Санкт-Петербургский академический университет —
научно-образовательный центр нанотехнологий РАН
(Академический университет)
Кафедра математических и информационных технологий

# О вычислительной эквивалентности высших индуктивных типов

К. В. Елагин
Научный руководитель: В. И. Исаев, СПбАУ

Санкт-Петербург
2016

## РЕФЕРАТ

Под теорией типов понимают формальную систему, оперирующую понятиями «терм» и «тип» и позволяющую выводить суждения вида «терм $x$ имеет тип $A$». На данный момент известны и активно изучаются несколько вариантов теории типов: теория типов Мартин-Лёфа, гомотопическая теория типов, кубическая теория типов, опетопическая теория типов.

Соответствие между интуиционистской логикой и теорией типов Мартин-Лёфа, известное как *соответствие Карри — Говарда*, позволяет использовать последнюю, а также иные основанные на ней теории типов, в качестве формальной логики для записи доказательств произвольных утверждений и последующей их алгоритмической проверки.

В настоящей работе изучается возможность построения доказательств гомотопической эквивалентности широкого класса пространств в теории типов с интервалом и условиями. Теория типов с интервалом и условиями представляет собой новую разновидность гомотопической теории типов, и обладает рядом преимуществ по сравнению со стандартной гомотопической теорией типов. Некоторые её преимущества, в частности, продемонстрированы в настоящей работе.

Основным результатом работы является техника доказательства эквивалентности типов, основанная на использовании аналогов заполнителей Кана в кубических множествах, позволяющая получать доказательства, имеющие вычислительный смысл.

Диссертационная работа состоит из введения, четырех глав, общих выводов и списка используемой литературы. Работа изложена на 36 страницах, содержит 6 рисунков, 11 листингов, список литературы из 22 наименований.

## ОГЛАВЛЕНИЕ

Стр.

# ВВЕДЕНИЕ

**Теория типов** — собирательное название семейства формальных систем, оперирующих понятиями «терм» и «тип», позволяющих выводить суждения вида «терм $x$ имеет тип $A$».

Считается, что первым сопоставлять термам типы предложил Рассел[1] после обнаружения им парадокса Рассела. Некоторые из этих формальных систем рассматриваются в качестве альтернативных оснований математики.

Изучаемые в настоящее время теории типов обычно основаны на интуиционистской теории типов Мартин-Лёфа. Она представляет интерес в силу существования соответствия между типами в теории типов и утверждениями в логике, что позволяет использовать теорию типов как систему формализации доказательств.

Разновидность теории типов, известная как *гомотопическая теория типов*, появилась из идеи рассматривать типы как объекты, к которым применима интуиция из абстрактной теории гомотопий. Оставаясь подходящей системой для формализации доказательств, гомотопическая теория типов позволяет более естественным образом работать с топологическими понятиями, и потому является удобным инструментом для формализации утверждений из алгебраической топологии.

При этом нередко результат, который выглядит простым и интуитивно понятным, оказывается довольно трудно формализовать (например, «тор гомотопически эквивалентен произведению двух окружностей» или «граница квадрата гомотопически эквивалентна окружности»). В настоящей работе изучаются подобные утверждения об эквивалентности, а также предлагается техника доказательства эквивалентности широкого класса пространств.

ГЛАВА 1

# ТЕОРИЯ ТИПОВ

## 1.1 Теория типов Мартин-Лёфа

**Теория типов Мартин-Лёфа**[2] (также известная как **интуиционистская теория типов**, или **конструктивная теория типов**) была предложена Мартин-Лёфом в качестве альтернативного теории множеств и классической логике основания математики, базирующегося на принципах конструктивизма.

Основная идея интуиционистской теории типов состоит в том, что можно отождествить утверждение и тип его доказательств, — это соответствие известно как *соответствие Карри — Говарда*[3].

В классической теории типов обычно рассматриваются следующие типы:

- Зависимые функции (П-тип).
- Зависимые пары ($\Sigma$-тип).
- Конечные типы (пустой, состоящий из одного элемента, …).
- Индуктивные типы (списки, натуральные числа, …).
- Вселенные типов.
- Типы равенства.

Последние типы — типы равенства — особенно интересны, поскольку интерпретация равенства в теории типов отличается от принятой в теории множеств. Два элемента множества равны если и только если они — один и тот же элемент. В теории типов же тип равенства является таким же типом, как и все другие, и, в частности, необязательно требовать, чтобы этот тип состоял лишь из одного элемента. Иными словами, два элемента типа A могут быть «равны несколькими разными способами».

Более того, поскольку тип равенства является типом, для него тоже существует тип равенства, элементы которого представляют собой равенства между равенствами.

Удобный способ думать об этом — представлять себе типы как топологические пространства, элементы типа — как точки в этом пространстве, а элементы типа равенства между двумя элементами — как пути между точками в пространстве.

## 1.2 Гомотопическая теория типов

**Гомотопическая теория типов**[4] — разновидность теории типов, основанная на естественной интерпретации типов равенства как пространств путей в теории гомотопий.

Такая интерпретация позволяет придать смысл стандартным свойствам равенства: рефлексивность гласит, что существует тривиальный путь из точки в неё саму; симметричность соответствует обращению пути; транзитивность — конкатенации путей.

Но можно заметить, что при этом не выполняются некоторые естественные соотношения, к примеру, конкатенация пути с обратным к нему — это не то же самое, что тривиальный путь. На помощь приходит упомянутый ранее тип равенств между равенствами. Конкретно в этом случае оказывается, что тождественный путь, действительно, равен конкатенации любого пути с обратным к нему, в смысле населенности типа путей между этими путями.

Тот факт, что гомотопическая теория типов использует те же базовые понятия, что и теория гомотопий, позволяет определять типы, соответствующие объектам, рассматриваемым в теории гомотопий, например: гомотопия, гомотопическая эквивалентность, расслоение, надстройка и так далее.

К сожалению, у базовой гомотопической теории типов есть существенный недостаток, и кроется он в том, как устроены пути между типами во вселенной. Естественно было бы ожидать, что доказательства равенства двух типов находятся во взаимно-однозначном соответствии с доказательствами гомотопической эквивалентности этих типов. Общепринятое решение заключается в том, чтобы добавить в теорию **аксиому унивалентности**, предложенную Воеводским[5, 6]. Аксиома позволяет получить путь между двумя типами из их гомотопической эквивалентности.

Однако, добавление *аксиомы* лишает систему хороших вычислительных свойств, унаследованных от теории типов Мартин-Лёфа, поскольку подходящей вычислительной интерпретации аксиомы унивалентности на данный момент неизвестно. Важное направление текущих исследований в области теории типов — поиск модели гомотопической теории типов, в которой аксиоме унивалентности можно было бы придать вычислительный смысл.

## 1.3 Кубическая теория типов

В 2014-м году вышла статья[7], авторы которой сделали важный шаг в сторону придания аксиоме унивалентности вычислительного смысла. В ней построена модель гомотопической теории типов в категории кубических множеств посредством задания на ней структуры *категории с семействами*[8, 9, 10]. На основе этой модели предложена разновидность теории типов, отличающаяся от обычной тем, что в неё были добавлены некоторые примитивы, имеющие интерпретацию в кубических множествах. Также была представлена реализация тайп-чекера для описанной теории[11].

В начале 2015-го года похожую идею применили к стандартной гомотопической теории типов, добавив в неё высшие типы, соответствующие кубам[12]. Все эти типы можно было определить и раньше, но добавление их в качестве базовых примитивов позволило формализовать некоторые примеры, считавшиеся до этого трудными.

Весной 2015-го года была представлена новая версия теории на основе кубических множеств, названная *кубической теорией типов*[13], также вместе с реализацией тайп-чекера[14]. Главное отличие этой теории типов заключается в том, что в неё добавили понятие *имени*, *формулы* с именами и возможность *применять* пути к формулам.

Использованная в данной работе *теория типов с интервалом и условиями* (описанная более подробно в следующей главе) представляет собой, скорее, разновидность стандартной гомотопической теории типов, поскольку не включает в себя принципиально новых понятий вроде имен и формул.

