Paterson Ross. De Bruijn notation as a nested datatype // Journal of functional programming. — 1999. — Vol. 9, no. 01. — P. 77–91.

[3] Church Alonzo. An unsolvable problem of elementary number theory // American journal of mathematics. — 1936. — Vol. 58, no. 2. — P. 345–363.

[4] Coquand Thierry, Huet Gérard. The calculus of constructions // Information and computation. — 1988. — Vol. 76, no. 2-3. — P. 95–120.

[5] Curry Haskell B. Grundlagen der kombinatorischen Logik // American journal of mathematics. — 1930. — Vol. 52, no. 4. — P. 789–834.

[6] De Bruijn Nicolaas Govert. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem // Indagationes Mathematicae (Proceedings) / Elsevier. — Vol. 75. — 1972. — P. 381–392.

[7] Howard William A. The formulae-as-types notion of construction // To HB Curry: essays on combinatory logic, lambda calculus and formalism. — 1980. — Vol. 44. — P. 479–490.

[8] Kmett Edward A. "bound: Making de Bruijn Succ Less". — Access mode: `https://goo.gl/TKhGDn` (online; accessed: 2017-05-27).

[9] Landin Peter J. The mechanical evaluation of expressions // The Computer Journal. — 1964. — Vol. 6, no. 4. — P. 308–320.

[10] Martin-Löf Per. An intuitionistic theory of types: Predicative part // Studies in Logic and the Foundations of Mathematics. — 1975. — Vol. 80. — P. 73–118.

[11] Norell Ulf. Towards a practical programming language based on dependent type theory. — Citeseer, 2007. — Vol. 32.

[12] Sakaguchi Kazuhiko. A Formalization of Typed and Untyped λ-Calculi in SSReflect-Coq and Agda2. — Access mode: `https://goo.gl/WtcdZE` (online; accessed: 2017-05-23).

[13] Schönfinkel Moses. Über die Bausteine der mathematischen Logik // Mathematische Annalen. — 1924. — Vol. 92, no. 3. — P. 305–316.

[14] Wiki Haskell. Arrow tutorial. — Access mode: `https://goo.gl/bIiCsy` (online; accessed: 2017-05-29).