УДК 1.168

ТЕОРЕТИКО-ТИПОВАЯ ЭКСПЛИКАЦИЯ МАТЕМАТИЧЕСКОГО АППАРАТА РОДОВ СТРУКТУР

О. А. Доманов
Институт философии и права СО РАН (г. Новосибирск)
domanov@philosophy.nsc.ru

Аннотация. Методология концептуального проектирования (С. П. Никаноров и др.) опирается на теорию родов структур Н. Бурбаки. Она использует классическую теорию множеств и логику предикатов. Вместе с тем, в последнее время в математике успешно применяются системы работы с доказательствами, основанные на интуиционистской теории типов (Coq, Agda, Lean и пр.) и обладающие большими возможностями в части вычисления, проверки доказательств и пр. Совмещение родоструктурного и теоретико-типового подходов может быть полезным и для того, и для другого. В статье представлена формализация теории родов структур в терминах теории типов (Cubical Agda, вариант гомотопической теории типов). Рассмотрено общее понятие рода структуры, ступени, М-графа, а также операции над термами ступеней и родами структур. На основе формализации сделаны два основных вывода: 1) несмотря на различие философских оснований и используемых концепций множества оба подхода имеют сходную формальную структуру и связаны через каррирование и декаррирование; 2) теория родов структур может служить метатеорией по отношению к теории типов. Оба подхода можно рассматривать как взаимодополнительные. Если первый описывает общую архитектуру теорий, их взаимодействие и генезис, то второй добавляет вычислительные и доказательственные возможности.

Ключевые слова: теория типов, концептуальное проектирование, роды структур, Бурбаки, Никаноров.

Для цитирования: Доманов, О. А. (2025). Теоретико-типовая экспликация математического аппарата родов структур. *Respublica Literaria*. Т. 6. № 3. С. 40-58. DOI: 10.47850/RL.2025.6.3.40-58.

SPECIES OF STRUCTURES: TYPE-THEORETICAL DEVELOPMENT

O. A. Domanov
Institute of Philosophy and Law SB RAS (Novosibirsk)
domanov@philosophy.nsc.ru

Abstract. Methodology of conceptual design (S. P. Nikanorov et al) is based on the theory of species of structures by N. Bourbaki. It uses classical set theory and predicate logic. Meanwhile, various proof assistants built on intuitionistic type theory (Coq, Agda, Lean and others) and possessing great capabilities in computation, proof checking etc. became successful recently in the area of mathematics. Combining the approaches of species of structures and type theory may be instrumental for both. The article presents a formalization of the theory of species of structures in terms of type theory (Cubucal Agda, a variant of homotopy type theory). General notion of species of structure, steps, M-graph, as well as operations on steps terms and species of structures are examined. The formalization leads to two conclusions: 1) in spite of differences in philosophical foundations and in the utilized concepts of sets both approaches have similar formal structure and are connected through carrying and uncurrying; 2) theory of species of structures may serve as a metatheory to type theory. Both approaches can be considered complimentary. While the former describes the general architecture of theories, theirs interaction and genesis, the latter brings in computational and proof capabilities.

Keywords: type theory, conceptual design, species of structures, Bourbaki, Nikanorov.

For citation: Domanov, O. A. (2025). Species of Structures: Type-Theoretical Development. *Respublica Literaria*. Vol. 6. No. 3. Pp. 40-58. DOI: 10.47850/RL.2025.6.3.40-58.

Методология концептуального проектирования [Кононенко и др., 2008] опирается на математический аппарат родов структур, который, в свою очередь, восходит к идеям Бурбаки [Бурбаки, 1965]. Аппарат обладает большой общностью и наглядностью, что позволяет разработать мощную технологию разработки концептуальных систем и работы с ними. При этом очень часто возникает потребность оперировать очень большими концептуальными схемами, что приводит к необходимости компьютеризации. И действительно, книга [Кононенко и др., 2008] содержит большой обзор программных средств, специально созданных для работы с родами структур. В ней большое внимание уделено программам для создания, редактирования, синтезирования родов структур, но мало, однако, говорится о компьютеризации работы с самими создаваемыми теориями – вывода и проверки теорем, тестировании корректности и пр. Можно предположить, что одна из важнейших причин этого состоит в том, что теория родов структур опирается на теорию множеств и традиционную логику предикатов, для которых имеется довольно ограниченный набор компьютеризованных решений (Isabelle/HOL, Metamath и др.). Между тем в последние годы в математике очень успешно применяются вычислительные системы, основанные на теориях типов, восходящих к интуиционистской логике П. Мартин-Лёфа [Martin-Löf, 1984] (называемые иногда современными теориями типов). В таких системах, как Coq или Agda, уже сформулирована значительная часть математики, они используются для проверки доказательств сложных теорем, язык Lean успешно применяется для расширения возможностей больших языковых моделей в части работы с математикой[1]. При этом особенность теории типов состоит в том, что она реализует так называемый вычислительный тринитаризм,[2] т. е. является одновременно логикой, онтологией и языком программирования. Это позволяет использовать ее не только в математике, но и для доказательства корректности программ, описания структур данных и пр. В то же время эти системы разрабатываются с ориентаций на нужды прежде всего математики, тогда как концептуальное проектирование может накладывать свои требования к оперированию с понятиями. Например, в упомянутых выше программных продуктах большое внимание уделено разработке графических сред для работы с родами структур, и это, вероятно, не случайно. Концептуальное проектирование в определенной мере является метаподходом в сравнении с собственно математическими – как и сама идея Бурбаки относится, вообще говоря, к метаматематике. В этом смысле объединение метауровня родов структур и вычислительных возможностей теории типов могло бы быть полезным и для того, и для другого. Дело, однако, осложняется тем, что подходы Бурбаки и теория типов опираются на разные философские основания. Хотя оба можно отнести к структурализму, второй, в отличие от первого, является также конструктивистским. Именно конструктивный подход придает теории типов вычислительный характер и делает ее не только логикой, но и языком программирования. Тем не менее шаги в этом направлении можно предпринять, и данная статья является одной из таких попыток. Мы рассмотрим экспликацию формализма родов структур в терминах языка теории типов и попробуем понять возможности их совмещения. В изложении аппарата родов структур я следую книгам [Кононенко и др., 2008,

[1] *AI Achieves Silver-Medal Standard Solving International Mathematical Olympiad problems*. (2024). [Online]. Available at: https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level (Accessed: 01 February 2021).

[2] Harper, R. (2011). *The Holy Trinity* [Online]. Available at: https://existentialtype.wordpress.com/2011/03/27/the-holy-trinity (Accessed: 27 March 2011).