Тем не менее, она тоже позволяет оперировать кубами (хотя кубы в ней представлены менее явно), и доказательства, записанные в теории типов с интервалом и условиями, очень похожи на аналогичные доказательства в кубической теории типов.

Кроме того, описанные в третьей главе настоящей работы кубические заполнители Кана упоминаются также и в [13], а в мартовской презентации Ликаты[15] приведено рекурсивное определение заполнителей в терминах кубической теории типов, по своей сути совпадающее с предложенным в настоящей работе определением в терминах теории типов с интервалом и условиями.

ГЛАВА 2

# ТЕОРИЯ ТИПОВ С ИНТЕРВАЛОМ И УСЛОВИЯМИ

## 2.1 Высшие индуктивные типы

В теорию типов можно добавить примитивы, позволяющие определять *высшие индуктивные типы*. В отличие от обычных индуктивных типов, при определении высшего индуктивного типа указывают «конструкторы», порождающие не только *точки* в типе, но и пути.

Высшие индуктивные типы представляют интерес, поскольку позволяют определять аналоги более сложных топологических пространств, а именно, представляющих собой нечто большее, чем дискретное множество точек.

При этом, при определении типа, можно, как и раньше, перечислить его точки, затем пути между этими точками, затем пути между путями, и так далее. Иными словами, появляется возможность добавлять в тип пути любой размерности.

Таким образом можно определить, например:

- Интервал, как тип, состоящий из двух точек (концов интервала) и пути между ними.
- Окружность, как тип, состоящий из одной точки и петли — пути из неё в неё же саму.
- $2$-сферу, как тип, состоящий из одной точки и $2$-пути между тождественным путём в этой точке и этим же тождественным путем.
- $n$-сферу, должным образом обобщив определение $2$-сферы.

Кроме того, появляется возможность перенести в теорию типов некоторые топологические конструкции:

- Цилиндр с типом A в основании это произведение A и интервала.
- Надстройка (suspension) над типом A это тип, состоящий из двух точек (полюсов), копии каждой точки из типа A, а также путей из обоих полюсов во все точки, скопированные из A. Иными словами, это цилиндр над A, у которого верхнюю поверхность сжали в одну точку и то же самое сделали с нижней поверхностью.

Легко видеть, что с помощью индуктивных типов можно представить любой CW-комплекс (клеточный комплекс, в котором границы $n$-дисков «приклеиваются» только к дискам меньшей размерности). Достаточно в каче-

стве $n$-дисков использовать $n$-пути, предварительно разбив образ граничного (приклеивающего) отображения на два $(n-1)$-пути.

## 2.2 Интервал и типы с условиями

В данной работе использовалась теория типов, отличающаяся от описанной ранее, но позволяющая работать с теми же типами, что и стандартная теория типов (включая высшие индуктивные типы). Формализованные утверждения записаны на языке hoq[16], реализующем эту теорию типов. Подробное описание языка можно найти в статье, посвященной теории типов с интервалом и условиями[17].

Основное отличие этой теории типов заключается в том, как определяется тип путей. А именно, вместо примитивного типа путей со своими правилами типизации, конструктором и элиминатором, в теорию добавляется тип интервала `I` с конструкторами `left : I` и `right : I`, а также элиминатором `coe`:

```
coe : (X : I -> Type) -> (i j : I) -> X i -> X j
```

Теперь путь из точки `a` в точку `b` в типе `X` можно определить как отображение из интервала в `X`, удовлетворяющее двум условиям: левую точку интервала оно отображает в `a`, а правую — в `b`.

Для описания ограничений такого рода теория поддерживает два расширения:

- типы данных (`data`) с условиями,
- записи (`record`) с условиями.

Условия позволяют *вычислительно* отождествить некоторые точки в типе. Таким образом можно определить тип путей из `a` в `b` как запись, состоящую из отображения с необходимыми условиями (листинг 2.1). Это определение можно обобщить до типа *зависимых* путей (листинг 2.2). В [17] показано, что для определенного таким образом типа путей выполняется аксиома J.

```
record Path' (A : Type) (a : A) (b : A) where
    constructor path'
    at : (i : I) -> A
  with
    at left  = a
    at right = b
```

Листинг 2.1 — Тип путей

```
record Path (A : I -> Type) (a : A left) (b : A right) where
    constructor path
    at : (i : I) -> A i
  with
    at left  = a
    at right = b
```

Листинг 2.2 — Тип зависимых путей

### 2.3 Произведение высших типов

Легко видеть, что с помощью интервала и типов данных с условиями можно определить любой высший индуктивный тип. Но теперь, когда о путях можно думать как об отображениях из интервала, а о высших путях, соответственно, как об отображениях из кубов (произведений интервала с самим собой несколько раз), типы, содержащие высшие пути, описываются более естественным образом.

К примеру, 2-сферу можно определить как тип, состоящий из выделенной точки и квадрата — произведения `I` на `I` — с дополнительным условием, утверждающим, что любая точка на границе квадрата совпадает с выделенной, как в листинге 2.3.

```
data S2 = base | loop2 I I
  with
    loop left  j = base
    loop right j = base
    loop i left  = base
    loop i right = base
```

Листинг 2.3 — Определение 2-сферы

Аналогично можно определить тип, являющийся аналогом любого конечного CW-комплекса, используя кубы в качестве дисков и накладывая условия, соответствующие склеивающим отображениям.

По сравнению со стандартной гомотопической теорией типов, мы получаем возможность работать с клетками в комплексе более удобным образом: у нас теперь есть не только элиминатор типа путей, но и возможность осуществлять паттерн-матчинг по дискам, извлекая «координаты» данной точки на диске.

В алгебраической топологии верно следующее утверждение, описывающее произведение двух CW-комплексов (доказательство можно найти в приложении A книги Хэтчера по алгебраической топологии[18]): если даны два CW-комплекса, и для любого $i$ первый содержит ровно $a_i$ клеток размерности $i$, а второй — $b_i$ клеток размерности $i$, то их произведение состоит из $a_i a_j$ клеток размерности $i + j$ для всех $i$ и $j$.

Мы можем доказать аналогичное утверждение в теории типов. Будем говорить, что тип соответствует конечному CW-комплексу, если:

- все его конструкторы имеют вид `cons` **`I`** … **`I`**,
- левые части всех уравнений в условиях представляют собой применение конструктора к **`left`** или **`right`** по одной из координат,
- правые части всех уравнений в условиях представляют собой применение конструктора к произвольным выражениям, причем размерность этого конструктора строго меньше размерности конструктора, стоящего в левой части уравнения,
- на каждый конструктор размерности $n$ приходятся ровно $2n$ условий с ним в левой части, соответственно, подставляющие **`left`** или **`right`** по очереди в каждую координату.

Пусть даны два типа, соответствующие конечным CW-комплексам, и при этом первый тип для любого $i$ содержит ровно $a_i$ конструкторов размерности $i$, а второй — $b_i$ конструкторов размерности $i$. Тогда их произведение гомотопически эквивалентно типу, содержащему $a_i a_j$ конструкторов размерности $i + j$.

Для доказательства этого утверждения явно построим тип, эквивалентный произведению данных, а также отображения, задающие эту эквивалентность. Итак, рассмотрим $s_k$ — некоторый конструктор первого типа размер-

ности $m$ и $t_l$ — некоторый конструктор второго типа размерности $n$. Тогда добавим в тип, который строим, конструктор $s_k t_l$ размерности $m + n$ и отобразим пару ($s_k\ i_1 \dots i_m$, $t_l\ j_1 \dots j_n$) в $s_k t_l\ i_1 \dots i_m j_1 \dots j_n$.

Чтобы такое определение отображения было корректным, необходимо учесть условия, наложенные на исходные типы. Для каждого конструктора первого типа $s_k$ рассмотрим все условия из первого типа, содержащие $s_k$ в левой части (затем то же самое надо будет сделать для всех конструкторов второго типа). Пусть некоторое условие имеет вид $s_k\ i_1 \dots \text{left} \dots i_m \equiv s_q\ v_1 \dots v_{m'}$ (аналогично для right). Когда отображение действует на пару, на первом месте в которой стоит выражение из левой части уравнения, должно получаться то же самое, что и когда оно действует на пару, на первом месте в которой стоит выражение из правой части. Это легко гарантировать, добавив условие $s_k t_l\ i_1 \dots \text{left} \dots i_m j_1 \dots j_n \equiv s_q t_l\ v_1 \dots v_{m'} j_1 \dots j_n$ для всех конструкторов второго типа $t_l$.

Легко убедиться, что получившийся тип также соответствует конечному CW-комплексу, поскольку удовлетворяет всем условиям просто по построению.

Второе же отображение всякий элемент $s_k t_l\ i_1 \dots i_m j_1 \dots j_n$ отображает в пару ($s_k\ i_1 \dots i_m$, $t_l\ j_1 \dots j_n$). Легко видеть, что это в точности обратное отображение к тому, которое мы построили. Из этого, в частности, следует, что оно корректно, поскольку в том типе, из которого оно действует, были склеены в точности те точки, которые оно отображает в одинаковые. Также из этого следует, что мы построили эквивалентность.

В качестве примера докажем, что тор гомотопически эквивалентен произведению двух окружностей. В стандартной гомотопической теории типов не известно короткого доказательства этого простого утверждения, но в теории типов с интервалом и условиями мы можем воспользоваться доказанным утверждением. Действительно, тор, как конечный CW-комплекс, состоит из одной 0-мерной ячейки, двух одномерных ячеек и одной двумерной ячейки, в качестве которой мы используем квадрат (листинг 2.4), а окружность — из одной 0-мерной ячейки и одной одномерной (листинг 2.5). Построив предложенным методом тип, эквивалентный произведению окружностей, замечаем, что он в точности является тором. Отображения, задающие эквивалентность, приведены в листинге 2.6.

```
data Torus = base-base | loop-base I | base-loop I | loop-loop I I
  with
    loop-base left  = base-base
    loop-base right = base-base
    base-loop left  = base-base
    base-loop right = base-base
    loop-loop i left  = loop-base i
    loop-loop i right = loop-base i
    loop-loop left  j = base-loop j
    loop-loop right j = base-loop j
```

Листинг 2.4 — Определение тора

```
data S1 = base | loop I
  with
    loop left  = base
    loop right = base

data S1xS1 = pair S1 S1
```

Листинг 2.5 — Определение `S1` и `S1×S1`

Таким образом, теория типов с интервалом и условиями в некоторых случаях оказывается удобнее стандартной теории с высшими индуктивными типами. Поскольку, как мы видели, любой высший индуктивный тип может быть определен в теории с интервалом и условиями, эта теория «не слабее» стандартной. Существуют ли типы, которые можно определить в новой теории, но нельзя в старой, на данный момент неизвестно.

```
f : S1xS1 -> Torus
f (pair base base) = base-base
f (pair (loop i) base) = loop-base i
f (pair base (loop j)) = base-loop j
f (pair (loop i) (loop j)) = loop-loop i j

g : Torus -> S1xS1
g base-base = pair base base
g (loop-base i) = pair (loop i) base
g (base-loop j) = pair base (loop j)
g (loop-loop i j) = pair (loop i) (loop j)

f-then-g : (x : S1xS1) -> x = g (f x)
f-then-g (pair base base) = idp
f-then-g (pair (loop _) base) = idp
f-then-g (pair base (loop _)) = idp
f-then-g (pair (loop _) (loop _)) = idp

g-then-f : (x : Torus) -> x = f (g x)
g-then-f base-base = idp
g-then-f (loop-base _) = idp
g-then-f (base-loop _) = idp
g-then-f (loop-loop _ _) = idp
```

Листинг 2.6 — Эквивалентность тора и произведения окружностей

ГЛАВА 3

# КУБИЧЕСКИЕ ЗАПОЛНИТЕЛИ КАНА

## 3.1 Кубические множества

Определим кубические множества так же, как Кокванд[7].

Для этого зафиксируем множество *имен* или *символов*. Рассмотрим категорию имен и подстановок, объектами которой являются все возможные подмножества множества имен, которые мы будем обозначать буквами $I, J, K, \ldots$, а морфизмами $I \to J$ — отображения $I \to J \cup \{0, 1\}$, такие что, если $f(i)$ и $f(j)$ лежат в $J$, то $f(i) = f(j) \Leftrightarrow i = j$. Для таких морфизмов несложно определить операцию композиции, а также тождественный морфизм для любого объекта.

О морфизмах можно думать как о подстановках с переименованием. Единственные значения, которые можно подставлять, это $0$ и $1$, а при переименовании двум разным переменным нельзя присваивать одно и то же новое имя.

В частности, для каждого $x \in I$ есть две подстановки $I \to I \setminus \{x\}$, отображающие $x$ в $0$ или $1$ соответственно, а остальные имена — в самих себя. Эти подстановки будем называть *отображениями граней*. Для каждого набора направлений размера $n$ есть $2n$ таких отображений.

**Кубическое множество** — это предпучок на категории, двойственной к категории имен и подстановок, или, иными словами, функтор из категории имен и подстановок в категорию множеств.

Пусть $X$ — кубическое множество. Если представить, что множество имен состоит из названий осей координат, то объект категории имен и подстановок — это какой-то набор направлений. Тогда вложение Йонеды каждому набору направлений $I$ сопоставляет некоторое кубическое множество, о котором можно думать как о формальном представлении $[0, 1]^I$, а $X(I)$ это множество, элементами которого являются кубы или «непрерывные» отображения (соответствующие естественным преобразованиям) $[0, 1]^I \to X$.

Альтернативный взгляд на кубические множества предложил Питтс[19], показав, что категория кубических множества эквивалентна категории номинальных множеств[20, 21] с операцией 01-подстановки.

### 3.2 Заполнение Кана

Для интерпретации гомотопической теории типов в кубических множествах[7] достаточно, чтобы в них выполнялось *условие Кана*, которое можно сформулировать следующим образом.

Для начала, введем понятие *открытого короба* (*open box*, *cubical horn*) — аналога *рога* (*horn*) в симплициальных множествах. Для этого зафиксируем подмножество направлений $J$, а также какое-либо направление $x \in I \setminus J$. Положим $O^\beta(J,x) = \{(y,b) \mid y \in J, b \in \{0,1\}\} \cup \{(x, 1-\beta)\}$ (при $\beta \in \{0,1\}$). Интуитивно, множество $O^0(J,x)$ описывает, какие грани должны присутствовать в коробе с $|J|$-мерной поверхностью, открытом в направлении $x$ со стороны нуля, а $O^1(J,x)$ — со стороны единицы. Тогда семейство граней $u$ называется **открытым $|J|$-коробом**, если:

а) Короб «закрыт» с двух сторон по всем направлениям из $J$ и с одной стороны в направлении $x$: $u = \{u_{yb} \in X(I-y) \mid (y,b) \in O^\beta(J,x)\}$.

б) Соседние грани короба «склеены правильным образом»: $\forall\,(y,b),(z,c) \in O^\beta(J,x)\ .\ z \neq y \Rightarrow u_{yb}(x \coloneqq c) = u_{zc}(y \coloneqq b)$.

**Условие Кана** гласит, что любой открытый короб может быть «заполнен», а именно должны быть даны операции $X{\uparrow}u$ и $X{\downarrow}u$, заполняющие короб $u$, открытый, соответственно, со стороны нуля и со стороны единицы:

- Если $u$ — короб, открытый со стороны нуля, то $X{\uparrow}u \in X(I)$ и $\forall\,(y,b) \in O^0(J,x)\ .\ (X{\uparrow}u)(y \coloneqq b) = u_{yb}$.
- Если $u$ — короб, открытый со стороны единицы, то $X{\downarrow}u \in X(I)$ и $\forall\,(y,b) \in O^1(J,x)\ .\ (X{\downarrow}u)(y \coloneqq b) = u_{yb}$.

Важное наблюдение состоит в том, что результатом заполнения короба, составленного из $n$-мерных ячеек, будет новая грань, тоже являющаяся $n$-мерной ячейкой, а также $(n+1)$-мерная ячейка, заполняющая образовавшийся промежуток. Топологически, мы получили $(n+1)$-путь между двумя $n$-путями: образованным гранями короба и новой гранью. Основная идея состоит в том, чтобы перенести эту конструкцию внутрь теории типов.