с. 472 и сл.] и [Пономарёв, 2007, с. 181 и сл.]. В частности, почти не учитывается дальнейшее развитие теории в [Никаноров, 2013]. Для экспликации используется гомотопическая теория типов, более конкретно – кубическая Агда (Cubical Agda, [Vezzosi, Mörtberg, Abel, 2021]). Я предполагаю некоторое знакомство читателя с языком Агда, однако буду подробно пояснять конструкции.[3]

Основные определения родо-структурного подхода

Согласно определению [Кононенко и др., 2008, с. 477], *род структуры* состоит из базисных множеств, родовых структур (или родовых отношений), аксиом, термов и теорем. Базисные множества подразделяются на основные и вспомогательные, но мы в нашем построении не будем их различать. Родовые структуры и термы имеют типы, которые называются ступенями. Ступени определяются как множества, которые строятся из базисных множеств с помощью двух операций: взятия булеана и образования декартова произведения нескольких сомножителей (декартиана). Таким образом, *ступень* это:

1) любое из базисных множеств $X_1, \dots, X_n$;
2) булеан от ступени $S_j = \mathfrak{B}(S_i)$;
3) декартово произведение ступеней: $S_1 \times \cdots \times S_m$.

Хотя в математике декартово произведение обычно обобщается на один или даже нуль сомножителей, мы далее будем рассматривать только произведения с не менее чем двумя сомножителями. Конструкция ступени задается последовательностью операций по ее построению. Ступени, таким образом, задают форму родовых структур и термов, которые они типизируют. Базисные множества, родовые структуры и аксиомы составляют ядро теории, а термы, типизирующие их ступени и теоремы – ее тело. Поскольку отношения являются элементами булеанов декартовых произведений, родовые структуры задают исходные отношения теории, а термы – производные отношения и объекты. Идея Бурбаки при введении родов структур состояла в том, что все математические понятия выражают некоторые отношения, которые являются подмножествами соответствующих декартовых произведений, т. е. элементами тех или иных ступеней. Строение любого математического понятия описывается, таким образом, родовыми структурами.

Перейдем к аппарату, используемому для формализации. Мы будем строить ее в языке кубической Агды [Vezzosi, Mörtberg, Abel, 2021], которая является вариантом гомотопической теории типов [Univalent Foundations Program, 2013]. В этой теории запись `x : A` означает: «терм/объект/элемент `x` относится к типу `A`». Понимать тип означает понимать, какие объекты могут к нему относиться. В частности, в теории типов пропозиции понимаются как тип их доказательств, поэтому понимать пропозицию означает понимать, что может служить ее доказательством. Типы сами являются элементами особых типов, которые называются универсумами или сортами. Для предотвращения парадоксов все типы разделяются по уровням (`Level`) от нулевого до бесконечного. Соответственно, запись `A : Type ℓ` означает: «тип `A` относится к универсуму типов `Type ℓ` уровня `ℓ`». Низший универсум `Type 0` сокращается до `Type`. Кроме того, имеется универсум `Typeω`, который выше любого из `Type ℓ`.

[3] Полную формализацию можно найти по адресу: https://github.com/odomanov/nikanorov.

Одна из важных особенностей теории Мартин-Лёфа состоит в наличии в ней зависимых типов, т. е. типов, зависящих от других типов. В Агде они определяются как функции в универсумы. Например, `B : A → Type` обозначает функцию, которая для каждого элемента `x : A` позволяет вычислить тип, зависящий от него, который обозначается как `B x` (применение функции `B` к аргументу `x`). В качестве примеров можно привести тип жителей города, который зависит от города, или тип кортежей длины `n`, который зависит от `n`.

Из типов можно конструировать новые типы по определенным правилам. Важнейшим производным типом является тип (зависимых) функций, который записывается как `(x : A) → B x`. Определить такую функцию означает указать способ, каким для каждого элемента `x : A` можно вычислить элемент типа `B x`. Если `B` не зависит от `A`, то тип функции записывается как `A → B`. Элементы типа функций (т. е. функции) записываются как `λ(x : A) → b`, что означает, что значением этой функции для аргумента `x` является `b`. Функции от многих переменных записываются в каррированном виде как `(x : A) → (y : B) → C x y` или `A → B → C` для типов и `λ(x : A)(y : B) → c` для функций. При этом типы последующих аргументов могут зависеть от предыдущих аргументов. Тип функций также можно записывать как `∀(x : A) → B x`. Если Агда способна вычислить `A`, то он может быть опущен: `∀ x → B x`, `λ x y → b`. Аргументы функций могут быть необязательными и не указываться при применении функции. Такие аргументы помечаются фигурными скобками вместо круглых. Например, применение функции `f : {x : A} (y : B) → C` к терму `b : B` записывается просто как `f b`. В этом случае Агда должна вычислить первый аргумент самостоятельно. Если она этого не может, его можно указать явно: `f {a} b`.

В теории типов различается равенство по определению, например, `f = λ x → x`, и равенство как тип `a ≡ b`, называемое также типом равенства или типом тождества (equality type, identity type). Оно понимается как тип доказательств того, что `a` равно `b` (или тип их идентификаций). Особенность гомотопической теории типов состоит в том, что она отказывается от так называемого принципа единственности доказательства тождества (UIP, uniqueness of identity proofs). Это означает, что тип `a ≡ b` может иметь более одного элемента-доказательства. В этом случае мы можем говорить о равенстве этих элементов. Последнее, в свою очередь, также является типом, допускающим несколько доказательств, которые снова можно сравнивать, и далее до бесконечности. В результате типы в гомотопической теории типов похожи не столько на множества, сколько на группоиды, что допускает геометрическую (собственно, гомотопическую) интерпретацию, в которой объекты представляются как точки пространства, доказательства равенства между ними – как пути, равенства путей – как гомотопии и т. д. Нам не потребуются детали этой теории равенства, важно лишь, что она позволяет, в частности, классифицировать типы по их поведению относительно равенства. Нам ниже понадобятся стягиваемые (contractible) типы, т. е. такие, в которых имеется элемент, которому все остальные типы равны – в этом смысле такой тип является синглетоном, т. е. имеет один элемент. Второй класс типов называется пропозициями или простыми пропозициями (mere propositions). Он определяется как тип, который может быть либо пустым, либо синглетоном – т. е. имеет не более одного элемента. В Агде универсум таких типов обозначается `hProp`. Третий класс нужных нам типов определяется как тип, в котором равенство для любых элементов является пропозицией, т. е. имеет не более одного

доказательства. Такие типы называются множествами и их универсум обозначаются `hSet`. Элементы `hProp` и `hSet` определяются как пары, состоящие из типа и доказательства того, что он является пропозицией или, соответственно, множеством. Для первой компоненты пары `X : hProp` или `X : hSet` (т. е. для соответствующего типа) вводится сокращение `⟨ X ⟩`. Как и универсумы вообще, эти универсумы также содержат указание на уровень: `hProp ℓ` и `hSet ℓ`.