### 3.3 Заполнители в теории типов

В теории типов естественно в качестве $n$-мерных ячеек рассматривать $n$-пути. Всегда ли можно заполнить короб, составленный из $n$-путей? Оказывается, что всегда.

Начнем с простейшего случая, с заполнения $0$-короба. По определению, $0$-короб состоит из одной точки:

```
record box-0 (X : I -> Type) where
    constructor box-0
    l : X left
 -- r is missing
```

Заполнить его — значит провести $1$-путь (в единственном имеющемся направлении), соединяющий точку `l` из `X left` с некоторой точкой из `X right`:

```
fill-1 : (X : I -> Type)
      -> box-0 X
      -> (i1 : I) -> X i1
```

Условие, что проведенный путь начинается в точке `l`, эквивалентно тому, что следующее определение проходит проверку типов:

```
cond-1-l : (X : I -> Type)
        -> (l : X left)
        -> fill-1 X (box-0 l) left = l
cond-1-l X l = idp
```

Заполнитель для $0$-короба — не что иное, как `coe`:

```
fill-1 X b i1 = coe X left (b.l) i1
```

Каждый следующий заполнитель выражается через предыдущий. Например, заполнитель для $1$-короба приведен в листинге 3.1.

В общем случае, имеются $(n+1)$ направление $i_1, i_2, \ldots, i_{n+1}$ и, соответственно, зависимый тип `X : I -> I -> ... -> I -> Type`. Идея состоит в том, чтобы перейти к пространству путей в каком-либо выбранном направлении, тем самым уменьшив размерность рассматриваемого пространства на единицу, и, по индукции, взять заполнитель в полученном пространстве меньшей размерности.

При этом, есть два выделенных направления: то, в котором отсутствует грань, и то, которое будет «схлопнуто» переходом к путям в этом направлении. Чтобы термы имели простой вид, важно правильным образом выбрать эти два направления. Удобнее всего оказывается схлопывать последнее направление ($i_{n+1}$), поскольку в этом случае первые $n$ направлений остаются нетронутыми, что хорошо для индукционного вызова; соответственно, отсутствовать грань должна в направлении $i_1$.

Также, для удобства, необходимо зафиксировать некоторый способ перечисления поверхностей короба. Например, можно перечислять их в порядке возрастания размерности, затем в лексикографическом порядке неизменяющихся координат. Иными словами, каждой поверхности куба можно сопоставить вектор из $\{?, 0, 1\}^{n+1}$ (где число вопросиков показывает размерность поверхности); на этих векторах можно ввести порядок: сначала два вектора сравниваются по числу вопросиков, затем — лексикографически ($? < 0 < 1$); поверхности следует перечислять в соответствии с порядком на сопоставленных им векторах. В примерах кода в названиях переменных вместо `?`, `0`, `1` использованы `_`, `l`, `r`.

Ещё один важный вопрос — в каком направлении проводить пути, являющиеся гранями куба. К примеру, в трехмерном кубе грани представляют собой 2-пути, то есть пути в пространстве путей. Есть несколько вариантов описания такого типа: можно сказать, что это путь вдоль $i_1$ между путями, идущими вдоль $i_2$, а можно — что путь вдоль $i_2$ в пространстве путей вдоль $i_1$. Записывая типы, мы перечисляем координаты в естественном порядке, тем самым гарантируя, что всякий $k$-путь, который будет использован в индуктивном переходе, уже сам по себе является $(k-1)$-путем в пространстве путей вдоль $i_{n+1}$, так что его можно сразу же подставлять в рекурсивный вызов, никак не преобразовывая.

На рисунке 3.1 изображено графически определение 2-короба, а листинг 3.2 содержит схему определения типа `box-n` для произвольного $n$ на языке hoq.

Имея `box-n`, можно записать тип заполнителя в общем случае:

```
fill-n : (X : I -> I -> … -> I -> Type)
      -> box-(n-1) X
      -> (i1 i2 … in : I) -> X i1 i2 … in
```

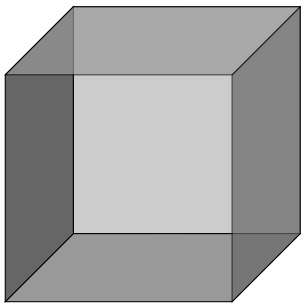

Рис. 3.1 — Открытый $2$-короб (отсутствует передняя грань)

Как было сказано выше, ключевая идея при построении терма, имеющего такой тип, — перейти к пространству путей вдоль направления $i_{n+1}$ и, поскольку его размерность на единицу меньше, по индукции найти заполнитель. Выбранный способ перечисления вершин короба позволяет чисто механически получить необходимый короб меньшей размерности: достаточно перечислять поверхности исходного короба в том же порядке, заменяя последний символ в имени поверхности на подчеркивание (тем самым, повышая её размерность, ведь вершины в новом пространстве это $1$-пути, то есть ребра исходного куба). В листинге 3.3 приведена схема построения заполнителя произвольной размерности.

### 3.4 Применения заполнителей

В терминах заполнителей можно описать многие стандартные операции с путями. Но сначала обратимся к важному вычислительному свойству заполнителя $1$-короба, которое позволит в дальнейшем судить о вычислительных свойствах других операций, а именно: в результате заполнения короба, изображенного на рисунке 3.2, получается вычислительно тот же самый путь:

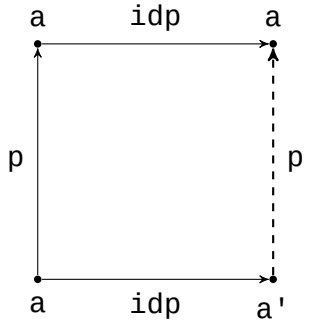


Рис. 3.2 — Короб с двумя тривиальными путями

```
fill-id : {A : Type} {a b : A} (p : a = b)
       -> path (\j ->
            fill-2 (\_ _ -> A) (box-1 a b a b idp idp p) right j)
          = p
fill-id _ _ _ _ = idp
```

Теперь перейдем к простым операциям над путями.

Операция **обращения** пути, доказывающая симметричность равенства, может быть определена через заполнение короба, изображенного на рисунке 3.3, следующим образом:

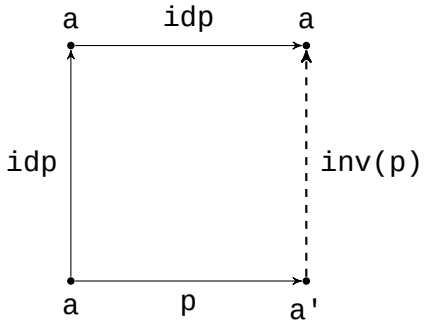


Рис. 3.3 — Обращение пути с помощью заполнителя

```
inv : {A : Type} {a a' : A} -> a = a' -> a' = a
inv A a a' p = path (\j -> fill-2 (\_ _ -> A)
                                  (box-1 a a a' a p idp idp) right j)
```

При этом, она обладает ожидаемым свойством: путь, обратный к тривиальному, также тривиален, причем вычислительно (что следует из упомянутого в начале свойства операции заполнения):

```
idp-inv : {A : Type} {a : A} -> inv (idp' a) = idp' a
idp-inv _ _ = idp
```

**Конкатенацию** путей, являющуюся доказательством транзитивности равенства, можно определить несколькими способами. Чтобы сделать это, нам потребуется вспомогательная функция, конкатенирующая *три* пути. Она всего лишь берет заполнитель, предварительно обратив один из путей:

```
concat3 : {A : Type} {a b c d : A}
       -> a = b -> b = c -> c = d -> a = d
concat3 A a b c d ab bc cd =
  path (\j -> fill-2 (\_ _ -> A) (box-1 b c a d (inv ab) cd bc) right j)
```

С помощью `concat3` конкатенацию *двух* путей можно определить тремя способами — подставив тривиальный путь в качестве первого, второго или третьего.

В первом случае, получим определение, обладающее интересным вычислительным свойством при конкатенации с тривиальным путем справа:

```
concat-left : {A : Type} {a b c : A} -> a = b -> b = c -> a = c
concat-left _ _ _ _ p q = concat3 idp p q

concat-left-idp : {A : Type} {a b : A} -> (p : a = b)
                -> concat-left p idp = p
concat-left-idp _ _ _ _ = idp
```

В третьем случае — при конкатенации с тривиальным слева:

```
concat-right : {A : Type} {a b c : A} -> a = b -> b = c -> a = c
concat-right _ _ _ _ p q = concat3 p q idp

concat-right-idp : {A : Type} {a b : A} -> (p : a = b)
                 -> concat-right idp p = p
concat-right-idp _ _ _ _ = idp
```

Во втором же случае никаких полезных вычислительных свойств не получится, но зато такое определение более симметрично.

```
concat-mid : {A : Type} {a b c : A} -> a = b -> b = c -> a = c
concat-mid _ _ _ _ p q = concat3 p idp q
```

Это определение эквивалентно определению конкатенации двойной индукцией по путям, принятому в Книге HoTT[4]. Два другие же эквивалентны упомянутым в Книге определениям с помощью индукции только по одному из двух имеющихся путей.

Так определить перечисленные операции можно и в общепринятой («книжной») гомотопической теории типов. Более интересный вариант использования заполнителя специфичен для теории с интервалом и условиями.

Пусть у нас есть тип `A`, содержащий, помимо прочего, две точки и путь между ними:

```
data A = a | b | ab I | …
```

```
  with
    ab left  = a
    ab right = b
    …
```

Пусть также есть некоторый тип `B` и два отображения — `f` и `g` — типа `A -> B`. Изобразим пути `pmap f ab` и `pmap g ab` (рисунок 3.4) и заметим, что отображения гомотопны если, и только если найдутся пути из `f a` в `g a`, из `f b` в `g b`, а также если можно заполнить получившийся квадрат. Это верно, поскольку горизонтальные пути, проходящие через квадрат, и есть те самые пути, которые необходимо найти для доказательства гомотопности, — горизонтальный путь на высоте `j` имеет тип `(pmap f ab) @ j = (pmap g ab) @ j`, или, что то же самое по определению `pmap`, `f (ab j) = g (ab j)`. При этом, пути `p` и `q` это те самые пути, которые необходимо использовать для доказательства `f a = g a` и `f b = g b` соответственно, чтобы выполнялись граничные условия.

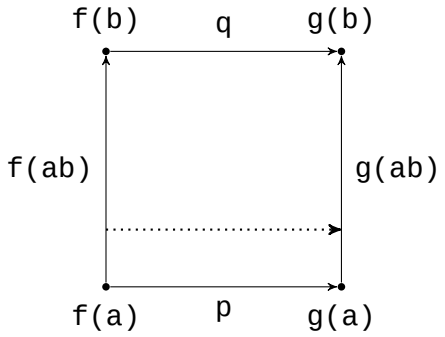


Рис. 3.4 — Квадрат, заполненный гомотопией

Кроме того, если `p` и `q` зафиксированы заранее, а `g` позволено выбрать какой угодно, то в качестве образа `path ab` под действием `g` можно взять путь, получающийся в результате заполнения короба, состоящего из `pmap f ab`, `p` и `q`. Тогда отображения окажутся заведомо гомотопны, и доказательство этого будет легко получить с помощью внутренности заполненного квадрата. Эта идея будет использована в следующей главе.

Ещё одна полезная операция с путями, которую можно определить с помощью заполнителей, называется `ptrim`. Она имеет следующий тип:

```
ptrim : {A : Type} {a a' : A} (p : a = a') (i : I) -> a = p @ i
```

Иными словами, она позволяет «вырезать начальный отрезок» пути, чтобы получить путь из начальной точки в какую-либо промежуточную точку

исходного пути. При этом мы потребуем дополнительно, чтобы эта функция вела себя естественным образом в граничных случаях: если мы вырезаем отрезок длиной в весь путь (`ptrim p right : a = a'`), возвращала исходный путь, а если вырезаем пустой отрезок (`ptrim p left : a = a`), возвращаемый путь был тривиален. По сути, мы ищем зависимый 2-путь между `idp` и `p`. Чтобы удовлетворить обоим условиям, двумерного заполнителя недостаточно, так как он позволяет выполнить только одно условие. Но отсутствующая грань изображенного на рисунке 3.5 короба — как раз то, что нам нужно. В этом коробе фигурирует путь `q`, получающийся в результате взятия двумерного заполнителя, а квадрат, недостающим ребром которого он является, дважды используется в качестве известной грани куба. Таким образом, определить `ptrim` можно в два шага с помощью трехмерного заполнителя.

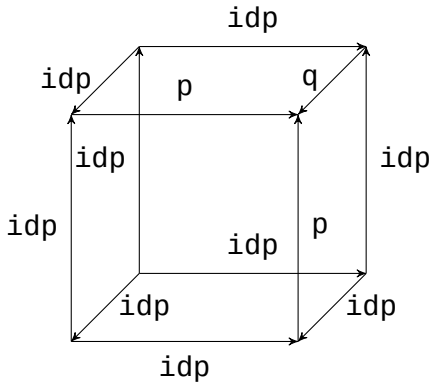


Рис. 3.5 — Короб из определения `ptrim`

Двойственная к `ptrim` функция `ptrim'` имеет следующий тип:

```
ptrim' : {A : Type} {a a' : A} (p : a = a') (i : I) -> p @ i = a'
```

Она, наоборот, позволяет «выкинуть начальный отрезок» пути и, если выкидывается пустой отрезок (`ptrim' p left : a = a'`), возвращает исходный путь, а если выкидывается весь путь (`ptrim' p right : a' = a'`), возвращает тривиальный. Такую функцию можно определить аналогично с помощью трехмерного заполнителя.

```
record box-1 (X : I -> I -> Type) where
    constructor box-1
    -- Vertices
    ll : X left  left
    lr : X left  right
    rl : X right left
    rr : X right right
    -- Edges
    _l : Path (\i1 -> X i1 left ) ll rl
    _r : Path (\i1 -> X i1 right) lr rr
    l_ : Path (\i2 -> X left  i2) ll lr
 -- r_ is missing

fill-2 : (X : I -> I -> Type)
      -> box-1 X
      -> (i1 i2 : I) -> X i1 i2
fill-2 X b i1 i2 = fill-1 (\i1 -> Path (\i2 -> X i1 i2)
                                       (b._l @ i1) (b._r @ i1))
                          (box-1 (b.l_))
                          i1 @ i2

cond-2-_l : (X : I -> I -> Type)
            (b : box-1 X)
         -> (i1 : I) -> fill-2 X b i1 left = (b._l) @ i1
cond-2-_l _ _ _ = idp

cond-2-_r : (X : I -> I -> Type)
            (b : box-1 X)
         -> (i1 : I) -> fill-2 X b i1 right = (b._r) @ i1
cond-2-_r _ _ _ = idp

cond-2-l_ : (X : I -> I -> Type)
            (b : box-1 X)
         -> (i2 : I) -> fill-2 X b left i2 = (b.l_) @ i2
cond-2-l_ _ _ _ = idp
```

Листинг 3.1 — Заполнитель для 2-короба

```
record box-n (X : I -> I -> … -> I -> Type) where
    constructor box-n
    -- Vertices
    …lll : X left  … left  left
    …llr : X left  … left  right
    …lrl : X left  … right left
    …lrr : X left  … right right
    …
    …rll : X right … left  left
    …rlr : X right … left  right
    …rrl : X right … right left
    …rrr : X right … right right

    -- Edges
    …_lll : Path (\i1 -> X i1 left  … left  left ) …llll …rlll
    …_llr : Path (\i1 -> X i1 left  … left  right) …lllr …rllr
    …_lrl : Path (\i1 -> X i1 left  … right left ) …llrl …rlrl
    …_lrr : Path (\i1 -> X i1 left  … right right) …llrr …rlrr
    …
    …_rll : Path (\i1 -> X i1 right … left  left ) …lrll …rrll
    …_rlr : Path (\i1 -> X i1 right … left  right) …lrlr …rrlr
    …_rrl : Path (\i1 -> X i1 right … right left ) …lrrl …rrrl
    …_rrr : Path (\i1 -> X i1 right … right right) …lrrr …rrrr

    …

    …rrl_ : Path (\in+1 -> X right right in+1) …rrll …rrlr
    …rrr_ : Path (\in+1 -> X right right in+1) …rrrl …rrrr

    -- 2-dimensional surfaces
    …__ll : Path (\i1 -> Path (\i2 -> X i1 i2 left … left)
                                (…_lll @ i1) (…_rll @ i1))
                   …l_ll …r_ll

    …

    -- n-dimensional surfaces
    …
    …l__ : Path (\i2 -> Path (\i3 -> … Path (\in+1 ->
              X left i2 … in+1)…))
                   …
                   …ll__ …lr__
 -- …r__ is missing
```