Типы в Агде определяются с помощью ключевого слова `data` и конструкторов, которые должны быть функциями, значением которых являются элементы этих типов. Например, тип натуральных чисел определяется как

```
data ℕ : Type where
  zero : ℕ
  suc  : (n : ℕ) → ℕ
```

Это тип универсума уровня 0, имеющий два конструктора: `zero` для первого числа и `suc n` для числа, следующего за `n`. Помимо прочего, конструкторы задают возможные формы элементов типа, которые могут быть использованы для поиска по шаблону (pattern matching).

Нам также понадобится тип `Vec A n` – тип кортежей элементов типа `A` длины `n`. Он определяется в библиотеке Агды следующим образом:

```
data Vec {ℓ} (A : Type ℓ) : ℕ → Type ℓ where
  []  : Vec A zero
  _::_ : ∀ {n} (x : A) (xs : Vec A n) → Vec A (suc n)
```

Тип имеет два конструктора. Первый – `[]` – конструирует пустой список типа `Vec A zero`. Второй – `_::_` – присоединяет объект `x` типа `A` к списку `xs` типа `Vec A n`, образуя на единицу больший список типа `Vec A (suc n)`. Подчерк в имени функции означает, что она может употребляться в инфиксной нотации: например, `a :: []` означает список из одного элемента, `a :: (b :: (c :: []))` – из трех элементов и т. д. Конструктор `_::_` определен как правоассоциативный, поэтому в последнем выражении скобки можно опустить: `a :: b :: c :: []`.

Наконец, для любого типа `A : Type ℓ₁` и зависимого от него типа `B : A → Type ℓ₂` определяется тип пар `Σ A B`, элементы которого являются парами, состоящими из объектов `x` типа `A` и объектов типа `B x`. Если понимать последние как доказательства `B x`, то `Σ`-тип можно интерпретировать как экзистенциальное утверждение «существует `x`, такой, что `B`». Доказательства этого утверждения можно также понимать как элементы подмножества `A`, т. е. тип тех `x : A`, для которых имеется элемент `B x`. Для `Σ`-типа имеется также синтаксический вариант `Σ[ x ∈ A ] B x`, который напоминает выражение с квантором существования. Функции `fst` и `snd` извлекают первую и вторую компоненту пары из элементов `Σ`-типа. Сама пара записывается как `a , b`. Декартово произведение `A × B` определяется как частный случай `Σ`-типа, при котором `B` не зависит от `A`.

Таковы нужные нам сведения о языке Агда, остальные будут вводиться по мере необходимости.[4]

[4]Полную документацию можно посмотреть по адресу: *Agda's documentation* [Online]. Available at: https://agda.readthedocs.io/ (Accessed: 10 August 2025).

Приступим теперь к формализации родо-структурных понятий. Для ее построения определим прежде всего понятие списка *базисных типов* (не различая основные и вспомогательные):

```
data BaseSets : ∀ {n} → Vec Level n → Typeω where
  []  : BaseSets []
  _::_ : ∀ {n ℓ} {Lℓ : Vec Level n} → hSet ℓ → BaseSets Lℓ → BaseSets (ℓ :: Lℓ)
```

Эта запись определяет тип `BaseSets`, зависящий от числа `n` и списка уровней `Vec Level n`. То есть `BaseSets Lℓ` представляет собой список типов длины `n`, относящихся к уровням, содержащимся в списке `Lℓ`. `BaseSets` имеет те же два конструктора, что и `Vec` – Агда, как правило, способна различить, конструктор какого именно типа следует использовать. Конструктор `[]` обозначает пустой список, индексированный пустым же списком уровней. Конструктор `_::_` из множества `hSet ℓ` и списка, индексированного `Lℓ`, конструирует список, индексированный расширенным списком уровней `ℓ :: Lℓ`. Он также декларирован как правоассоциативный. Таким образом, в `BaseSets` список типов и список уровней всегда согласованы по построению. Из определения также видно, что в качестве базисных типов мы выбираем только множества из универсумов `hSet ℓ`. Будем также обозначать функцию для извлечения `i`-го типа из списка `BS` как `BS [ i ]`.

Для определения понятия ступени определим сначала *схему ступени* (ср. [Пономарёв, 2007, с. 181]):

```
data СхемаСтупени (n : ℕ) : Type where
  base : Fin n → СхемаСтупени n
  𝔅    : СхемаСтупени n → СхемаСтупени n
  𝔇    : ∀ {m : ℕ} → Vec (СхемаСтупени n) (2 + m) → СхемаСтупени n
```

Схема зависит от числа базисных типов `n` и может иметь три формы (конструктора). Форма `base i` соответствует базисному типу номер `i` (здесь тип `Fin n` это тип натуральных чисел, меньших `n`, поэтому `i` всегда меньше `n`; для первых нескольких элементов типа `Fin` мы будем использовать имена `zero`, `one`, `two` и т. д.). Вторая форма `𝔅 S` обозначает схему булеана схемы `S`. Третья форма `𝔇 L` обозначает схему декартиана нескольких сомножителей. Здесь `L` обозначает кортеж схем ступеней длины `2 + m`. Поскольку `m` – натуральное число, длина этого списка не может быть меньше двух.

Множество схем ступеней собирается в направленный граф, который называется М-граф. Мы будем производить эту сборку по индуктивному алгоритму [Erwig, 2001], практически повторяющему процедуру описанную в [Кононенко и др., 2008, с. 487] и [Пономарёв, 2007, с. 182-183]. Говоря конкретно, мы начинаем с пустого графа и затем присоединяем ступени по одной. Ступени вида `base i` просто присоединяются к графу как отдельно стоящая вершина. Ступени вида `𝔅 S` присоединяются как вершина, связанная ребром с уже имеющейся в графе вершиной `S`. Ступени вида `𝔇 L` присоединяются как вершина, связанная со всеми ступенями, перечисленными в списке `L`. Таким образом, *М-граф* определяется следующим образом:

```
data М-граф (n : ℕ) : (k : ℕ) → Type where
  ∅   : М-граф n 0
  _&_ : ∀ {k} → М-узел n k → М-граф n k → М-граф n (1 + k)
```

Он индексирован двумя параметрами – числом базисных типов `n` и числом узлов графа `k` – и имеет два конструктора: `∅` – для пустого графа и `_&_` – для добавления одного узла. Таким образом, выражение `N & M` обозначает граф, построенный добавлением узла `N` к графу `M`. Узлы могут быть трех видов и определяются следующим образом:

```
data M-узел (n : ℕ) : ℕ → Type where
  base : ∀ {k} → Fin n → M-узел n k
  𝔅    : ∀ {k} → Fin (1 + k) → M-узел n (1 + k)
  𝔇    : ∀ {k m} → Vec (Fin (1 + k)) (2 + m) → M-узел n (1 + k)
```