Листинг 3.2 — Схема определения открытого $n$-короба

```
fill-n : (X : I -> I -> … -> I -> Type)
      -> box-n X
      -> (i1 i2 … in+1 : I) -> X i1 i2 … in+1
fill-n X b i1 i2 … in+1 =
  fill-n-1 (\i1 … in -> Path (\in+1 -> X i1 … in in+1)
                             ((b.…__l) @ i1 @ … @ in)
                             ((b.…__r) @ i1 @ … @ in))
         (box-n-1 (b.…lll_) (b.…llr_) (b.…lrl_) (b.…lrr_)
                  …
                  (b._
                  (b.…l__))
         i1 … in @ in+1
```

Листинг 3.3 — Схема заполнения открытого $n$-короба

ГЛАВА 4

# ДОКАЗАТЕЛЬСТВА ЭКВИВАЛЕНТНОСТИ

## 4.1 Эквивалентные определения

Часто одно и то же понятие можно определить несколькими способами. При этом, в различных ситуациях удобнее оказывается использовать разные определения. Таким образом, возникает задача доказательства эквивалентности определений или, точнее, доказательства гомотопической эквивалентности типов.

В качестве примера, можно привести несколько вариантов определения $n$-сферы. Первое, самое простое определение: $n$-сфера состоит из выделенной точки (`base`) и $n$-мерной поверхности, все точки на границе которой отождествлены с выделенной точкой. Формально:

```
data n-Sphere = base | surface I I … I
  with
    surface left  i2 … in = base
    surface right i2 … in = base
    …
    surface i1 i2 … left  = base
    surface i1 i2 … right = base
```

Это определение часто используется на практике из-за его простоты. Также иногда полезным оказывается определение $n$-сферы как поверхности $(n+1)$-куба. Чтобы записать его формально, необходимо сделать нечто похожее на то, что происходило при определении открытого короба: у типа будут $2(n+1)$ конструкторов, каждый из которых принимает точку на $n$-кубе — эти конструкторы описывают точки на гранях $(n+1)$-куба; также необходимо добавить $2n(n+1)$ (это число поверхностей размерности $n-1$ в $(n+1)$-кубе) условий, говорящих, что грани правильным образом склеены на границах:

```
data n-Cube = surface I I … I

data n-dCube = …l__ n-Cube | …r__ n-Cube
             …
             | …__l n-Cube | …__r n-Cube
  with
    …l__ left  i3 … in = …_l__ left  i3 … in
    …l__ right i3 … in = …_r__ right i3 … in
```

```
    ...
```

Это определение оказывается полезным, поскольку самый простой способ получить $n$-мерную поверхность в рассматриваемой теории — взять произведение интервала с самим собой $n$ раз, и в результате получается куб. Если возникает необходимость построить отображение из или на сферу, иногда оказывается проще построить отображение из или на поверхность куба, поскольку в этом случае отображение может действовать покоординатно.

Приведенные выше «определения» нельзя назвать определениями в полном смысле этого слова, это, скорее, схемы определений, поскольку их требуется выписывать для каждого $n$ по-отдельности. С их помощью можно доказывать утверждения вида «для любого $n$» в мета-теории, но не в самой теории типов.

Но можно записать аналогичные определения по индукции, с использованием *надстройки* (*suspension*) и *гомотопических амальгам* (*homotopy pushout*), что позволит доказывать утверждения для любого $n$ *внутри* теории. Предложенная далее техника оказывается полезной и при доказательстве эквивалентности таких индуктивных определений, но в данной работе они не рассматриваются в качестве примеров, поскольку применение этой техники в случае работы с такими типами оказывается, хоть и продуктивным, но не слишком интересным.

## 4.2 Заполнители в доказательствах эквивалентности

Начнем с простейшего случая, а именно, проверим, что отрезок гомотопически эквивалентен трем отрезкам, склеенным последовательно. Определим эти типы следующим образом:

```
data I1 = u | v | uv I
  with
    uv left  = u
    uv right = v

data I3 = a | b | c | d | bc I | ba I | cd I
  with
    bc left  = b
    bc right = c
```

```
    ba left  = b
    ba right = a
    cd left  = c
    cd right = d
```

Для этого нам потребуется построить отображение `f : I3 -> I1`, отображение `g : I1 -> I3`, а также доказать, что композиция этих двух отображений в любом порядке гомотопна тождественному отображению.

Само по себе это утверждение очевидно, и может быть доказано очень легко, ведь оба типа стягиваемы. Поэтому мы предъявим к эквивалентности дополнительное требование: необходимо, чтобы оба отображения переводили границу в границу, то есть:

```
f a = u
f d = v

g u = a
g v = d
```

Такое требование исключает возможность использования тождественных отображений для построения эквивалентности, а также позволит в будущем применить тот же самый подход в более интересных ситуациях. Например, если мы дополнительно отождествим в обоих типах граничные точки, то получим «окружности», в одном случае, состоящую из одного отрезка, а в другом — из трех. Но если наше доказательство переводило граничные точки в граничные, то оно сработает и теперь.

Приступим к построению отображений. Отображение `f : I3 -> I1` построить несложно. Отобразим центральный отрезок в исходном типе в единственный отрезок в целевом, а остальные два отразим в вырожденные отрезки:

```
f : I3 -> I1
f a      = u
f (ba _) = u
f b      = u
f (bc i) = uv i
f c      = v
f (cd _) = v
f d      = v
```

Каким должно быть отображение в обратную сторону, менее очевидно. Дело в том, что отрезок в `I1` необходимо отобразить в какой-то отрезов в `I3`, но ни один из имеющихся там отрезков не подходит из-за требования про отображение границы. Единственный выход — отображать в «конкатенацию» всех трех отрезков, и это можно сделать с помощью заполнителя:

```
g : I1 -> I3
g u      = a
g (uv i) = fill-2 (\_ _ -> I3)
                  (box-1 b c a d (path ba) (path cd) (path bc))
                  right i
g v      = d
```

Следующее, что надо сделать, это доказать, что композиции гомотопны тождественной функции. Как обсуждалось ранее, в случае, когда одно из отображений определено с помощью заполнения, в поиске необходимых путей помогает сам заполнитель. Мы должны построить терм, имеющий следующий тип:

```
g-then-f : (x : I1) -> x = f (g x)
```

Поскольку обе функции определены в терминах отображения отрезков в отрезки, проще думать о том, что там нужно найти путь между путями `path uv` и `pmap f (pmap g (path uv))`. Как мы помним, функция `g`, по определению, отображает отрезок в конкатенацию трех отрезков, так что этот тип вычислительно равен следующему: `path uv = pmap f (concat3 (inv (path ba)) (path bc) (path cd))`, а если раскрыть по определению `concat3` правую часть равенства, то получим `path uv = pmap f (fill-2 … (path ba) (path cd) (path bc))`.

Теперь можно было бы доказать, что применение функции коммутирует с взятием заполнителя, и, поскольку `pmap f (path ba) = pmap f (path cd) = idp` по определению `f`, получить в правой части `pmap f (path bc)`, что равно `path uv` по определению `f`, но мы поступим иначе.