Они также имеют два параметра: число базисных типов `n` и число узлов графа, к которому присоединяется данный узел. Последнее число ограничивает номера узлов, с которыми может быть связан присоединяемый узел; эти номера начинаются с нуля и отсчитываются, начиная с «вершины» графа (см. примеры ниже). В результате, `base i` – это узел, являющийся базисным типом. Далее, `𝔅 i` – это узел булеана, причем `i` это номер узла, с которым он связывается ребром. Как видно из определения, узел может быть присоединен только к графам, имеющим как минимум один узел. Наконец, `𝔇 L` – это узел декартиана, в котором список `L` состоит из номеров, с которыми должен быть связан этот узел. Из его определения видно, что он имеет длину `2 + m` не меньше двух и также может присоединяться только к графу, в котором есть как минимум один узел.[5] Таким образом, M-граф формализует процедуру построения рода структур из базисных множеств.

Для графа определим функцию извлечения схемы по ее номеру (при определении функции сначала указывается ее тип, а затем значения для всех возможных значений аргументов):

```
M-Схема : ∀ {n k} → M-граф (1 + n) (1 + k) → Fin (1 + k) → СхемаСтупени (1 + n)
M-Схема (base i & _) zero = base i
M-Схема (𝔅 i & M) zero = 𝔅 (M-Схема M i)
M-Схема (𝔇 is & M) zero = 𝔇 (map (λ k → M-Схема M k) is)
M-Схема {k = suc k} (_ & M) (suc i) = M-Схема M i
```

Здесь `map f L` вычисляет новый список, применяя последовательно функцию `f` к элементам списка `L`. Подчерк вместо выражения означает, что это выражение либо не используется в вычислении, либо Агда сама в состоянии его вычислить.

Перейдем теперь к основной части нашего построения – роду структуры. Агда является, помимо прочего, языком программирования и содержит развитую модульную систему, позволяющую разбивать программу на независимые части и затем собирать их воедино. В частности, модули в Агде могут иметь входные параметры, позволяющие абстрагировать какие-то из их внутренних определений. Мы будем строить теорию родов структур как модуль с тремя параметрами (два из которых неявные):

```
module РодСтруктуры {n : ℕ} {Lℓ : Vec Level (1 + n)} (BS : BaseSets Lℓ)
```

[5]Ограничения на применимость, которые мы здесь видим, относятся к характерным особенностям теории типов: понятия определяются через конструкторы, причем так, что некорректные объекты просто не могут быть построены.

Здесь `n` – число базисных типов, `Lℓ` – список уровней, к которым они относятся, и `BS` – сами соответствующие типы. Как видно, мы ограничимся непустым множеством базисных типов.

Булеан `ℙ`, т. е. тип подтипов, определен в библиотеке Агды как тип характеристических функций:

```
ℙ : Type ℓ → Type (ℓ-suc ℓ)
ℙ X = X → hProp _
```

При этом, как можно показать, `ℙ X` оказывается множеством для любого типа `X`. Для элементов `ℙ X` определен предикат принадлежности `x ∈ P`. Кроме того, для каждого `P : ℙ X` мы можем определить порождаемое им подмножество `X` как тип пар, состоящих из элементов `X` и доказательств того, что для них выполнено `P`:

```
ℙΣ : ∀ {X : Type ℓX} (P : ℙ X) → Type ℓX
ℙΣ {X = X} P = Σ[ x ∈ X ] ⟨ P x ⟩
```

Как можно показать, `ℙΣ` также является множеством.

Теперь, имея базисные типы, определим функцию, которая вычисляет по каждой схеме соответствующий ей тип теории типов, т. е. ступень:

```
Ступень : (S : СхемаСтупени (1 + n)) → Type (ℓtot S)
Ступень (base i) = fst (BS [ i ])
Ступень (𝔅 S) = ℙ (Ступень S)
Ступень (𝔇 (S1 :: S2 :: [])) = Ступень S1 × Ступень S2
Ступень (𝔇 (S1 :: S2 :: S3 :: tail)) = Ступень S1 × Ступень (𝔇 (S2 :: S3 :: tail))
```

Здесь `ℓtot` это функция, вычисляющая уровень универсума, соответствующий схеме, по правилу:

```
ℓtot : СхемаСтупени (1 + n) → Level
ℓtot (base i) = lookup i Lℓ
ℓtot (𝔅 S) = ℓ-suc (ℓtot S)
ℓtot (𝔇 (S1 :: S2 :: [])) = ℓ-max (ℓtot S1) (ℓtot S2)
ℓtot (𝔇 (S1 :: S2 :: S3 :: tail)) = ℓ-max (ℓtot S1) (ℓtot (𝔇 (S2 :: S3 :: tail)))
```

(библиотечная функция `lookup i L` извлекает `i`-й элемент из списка `L`; функции `ℓ-suc` и `ℓ-max` действуют на уровни универсума и вычисляют, соответственно, следующий и максимальный уровень). Мы видим, что ступени представляют собой типы языка Агда, построенные с помощью операций булеана `ℙ` и декартова произведения `×`. Можно доказать, что коль скоро базисные типы являются множествами, все ступени также оказываются множествами. Наконец, мы можем определить функцию для извлечения ступени из графа по индексу:

```
M-Ступень : ∀ {k} → (M : M-граф (1 + n) (1 + k)) → (i : Fin (1 + k)) → Type _
M-Ступень M i = Ступень (M-Схема M i)
```

На этом мы закончили определение родовых структур и типизирующих их ступеней. Поскольку ступени у нас определены как типы Агды, аксиомы и теоремы также могут формулироваться в языке Агды. Термы теории родов структур также оказываются термами Агды. Нам осталось рассмотреть операции, определенные для термов и родов структур. Начнем с операций для термов [Кононенко и др., 2008, с. 480-484].

Операции над термами

Понижение размерности

Операция применяется только к синглетонам, т. е. элементам булеанов, состоящим из одного объекта, и имеет своим результатом этот объект как элемент низшей ступени. В кубической Агде предикат для проверки того, что тип является синглетоном, обозначается `isContr` (англ. contractible, стягиваемый) и означает, что имеется элемент, которому равны все остальные элементы:

```
isContr : Type ℓ → Type ℓ
isContr A = Σ[ x ∈ A ] (∀ y → x ≡ y)
```

В нашем случае синглетоном должен быть соответствующий тип `ℙΣ`, и операция понижения размерности выглядит следующим образом:

```
Ступень↓ : ∀ {S : СхемаСтупени (1 + n)}
         → (t : Ступень (𝔅 S))
         → isContr (ℙΣ t)
         → Ступень S
Ступень↓ _ p = fst (fst p)
```

Как видно, она определена только для элементов ступеней вида `𝔅 S`, причем являющихся синглетонами, т. к. при применении функции `Ступень↓` мы должны указывать доказательство для `isContr (ℙΣ t)`.

Малая проекция

Она определяется следующим образом:

```
pr : ∀ {m} {LS : Vec (СхемаСтупени (1 + n)) (2 + m)}
   → Ступень (𝔇 LS)
   → (i : Fin (2 + m))
   → Ступень (lookup i LS)
pr {LS = _ ∷ _ ∷ []}    (x , _)     zero = x
pr {LS = _ ∷ _ ∷ []}    (_ , x)      one = x
pr {LS = _ ∷ _ ∷ _ ∷ _} (x , _)     zero = x
pr {LS = _ ∷ _ ∷ _ ∷ _} (_ , xs) (suc i) = pr xs i
```

Операция применима лишь к элементам ступеней вида `𝔇 LS` – т. е. кортежам – и выделяет `i`-й элемент кортежа.

Повышение размерности

Эта операция переводит элемент ступени `S` в элемент `𝔅 S`, являющийся содержащим его синглетоном.

```
Ступень↑ : ∀ {S : СхемаСтупени (1 + n)} → Ступень S → Ступень (𝔅 S)
Ступень↑ {S} t = λ x → (t ≡ x) , СтупеньIsSet S t x
```

Мы можем проверить, что результат этой операции – синглетон:

```
Ступень↑IsSingl : {S : СхемаСтупени (1 + n)}
                → (t : Ступень S)
                → isContr (ℙΣ (Ступень↑ {S = S} t))
Ступень↑IsSingl t = isContrSingl t
```

Произведение элементов

Эта операция из набора термов некоторых ступеней составляет кортеж, являющийся элементом декартового произведения этих ступеней. Для ее формализации определим сначала тип списков термов.

```
data TermList : ∀ {m} (LS : Vec (СхемаСтупени (1 + n)) m) → Typeω where
  [] : TermList []
  _::_ : ∀ {m} {LS : Vec (СхемаСтупени (1 + n)) m} {S : СхемаСтупени (1 + n)}
      → Ступень S → TermList LS → TermList (S :: LS)
```

Это список, индексированный кортежами ступеней. Тогда искомая операция равна:

```
prodN : ∀ {m} {LS : Vec (СхемаСтупени (1 + n)) (2 + m)}
      → (Lt : TermList LS) → Ступень (𝔇 LS)
prodN (t1 :: t2 :: []) = t1 , t2
prodN (t1 :: t2 :: t3 :: tail) = t1 , prodN (t2 :: t3 :: tail)
```

Большая проекция отношения

Операция действует на отношениях, т. е. на булеанах декартианов, и выделяет множество тех элементов, которые служат малыми проекциями для некоторого индекса `i`:

```
Pr : ∀ {m} {LS : Vec (СхемаСтупени (1 + n)) (2 + m)}
   → Ступень (𝔅 (𝔇 LS))
   → (i : Fin (2 + m))
   → ℙ (Lift {j = ℓ-max (ℓtot (lookup i LS)) (ℓtot (𝔇 LS))} (Ступень (lookup i LS)))
Pr {m} {LS} S i = λ x → isPr (lower x) , squash₁
  where
  isPr : Ступень (lookup i LS) → Type _
  isPr s = ∃[ y ∈ Ступень (𝔇 LS) ] (y ∈ S) × (s ≡ pr y i)
```

Здесь нам требуется аккуратная разметка уровней универсумов. Дело в том, что степень `ℙ` определена как функция между типами одного универсума, у нас же условие `isPr` может, вообще говоря, относиться к какому угодно универсуму. В результате, значение `ℙ (Ступень (lookup i LS))` может относиться к универсуму, который гораздо выше, чем `Ступень (lookup i LS)`. Поэтому для согласования нам требуется поднять последнюю в подходящий универсум с помощью функции `Lift`.

Кроме того, для определения `Pr` как элемента `ℙ` нам требуется функция в тип пропозиций `hProp`, поэтому мы определяем здесь условие с помощью типа `∃`, который определяется как усечение (truncation) `Σ`-типа до пропозиции путем отождествления всех его элементов (если они есть). Объект `squash₁` является доказательством того, что `∃`-тип является пропозицией.

Задание множества перечислением

```
enum : ∀ {S : СхемаСтупени (1 + n)} {m} {LS : Vec (Ступень S) m}
     → (Ступень (𝔅 S))
enum {S} {m} {LS} = λ x → any (λ y → (x ≡ y) , СтупеньIsSet S x y) LS
  where
  open import Cubical.Functions.Logic
  ⊥ʰ : ∀ {ℓ} → hProp ℓ
  ⊥ʰ = ⊥* , λ ()

  any : ∀ {ℓ} {A : Type ℓ} (P : A → hProp ℓ) → Vec A m → hProp ℓ
  any P V = foldr (λ x y → (P x) ⊔ y) ⊥ʰ V
```

Здесь `⊔` – это дизъюнкция на пропозициях, определенная в модуле `Cubical.Functions.Logic`, а пропозиция `any P V` определена как дизъюнкция `P x` для всех `x` из `V`. Таким образом, она имеет доказательство, если какой-то элемент `V` удовлетворяет предикату `P`. В определении `enum` этим предикатом является равенство, поэтому элементами `enum` как подмножества будут те `x`, которые равны какому-то элементу `LS`, что и соответствует заданию его перечислением.

Простое объединение и пересечение, импликация, отрицание, разность, симметрическая разность, произведение, возведение в степень

Все эти операции определяются на ступенях вида `𝔅 S` и сводятся к операциям на степени `ℙ`. Например, для пересечения имеем:

```
_∩ᵖ_ : ℙ X → ℙ X → ℙ X
A ∩ᵖ B = λ x → A x ⊓ B x
```

Здесь `⊓` – логическая операция конъюнкции на пропозициях `hProp`. Остальные операции строятся аналогично.