Вместо этого, мы напрямую построим искомые пути `uv j = f (g (uv j))`. На рисунке 4.1 изображен заполнитель, использованный в определении `g`. Подействуем на все изображенные пути отображением `f` и заметим, что горизонтальные пути, проходящие через заполненный квадрат, это как раз то, что мы ищем. Остается только выписать доказательство:

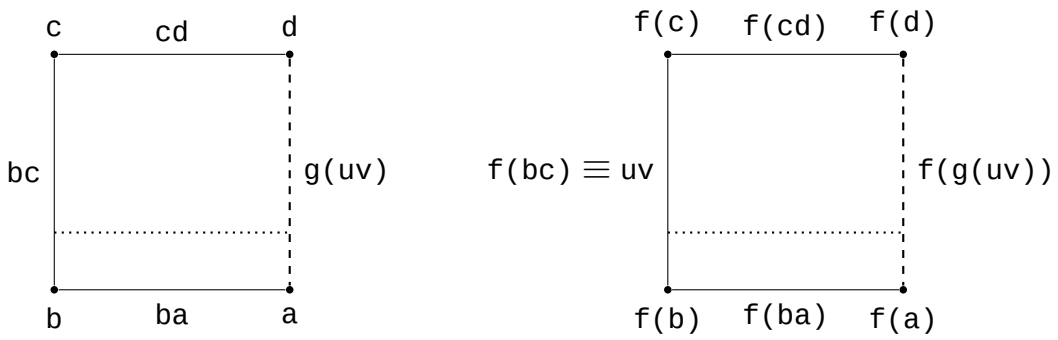


Рис. 4.1 — Поиск пути в заполнителе

```
g-then-f : (x : I1) -> x = f (g x)
g-then-f u      = idp
g-then-f v      = idp
g-then-f (uv j) =
  path (\i ->
    f (fill-2 (\_ _ -> I3)
              (box-1 b c a d (path ba) (path cd) (path bc)) i j))
```

Обратимся теперь к доказательству в другую сторону. Ищем терм типа:

```
f-then-g : (x : I3) -> x = g (f x)
```

Интересный случай — когда `x` это `bc j`, а все остальные получатся из граничных условий и элементарных соображений. В этом случае требуется доказать, по определению `f`, что `bc j = g (uv j)`, но мы только что видели именно такой путь! Только в этот раз даже не надо действовать отображением `f`.

В точках `b` и `c`, как видно из рисунка 4.1, возвращать необходимо пути `path ba` и `path cd`. В точке `ba j` нужен путь `(path ba) @ j = a`, причем при `j = left` он должен обращаться в `path ba`, а при `j = right` — в тождественный. Именно для этого нужна функция `ptrim'`, упомянутая в предыдущей главе. В итоге получаем:

```
f-then-g : (x : I3) -> x = g (f x)
f-then-g a      = idp
f-then-g (ba j) = ptrim' (path ba) j
f-then-g b      = path ba
f-then-g (bc j) =
  path (\i ->
      (fill-2 (\_ _ -> I3)
              (box-1 b c a d (path ba) (path cd) (path bc)) i j))
f-then-g c      = path cd
```

```
f-then-g (cd j) = ptrim' (path cd) j
f-then-g d      = idp
```

Таким образом, мы доказали, что **типы `I1` и `I3` гомотопически эквивалентны**.

Особо следует отметить, что в процессе доказательства не использовались никакие аксиомы, в том числе аксиома универсальности. И хотя в отдельных случаях использующее аксиому универсальности доказательство может оказаться проще, как, например, для упомянутого в начале главы утверждения об эквивалентности сферы и поверхности куба, определенных по индукции для любого $n$ внутри теории, в результате применения предложенной техники получается полностью *вычислимое* доказательство, что существенно облегчает дальнейшее использование полученной эквивалентностью.

### 4.3 Многомерные поверхности

Фактически, в предыдущем разделе мы научились строить эквивалентность одномерного куба (отрезка) и открытого $1$-короба, причем обладающую свойством сохранения границы. Предложенная техника так же хорошо работает и при большем числе измерений.

С ростом числа измерений растет и число граней в кубе, соответственно, и число стенок у короба, который можно заполнить, — у открытого $n$-короба будет $(2n + 1)$ стенка. В остальном же принцип построения отображений и доказательства эквивалентности остается прежним.

Отображение из $n$-короба в $n$-куб строится следующим образом:

- Грань короба, расположенная напротив отсутствующей, отображается на куб очевидным образом (она сама является $n$-кубом).
- Остальные грани сжимаются вдоль одной координаты и отображаются каждая в свою часть границы куба.

Обратное отображение строится так же, как и раньше:

- Короб заполняется и $n$-куб отображается на вновь появившуюся грань очевидным образом.

Доказательства гомотопности тоже мало отличаются от одномерного случая.

В первом случае (отображение точки куба на короб и обратно на куб) на границе пути будут тривиальны, поскольку точки возвращаются сами в себя, а для точек внутри куба достаточно посмотреть на соответствующую точку на грани короба, расположенной напротив отсутствующей, и провести через внутренность заполненного короба путь к соответствующей точке на появившейся новой грани, затем подействовать на этот путь построенным ранее отображением.

Во втором случае (отображение точки короба в куб, затем обратно на короб), для точек с грани, расположенной напротив отсутствующей, достаточно построить пути, как в предыдущем случае (и это будут уже те самые искомые пути, преобразовывать их никак не надо), а для точек с боковых граней достаточно сделать `ptrim'` (в направлении, в котором отсутствует грань) пути, проходящего по грани в сторону отсутствующей.

### 4.4 Разбиение на произвольное число поверхностей

Для начала вспомним, что мы накладывали на отображения, задающие эквивалентность, дополнительное условие: они оба должны были отображать границу на границу. Для чего это было нужно?

Во-первых, так мы гарантируем, что описанный метод останется полезен, если границу куба каким-то образом склеить. Например, в случае с типом `I1`, мы можем, добавив условий к определению типа, приклеить оба конца отрезка к точке и получить таким образом окружность. То же самое можно сделать с типом `I3`, соединив точки `a` и `d` в одну, и получив в результате треугольник. При этом доказательство эквивалентности останется, по сути своей, прежним.

Второе, для чего это условие необходимо, это работа с типами, содержащими большее или меньшее число отрезков. В случае открытого $n$-короба, мы можем склеить $(2n+1)$ грань в один $n$-куб, но что делать, если тип состоит из *меньшего* числа кубов? В этом случае всё предельно просто: достаточно представить, что кубов ровно нужное число, просто некоторые из них тривиальные.

В случае же, когда кубов *больше*, чем надо, самый простой способ — действовать поэтапно, склеивая кубы по очереди и доказывая промежуточные

эквивалентности. В качестве примера докажем, что граница квадрата эквивалентна сфере.

Границу квадрата определим следующим образом:

```
data Sq = a | b | c | d | bc I | ba I | cd I | ad I
  with
    bc left  = b
    bc right = c
    ba left  = b
    ba right = a
    cd left  = c
    cd right = d
    ad left  = a
    ad right = d
```

Она состоит из четырех точек и четырех отрезков между ними. Окружность же будет состоять из одной точки и одной петли:

```
data S1 = base | loop I
  with
    loop left  = base
    loop right = base
```

В поверхности квадрата имеются четыре отрезка, а в окружности — только один. Поскольку случай одномерный, за один раз мы можем склеить три отрезка в один, что меньше, чем надо. Потому введем промежуточный тип, состоящий из двух точек и двух отрезков:

```
data El = s | t | st1 I | st2 I
  with
    st1 left  = s
    st1 right = t
    st2 left  = s
    st2 right = t
```

Теперь нам предстоит доказать две эквивалентности (между `Sq` и `El`, а также между `El` и `S1`), а затем воспользоваться транзитивностью эквивалентности.

Отображения между `Sq` и `El` будут состоять из двух независимых «компонент»:

- Отрезку `st1` сопоставляется конкатенация `ab`, `bc` и `cd` и наоборот.

- Отрезку `st2` сопоставляется отрезок `ad` и наоборот.

Доказательства гопотопности точно так же делятся на две независмые «компоненты» каждое: для точек из `ab`, `bc` и `cd`, а также `st1` это точно такое же доказательство как было с `I1` и `I3`; для точек же из `ac` и `st2` доказательство тривиально, поскольку все эти точки переходят сами в себя.

В листинге 4.1 приведены определения отображений и полное доказательство эквивалентности.

Следующий шаг — эквивалентность `El` и `S1`. Здесь ситуация противоположная, путей, наоборот, на один меньше, чем надо. Как уже говорилось, это просто частный случай, когда один из путей тривиален.

Нужным образом модифицированное доказательство утверждения про `I1` и `I3` приведено в листинге 4.2.