Особого внимания требует возведение в степень. Для множеств A и B это построение множества функций B^A из A в B. В [Кононенко и др., 2008, с. 484] оно строится заданием условия на множество пар, выражающего функциональность соответствующего бинарного отношения. Однако в теории типов функции из типа в тип сами составляют тип, поэтому мы можем выразить это условие через существование некоторой функции:

```
_^ᵖ_ : ∀ {ℓX ℓY} {X : Type ℓX} {Y : Type ℓY} → ℙ Y → ℙ X → ℙ (ℙ (X × Y))
_^ᵖ_ {ℓX} {ℓY} {X} {Y} B A pairs = isfunA→B pairs , squash₁
  where
  isfunA→B : ℙ (X × Y) → Type _
  isfunA→B ps = ∃[ f ∈ (ℙΣ A → ℙΣ B) ]
                (ℙΣ ps) ≡ (Σ[ x ∈ ℙΣ A ] ⟨ (B (fst (f x))) ⟩)
```

Здесь `isfunA→B ps` выражает условие функциональности, которое утверждает, что существует функция `f`, такая что множество пар `ℙΣ ps` равно множеству пар аргументов и значений этой функции.

В итоге, операции на термах ступеней совпадают с операциями на ℙ:

```
_∪_ : ∀ {S : СхемаСтупени (1 + n)} → (A B : Ступень (𝔅 S)) → Ступень (𝔅 S)
A ∪ B = A ∪ᵖ B

_^_ : ∀ {SA SB : СхемаСтупени (1 + n)}
    → (B : Ступень (𝔅 SB))
    → (A : Ступень (𝔅 SA))
    → Ступень (𝔅 (𝔅 (𝔇 (SA :: SB :: []))))
B ^ A = B ^ᵖ A
```

и так далее.

Таким образом, операции над термами могут быть определены в языке Агда. Прежде чем мы перейдем к операциям на родах структур, рассмотрим небольшой пример.

Пример: бинарные отношения на множестве

Построим теорию бинарных отношений на произвольном множестве `hX : hSet ℓ`. Проделаем это внутри параметризованного модуля:

```
module ExampleBinaryRel {ℓ : Level} (hX : hSet ℓ)
```

Зададим род структуры с единственным базисным множеством, открыв определенный ранее модуль `РодСтруктуры` с соответствующим параметром:

```
open РодСтруктуры (hX :: [])
```

Далее определим ступень

```
S = Ступень (𝔅 (𝔇 (base zero :: base zero :: [])))
```

Можно показать, что `S ≡ ℙ (X × X)`, где `X = ⟨ hX ⟩`, т. е. `S` является множеством подмножеств декартова произведения `X × X`. Элементы `S` это различные бинарные отношения. Построим для примера рефлексивное отношение. В теории типов отношения определяются как зависимые типы, т. е. как функции в универсумы типов вида `X → Y → ... → Type ℓ`. Это обобщение понятия характеристической функции. Мы рассмотрим частный случай функций в универсум типов `hProp` и определим следующий тип бинарных отношений:

```
PropRel : ∀ {ℓX ℓY} (X : Type ℓX) (Y : Type ℓY) (ℓ : Level) → Type _
PropRel X Y ℓ = X → Y → hProp ℓ
```

Условие рефлексивности для таких отношений выглядит следующим образом:

```
IsRefl : ∀ {ℓ ℓ'} {X : Type ℓ} → PropRel X X ℓ' → Type _
IsRefl R = ∀ x → fst (R x x)
```

Нам для наложения условия на `X × X` требуется формулировка для множества пар; она связана с `IsRefl` с помощью операции декаррирования (`uncurry`):

```
IsRefl× : ∀ {ℓ} {X : Type ℓ} → ℙ (X × X) → hProp _
IsRefl× R× = (∃[ R ∈ PropRel _ _ _ ] (IsRefl R) × (R× ≡ uncurry R))
                   , squash₁
```

Формулировка означает, что элемент `R×` булеана `ℙ (X × X)` является рефлексивным отношением в смысле рода структуры, если существует рефлексивное отношение в смысле теории типов, такое что он равен его декаррированному варианту. Тогда множество рефлексивных отношений определяется следующим образом:

```
s : ∀ {Sh : СхемаСтупени 1} → Ступень (𝔅 (𝔅 (𝔇 (Sh :: Sh :: []))))
s = IsRefl×
```

Мы видим, что этот терм просто равен условию рефлексивности. Это неудивительно, поскольку то и другое представляет собой один и тот же предикат, а именно предикат, выделяющий множество рефлексивных отношений из множества всех отношений. Это следствие нашего определения булеана `ℙ X` как предиката `X → hProp _`. Мы видим здесь как связаны подходы теории типов и родов структур: если в первом отношения определяются как функции в универсум типов (т. е. зависимые типы), то во втором – как элементы булеана декартова произведения. Эти подходы, таким образом, связаны через каррирование и декаррирование.

Мы встречаемся здесь с различием в понимании множества. Теория родов структур опирается на традиционную теорию множеств, основанную на понятии принадлежности элемента множеству. В теории типов же принадлежность может пониматься в двух смыслах. Первый из них выражается в суждении типизации `a : A` – его можно понимать как утверждение о принадлежности к множеству объектов типа `A`. Важно, что это утверждение не является пропозицией, т. е. тем, что может быть доказано, что может иметь доказательство. Это суждение знания, опирающееся не на доказательство, а на очевидность [Martin-Löf, 1987]. Принадлежность же как пропозиция должна быть специальным образом построена. Например, в определении `ℙ` она определяется для подтипов определенного типа как выполнение предиката: каждое подмножество-элемент `ℙ X` является предикатом на `X`, и принадлежать этому подмножеству означает выполнять этот предикат (что и выражено, в частности, в определении `ℙΣ`). Это принадлежность в смысле обладания свойством. Закономерно поэтому, что множество `s` выше просто совпадает с определенным свойством. Исходно же принадлежность в теории типов «плоская» – мы не можем без специальной конструкции говорить о принадлежности типов другим типам. В теории родов структур мы встречаем принадлежность как в первом смысле (базисные множества), так и во втором (булеан, т. е. подмножества как свойства). Если понимать

теорию типов как теорию множеств (что лишь ограниченно корректно), то это теория не столько принадлежности, сколько функций на множествах; это, в частности, помогает ей избегать известных парадоксов. Тем не менее, хотя рассматриваемые нами подходы основаны на разных понятиях множества, формальная структура требующихся нам понятий, как мы видим, сходна.