Комбинируя эти идеи, получаем возможность доказывать утверждения об эквивалентности для типов, склеенных из произвольного числа поверхностей произвольной размерности.

```
f2 : Sq -> El
f2 a      = s
f2 (ba _) = s
f2 b      = s
f2 (bc j) = st1 j
f2 c      = t
f2 (cd _) = t
f2 d      = t
f2 (ad j) = st2 j

g2 : El -> Sq
g2 s       = a
g2 (st1 j) = fill-2 (\_ _ -> Sq)
                    (box-1 b c a d (path ba) (path cd) (path bc))
                    right j
g2 (st2 j) = ad j
g2 t       = d

g2-then-f2 : (x : El) -> x = f2 (g2 x)
g2-then-f2 s       = idp
g2-then-f2 (st1 j) =
  path (\i -> f2 (fill-2 (\_ _ -> Sq)
                         (box-1 b c a d (path ba) (path cd) (path bc))
                         i j))
g2-then-f2 (st2 _) = idp
g2-then-f2 t       = idp

f2-then-g2 : (x : Sq) -> x = g2 (f2 x)
f2-then-g2 a      = idp
f2-then-g2 (ba j) = ptrim' (path ba) j
f2-then-g2 b      = path ba
f2-then-g2 (bc j) =
  path (\i -> (fill-2 (\_ _ -> Sq)
                      (box-1 b c a d (path ba) (path cd) (path bc))
                      i j))
f2-then-g2 c      = path cd
f2-then-g2 (cd j) = ptrim' (path cd) j
f2-then-g2 d      = idp
f2-then-g2 (ad _) = idp
```

Листинг 4.1 — Эквивалентность `Sq` и `El`

```
f1 : El -> S1
f1 s       = base
f1 (st1 _) = base
f1 (st2 j) = loop j
f1 t       = base

g1 : S1 -> El
g1 base     = t
g1 (loop j) = fill-2 (\_ _ -> El)
                     (box-1 s t t t (path st1) idp (path st2))
                     right j

g1-then-f1 : (x : S1) -> x = f1 (g1 x)
g1-then-f1 base     = idp
g1-then-f1 (loop j) =
  path (\i -> f1 (fill-2 (\_ _ -> El)
                 (box-1 s t t t (path st1) idp (path st2)) i j))

f1-then-g1 : (x : El) -> x = g1 (f1 x)
f1-then-g1 s       = path st1
f1-then-g1 (st1 j) = ptrim' (path st1) j
f1-then-g1 (st2 j) =
  path (\i -> (fill-2 (\_ _ -> El)
                      (box-1 s t t t (path st1) idp (path st2))
                      i j))
f1-then-g1 t       = idp
```

Листинг 4.2 — Эквивалентность `El` и `S1`

# ЗАКЛЮЧЕНИЕ

В настоящей работе исследована возможность практического применения гомотопической теории типов, отличной от общепринятой, а именно *теории типов с интервалом и условиями*. Продемонстрированы преимущества этой теории типов на примере доказательства эквивалентности типов, являющихся аналогами CW-комплексов в топологии.

Изучена концепция *кубического заполнителя Кана*, позаимствованная из теории кубических множеств. Показано существование заполнителей в произвольной размерности, рассмотрены различные варианты их применения.

Предложена техника доказательства эквивалентности для широкого класса типов, основанная на использовании заполнителей Кана и позволяющая получать доказательства, имеющие вычислительный смысл. Приведены примеры применения этой техники.

Представляется целесообразным дальнейшее изучение свойств заполнителей и новых вариантов их применения в теории типов.

Существует возможность ещё большего упрощения доказательств с помощью расширения языка новыми конструкторами и элиминаторами для интервала — позволяющими «делить интервал на два». Необходимо продемонстрировать корректность такого расширения, например посредством построения интерпретации таких конструкторов и элиминаторов в модели теории.

Также важным направлением дальнейших исследований является изучение возможности применения полученных результатов в сочетании с опетопической теорией типов[22], оставшейся за рамками настоящей работы.

# СПИСОК ЛИТЕРАТУРЫ


1. Russell Bertrand. The Principles of Mathematics. Cambridge: Cambridge University Press, 1903.
2. Martin-Lef Per. Intuitionistic type theory // Naples: Bibliopolis. 1984. URL: http://people.csail.mit.edu/jgross/personal-website/papers/academic-papers-local/Martin-Lof80.pdf.
3. Sørensen Morten Heine B., Urzyczyn Pawel. Lectures on the Curry-Howard Isomorphism. 1998.
4. The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study: http://homotopytypetheory.org/book, 2013.
5. Voevodsky Vladimir. Univalent foundations project // NSF grant application. 2010. URL: http://www.math.ias.edu/vladimir/files/univalent_foundations_project.pdf.
6. Kapulkin Chris, Lumsdaine Peter LeFanu, Voevodsky Vladimir. The Simplicial Model of Univalent Foundations. 2012. cite arxiv:1211.2851Comment: 55 pages. Some portions of this paper previously appeared in the shorter note ”Univalence in Simplicial Sets”, arXiv:1203.2553. URL: http://arxiv.org/abs/1211.2851.
7. Bezem Marc, Coquand Thierry, Huber Simon. A Model of Type Theory in Cubical Sets // 19th International Conference on Types for Proofs and Programs (TYPES 2013) / под ред. Ralph Matthes, Aleksy Schubert. T. 26 из *Leibniz International Proceedings in Informatics (LIPIcs)*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. C. 107–128. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4628. (Keywords: Models of dependent type theory, cubical sets, Univalent Foundations).
8. Dybjer Peter. Internal Type Theory // TYPES ’95: Selected papers from the International Workshop on Types for Proofs and Programs. London, UK: Springer-Verlag, 1996. C. 120–134.
9. Curien P.-L. Substitution Up to Isomorphism // Fundam. Inf. Amsterdam, The Netherlands, The Netherlands, 1993. сент. T. 19, № 1-2. C. 51–85. URL: http://dl.acm.org/citation.cfm?id=175469.175471.
10. Hoffman Martin. Syntax and Semantics of Dependent Types // Semantics and Logics of Computation / под ред. P. Dybjer, A. M. Pitts. Cambridge University Press, 1997. C. 241–298.
11. Cohen Cyril, Coquand Thierry, Huber Simon [и др.]. CUBICAL. [Online]. URL: https://github.com/simhu/cubical.
12. Licata Dan, Brunerie Guillaume. A Cubical Approach to Synthetic Homotopy Theory // Preprint. 2015. URL: http://dlicata.web.wesleyan.edu/pubs/lb15cubicalsynth/lb15cubicalsynth.pdf.
13. Coquand Thierry. Cubical Type Theory. [Online]. 2015. май. URL: http://www.cse.chalmers.se/~coquand/rules5.pdf.
14. Cohen Cyril, Coquand Thierry, Huber Simon [и др.]. Cubical Type Theory. [Online]. URL: https://github.com/mortberg/cubicaltt.
15. Licata Dan. Cubical Type Theory. [Online]. URL: http://dlicata.web.wesleyan.edu/pubs/lb14cubical/lb14cubes-cmu-march-2015.pdf.
16. Isaev Valery. hoq. [Online]. URL: https://github.com/valis/hoq/.
17. Isaev Valery. Homotopy Type Theory with an interval type. [Unpublished]. URL: https://valis.github.io/doc.pdf.
18. Hatcher Allen. Algebraic topology. Cambridge, New York: Cambridge University Press, 2002.
19. Pitts Andrew M. An Equivalent Presentation of the Bezem-Coquand-Huber Category of Cubical Sets. // CoRR. 2014. T. abs/1401.7807. URL: http://arxiv.org/abs/1401.7807.
20. Gabbay Murdoch J. Foundations of nominal techniques: logic and semantics of variables in abstract syntax // Bulletin of Symbolic Logic. 2011. T. 17, № 02. C. 161–229. URL: http://journals.cambridge.org/abstract_S1079898600000536.
21. Pitts Andrew M. Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press, 2013. T. 57. URL: http://www.cl.cam.ac.uk/~amp12/talks/nominal-sets-1.pdf.
22. Finster Eric. Type Theory and the Opetopes. 2012. июнь. URL: http://sma.epfl.ch/~finster/opetope/types-and-opetopes.pdf.