Операции над родами структур

Другими важными операциями теории родов структур являются действия с самими родами структур, такие как их синтез, разделение и пр. [Кононенко и др., 2008, с. 488-496]. Естественным способом было бы определение их через соответствующие операции с М-графами, но теория типов обладает своими средствами композиции и декомпозиции теорий, которые могут оказаться более удобными. Например, операции порождения множества структур и обратная ей операция порождения структуры могут быть определены следующим образом:

```
→𝔅 : ∀ {Sh : СхемаСтупени (1 + n)}
   → (Ступень Sh → hProp _)
   → Ступень (𝔅 Sh)
→𝔅 P = P

←𝔅 : ∀ {Sh : СхемаСтупени (1 + n)}
   → Ступень (𝔅 Sh)
   → Ступень Sh → hProp _
←𝔅 P = P
```

Они оказываются простыми, поскольку условие на ступень `Ступень Sh → hProp ℓ` является предикатом и совпадает с элементом ступени `Ступень (𝔅 Sh)`; эти две операции как раз демонстрируют их изоморфизм. По этой причине неясно, какие из родо-структурных операций нам могут понадобиться при построении теории. Вполне возможно, что многие из них можно заменить правилами оперирования типами и термами из теории типов. Например, теория типов следует своим способам построения математических понятий функций, порядка и пр., приводимых в качестве примеров в [Кононенко и др., 2008] и [Пономарёв, 2007]. Само по себе это требует отдельного исследования. Я здесь ограничусь лишь несколькими замечаниями и примерами.

Рассмотрим синтез двух простых структур. Первая имеет следующий М-граф:

```
M1 : М-граф 1 _
M1 = 𝔅 zero & base zero & ∅
```

Это теория, состоящая из одного базисного множества и содержащая его самого и его булеан. Вторая структура формализует бинарные отношения на двух множествах:

```
M2 : М-граф 2 _
M2 = 𝔅 zero
   & 𝔇 (zero :: one :: [])
   & base zero
   & base one
   & ∅
```

Тип натуральных чисел является множеством, поэтому, обозначив соответствующий элемент `hSet` через `hℕ`, мы можем определить список базисных множеств и ступени:

```
BS1 : BaseSets _
BS1 = hℕ ∷ []

S1 = M-Ступень BS1 M1 zero
S2 = M-Ступень BS1 M1 one
```

Будучи пересчитаны в типы теории типов, `S1` и `S2` равны, соответственно, `ℙ ℕ` и `ℕ`. Чтобы построить бинарное отношение на этих множествах, синтезируем две теории, положив `S1` и `S2` как базисные множества для схемы `M2` (предварительно построив соответствующие элементы `hSet`):

```
hS1 : hSet _
hS1 = S1 , СтупеньIsSet BS1 (M-Схема M1 zero)

hS2 : hSet _
hS2 = S2 , isSetℕ

BS2 : BaseSets _
BS2 = hS1 ∷ hS2 ∷ []

S3 = M-Ступень BS2 M2 zero
S4 = M-Ступень BS2 M2 one
S5 = M-Ступень BS2 M2 two
S6 = M-Ступень BS2 M2 three
```

Здесь `S3 ≡ ℙ (ℙ ℕ × ℕ)`, `S6 ≡ ℕ` и т. д. Построим для примера понятие универсального отношения в этой структуре. Для этого определим соответствующие предикаты аналогично `IsRefl` и `IsRefl×` выше:

```
IsUniversal : ∀ {ℓX ℓY ℓ'} {X : Type ℓX} {Y : Type ℓY} → PropRel X Y ℓ' → Type _
IsUniversal R = ∀ x y → fst (R x y)

IsUniversal× : ∀ {ℓX ℓY} {X : Type ℓX} {Y : Type ℓY} → ℙ (X × Y) → hProp _
IsUniversal× R× = (∃[ R ∈ PropRel _ _ _ ] (IsUniversal R) × (R× ≡ uncurry R))
                  , squash₁
```

Добавляем ступень:

```
S7 = M-Ступень BS2 (𝔅 zero & M2) zero
```

и строим множество универсальных отношений:

```
universal : S7
universal = IsUniversal×
```

Терм `universal` является элементом булеана и соответствует множеству универсальных отношений на множествах $\mathbb{P}\ \mathbb{N}$ и $\mathbb{N}$, которые взяты из теории `M1`. Заметим, что, как и раньше, терм совпадает с условием универсальности.

В качестве базисных мы можем также выбирать элементы ступеней другой теории, т. е. ее термы. Пусть, например, в первой теории мы имеем множество четных чисел:

```
s : S1
s x = isEvenT x , isPropIsEvenT x
```

Построим для него соответствующий `hSet` и список базисных множеств:

```
hs : hSet _
hs = ℙΣ s , isSetΣSndProp isSetℕ λ n → snd (s n)

BS3 : BaseSets _
BS3 = hs ∷ hs ∷ []
```

Тогда, например, `M-Ступень BS3 M2 zero` равна $\mathbb{P}$ `(⟨ hs ⟩ × ⟨ hs ⟩)` и мы можем строить в синтезированной теории отношения на множестве четных чисел.

Все эти конструкции оказываются возможными, потому что мы определили `РодСтруктуры` как модуль, зависящий от списка базисных множеств как от параметра. Это позволяет проводить синтез теорий, просто выбирая в качестве параметра множества из разных теорий. Система модулей, используемая в Агде и других языках для композиции программ из независимо разрабатываемых частей, может оказаться более эффективным способом работы с теориями в целом, чем композиция и декомпозиция М-графов. К сожалению, модули в Агде не являются полноценными (first-class) единицами как в языках типа ML или Ocaml, но Агда содержит типы записей (records, называемые в других языках также структурами), которые во многом ведут себя как такие модули. Мы выше представляли родовые структуры в терминах степени $\mathbb{P}$, декартового произведения $\times$ и множества $\mathbb{P}\Sigma$. Однако декартово произведение является частным случаем Σ-типа, а последний – частным случаем более общего понятия записи. Например, Σ-тип в библиотеке Агды просто определяется как запись. Запись представляет собой кортеж с поименованными полями, которые выступают как функции, позволяющие извлекать соответствующие компоненты кортежа. В общем виде тип записи имеет (упрощенно) следующий вид:

```
record R {ℓ₁ ... ℓₙ} (X₁ : Type ℓ₁) ... (Xₙ : X₁ → ... → Type ℓₙ) : Type _ where
  field
    <fld₁> : T₁
    <fld₂> : T₂
    ...
    <fldₘ> : Tₘ
```

где типы T_i сконструированы из типов X_i и, возможно, предыдущих полей. В частности, эти типы сами могут быть записями. Записи используются для группирования различных объектов, и функции `<fld`$_i$`>` служат для их извлечения. Например, декартово произведение `n` не зависящих друг от друга сомножителей может определяться как:

```
record 𝔇ₙ {ℓ₁ ... ℓₙ} (X₁ : Type ℓ₁) ... (Xₙ : Type ℓₙ) : Type _ where
  field
    x₁ : X₁
    x₂ : X₂
    ...
    xₙ : Xₙ
```

Добавив условие, мы можем получить тип, который содержит только элементы, для которых это условие выполнено:

```
record 𝔚𝔇ₙ {ℓ₁ ... ℓₙ ℓ'} (X₁ : Type ℓ₁) ... (Xₙ : Type ℓₙ)
  (P : X₁ → ... → Xₙ → Type ℓ'): Type _ where
  field
    x₁ : X₁
    x₂ : X₂
    ...
    xₙ : Xₙ
    isP : P x₁ ... xₙ
```

Тем самым, мы получаем булеан от декартова произведения. В частности, для единственного `X` мы получим тип, эквивалентный булеану от `X`. Очевидно также, что само декартово произведение является частным случаем $\mathfrak{WD}_n$ для всегда выполненного условия `P`. В результате все ступени родов структур могут быть выражены через записи $\mathfrak{WD}_n$. При этом их синтез может осуществляться через параметры, как описано выше. Более того, помимо полей запись может содержать произвольные определения, которые также могут использоваться в определении полей. Каждая запись определяет модуль (с таким же именем), содержащий ее поля и дополнительные определения. Они ведут себя во многих отношениях как обычные модули, их можно открывать полностью или частично, они могут быть параметризованы и т. д. В этом смысле для задач построения сложных понятий и теорий из простых записи предоставляют богатый набор инструментов.

К сожалению, Агда не позволяет построить простые функции, подобные `Ступень` и `М-Ступень`, позволяющие переводить схемы и М-графы в записи теории типов.[6] В любом случае можно констатировать, что Агда содержит средства, требующиеся для работы с теориями, последовательного построения сложных теорий, их синтезирования и пр., однако содержит их неявно. В этом смысле теория родов структур может предоставить для теории типов метатеорию, облегчающую проектирование концептуальных структур.

[6]Она, тем не менее, имеет механизм рефлексии, позволяющий, в принципе, это сделать. Другим вариантом является теоретико-типовой язык Lean, содержащий развитые средства манипуляции с синтаксисом, в котором эта задача решается проще. Lean, с другой стороны, преимущественно ориентирован на классическую математику, в нем, например, невозможна гомотопическая теория типов. Выбор языка зависит от конкретных потребностей концептуального проектирования – достаточно ли, например, классической математики. Возможно также, что оптимальным является разработка специализированного языка (DSL), подобного Lean, Agda или другим теоретико-типовым языкам.

Заключение

Мы можем сделать два вывода из проведенной формализации.

Прежде всего мы видим, что с формальной точки зрения связь двух подходов осуществляется через операции каррирование и декаррирования. Несмотря на разницу онтологий, а также лежащих в их основе понятий множества, формальная структура оказывается сходной. Хотя в теории типов булеан (точнее, множество подтипов) вполне возможно определить, он не используется в ней активно. Выражаемые с его помощью структуры формализуются через зависимые типы или функции в универсумы (успех теории Мартин-Лёфа вообще во многом связан именно с понятием зависимого типа, который позволил формализовать утверждения с кванторами). Булеан и зависимые типы в определенном смысле взаимозаменимы. В нашем изложении, например, булеаны от декартовых произведений часто представлялись как типы многоместных функций в универсумы `Type ℓ` или `hProp ℓ`. В этом смысле, теории типов и родов структур оказываются двумя сходными способами представления математических (и не только) теорий.

Второй вывод состоит в том, что теория родов структур выступает как метатеория по отношению к теории типов: она касается не столько конкретного определения тех или иных понятий, сколько их взаимодействия, общей схемы теорий, синтеза различных теорий и пр. В теоретико-типовых построениях эта информация, разумеется, присутствует, но в неявном виде. В этом смысле теории родов структур и типов дополняют друг друга. Если первая описывает общую архитектуру теорий, их взаимодействие и генезис, то вторая способна добавить к этому вычислительные возможности, причем на уровне как конкретных теорий, так и их связей. Так, мы видели, что Агда выступала в качестве средства формализации как теорий, так и метатеории (схем ступеней, М-графа). Насколько этой способности Агды (или других теоретико-типовых языков) служить метаязыком достаточно для целей концептуального проектирования остается открытым вопросом.

Список литературы / References

Бурбаки, Н. (1965). *Теория множеств*. М.: Мир. 455 с.
Bourbaki, N. (1965). *Set Theory*. Moscow. 455 p.

Кононенко, А. А., Кучкаров, З. А., Никаноров, С. П., Никитина, Н. К. (2008). *Технология концептуального проектирования*. Никаноров, С. П. (ред.). М.: Концепт. 580 с.
Kononenko, A. A., Kouchkarov, Z. A., Nikanorov, S. P., Nikitina, N. K. (2008). *Technology of Conceptual Design*. Nikanorov, S. P. (ed.). Moscow. 580 p.

Никаноров, С. (2013). *Введение в аппарат ступеней множеств*. М.: Концепт. 350 с.
Nikanorov, S. P. (2013). *Introduction to the Apparatus of Set Levels*. Moscow. 350 p.

Пономарёв, И. Н. (2007). *Введение в математическую логику и роды структур: Учебное пособие*. М.: МФТИ. 244 с.

Ponomarev, I. N. (2007). *Introduction to Mathematical Logic and Species of Structures: Textbook*. Moscow. 244 p.

Erwig, M. (2001). Inductive graphs and functional graph algorithms. *Journal of Functional Programming*. Vol. 11. No. 5. Pp. 467-492.

Martin-Löf, P. (1984). *An Intuitionistic Type Theory. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980*. Napoli. Bibliopolis. 91 p.

Martin-Löf, P. (1987). Truth of a Proposition, Evidence of a Judgement, Validity of a Proof. *Synthese*. Vol. 73. Pp. 407-420. DOI: 10.1007/BF00484985.

Univalent Foundations Program (2013). *Homotopy Type Theory. Univalent Foundations of Mathematics*. Institute for Advanced Study. 589 p. Available at: http://homotopytypetheory.org/book.

Vezzosi, A., Mörtberg, A., Abel, A. (2021). Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *Journal of Functional Programming*. Vol. 31. P. e8. DOI: 10.1017/S0956796821000034.

Сведения об авторе / Information about the author

Доманов Олег Анатольевич — кандидат философских наук, старший научный сотрудник Института философии и права СО РАН, г. Новосибирск, Николаева, 8, e-mail: domanov@philosophy.nsc.ru, https://orcid.org/0000-0003-0057-3901.

Статья поступила в редакцию: 15.08.2025
После доработки: 04.09.2025
Принята к публикации: 15.09.2025

Oleg Domanov — Candidate of Philosophical Sciences, Senior Researcher of the Insitute of Philosophy and Law of the Siberian Branch of the Russian Academy of Sciences, Novosibirsk, Nikolaeva Str., 8, e-mail: domanov@philosophy.nsc.ru, https://orcid.org/0000-0003-0057-3901.

The paper was submitted: 15.08.2025
Received after reworking: 04.09.2025
Accepted for publication: 15.09.2025