УДК 1.168; 81:1

К ТЕОРЕТИКО-ТИПОВОЙ ФОРМАЛИЗАЦИИ СИТУАЦИОННОЙ СЕМАНТИКИ

О. А. Доманов
Институт философии и права СО РАН (г. Новосибирск)
odomanov@gmail.com

Аннотация. Ситуационная семантика является эффективным инструментом анализа семантических аспектов естественного языка с явной зависимостью от контекста, таких как референциальная непрозрачность доксических контекстов и пр. Использование в ней теоретико-типовых подходов не только во многих случаях делает её формализм более удобным, но и упрощает её перенос в компьютерные системы, более конкретно — формализацию в функциональных языках программирования. В статье представлен прототип теоретико-типового языка ситуационной семантики, реализованного в языке Racket. Описаны основные подходы, способы разрешения некоторых проблем формальной семантики, а также проблемы, требующие решения.

Ключевые слова: ситуационная семантика, теория типов, функциональные языки.

Для цитирования: Доманов, О. А. (2021). К теоретико-типовой формализации ситуационной семантики. *Respublica Literaria*. Т. 2. № 3. С. 32-41. DOI: 10.47850/RL.2021.2.3.32-41.

ON TYPE-THEORETICAL FORMALIZATION OF SITUATION SEMANTICS

O. A. Domanov
Institute of philosophy and Law SB RAS (Novosibirsk)
odomanov@gmail.com

Abstract. Situation semantics is an effective instrument for analysing semantical aspects of natural languages with explicit dependence on contexts, like referential opacity of belief contexts etc. Making use of type-theoretical approaches not only makes its formalism more practical in many ways, but also facilitates its migration to computer systems, specifically, the formalization in functional programming languages. The article deals with a prototype of the type theoretical language of situation semantics, implemented on the basis of the language Racket. It decribes principal approaches, methods of solving some problems of formal semantics as well as issues that need to be addressed.

Keywords: situation semantics, type theory, functional languages.

For citation: Domanov, O. A. (2021). On Type-Theoretical Formalization of Situation Semantics. *Respublica Literaria*. Vol. 2. no. 3. pp. 32-41, DOI: 10.47850/RL.2021.2.3.32-41.

Одна из главных проблем современной формальной семантики состоит в надлежащем учёте контекстуальной информации, особенно при наличии различных систем мнения (проблема референции в различных контекстах). Оценка семантических значений языковых выражений может производиться в разных контекстах, в разных ситуациях. Более того, многие из них требуют одновременного учёта нескольких несогласованных ситуаций (вероятно, наиболее известный пример здесь принадлежит Куайну [Quine, 1956]). Большая часть «загадок Фреге» [Salmon, 1986] относится к этому классу. Ситуационная семантика, предложенная Барвайсом и Перри [Barwise and Perry, 1983], является одной из удачных попыток разрешения проблем учёта контекстов. Однако в оригинальном варианте она использует в качестве средства формализации традиционную

формальную логику, что приводит к некоторым трудностям, из которых одной из существенных является сложность компьютеризации. В то же время, логические системы, опирающиеся на теорию типов, в последнее время показали свою эффективность именно в части удобства разработки компьютерных систем работы с доказательствами (proof assistants — наиболее известны Coq [Bertot and Castéran, 2004] и Agda [Norell, 2009], однако их в настоящее время имеется большое количество). Поскольку теория типов существенным образом ориентирована на представление знания (в виде суждений), она кажется удачным инструментом описания ситуаций, таких как системы мнений или знаний. Это обстоятельство было замечено несколько лет назад, и в настоящее время имеется несколько попыток теоретико-типовой формализации контекстов, [Cooper, 2005, 2017; Cooper and Ginzburg, 2015]. Они демонстрируют свою успешность в представлении анафор и других контекстно-зависимых явлений. Однако предлагаемый ими инструмент — TTR (Type theory with records, теория типов с записями) — всё же недостаточен для полного учёта всех аспектов ситуационной семантики. Более подходящим инструментом является понятие модуля, которое помимо объектов может также содержать, например, дефиниции и поэтому более адекватно для представления систем убеждений. Модуль представляет собой обособленную часть программы и поэтому, в идеале, может содержать всё, что может содержать программа вообще[1]. Модули широко используются в компьютерных науках, однако, разумеется, это использование ориентировано на цели самих этих наук, а не формальной семантики вообще. В частности, набор возможных операций с модулями ограничен, их, как правило, можно объединять, но затруднительно или невозможно сравнивать, определять подмодели и пр. Кроме того, невозможно учесть несогласованность или даже противоречивость модулей, что сплошь и рядом случается при моделировании систем убеждений. В настоящей статье предлагается набросок теоретико-типового языка, основанного на понятия модуля. Это язык описания ситуаций, операции над которыми проводятся на уровне синтаксиса. В статье представлено предварительное описание языка и рассмотрены способы, каким он позволяет разрешить некоторые проблемы формальной семантики.

Теория типов и представление знания о ситуации

Как логическая система, теория типов различает пропозиции и суждения. Пропозиция понимается как тип (или множество, что в нашем контексте одно и то же) её доказательств. Истина определяется через доказательство: пропозиция истинна, если она имеет доказательство. Тип определяется («объясняется», как выражается Пер Мартин-Лёф) путём объяснения (или понимания) того, что может служить элементом, термом или объектом данного типа. Основным видом суждения является суждение об отнесении определённого объекта к определённому типу (например, о том, что данный объект является доказательством данной пропозиции). Это суждение, по существу, является онтологическим. Его выносит субъект, опознающий некоторый существующий объект как относящийся к соответствующему типу (об онтологических и эпистемологических аспектах теории типов см. подробнее [Martin-Löf, 1987]). В то же время, это

[1]Существует также подход, объединяющий записи и модули в одно целое [Rossberg, 2015]. Это вариант языка ML, в котором записи, а также кортежи и структуры, задаются как вариант одной и той же формы — модуля. К сожалению, однако, как и в ML, в этом языке отсутствуют зависимые типы, что делает его неудобным для использования в формализации естественного языка.

суждение знания о некоторой ситуации: мы утверждаем, что в этой ситуации имеется объект данного типа. Это позволяет нам представить ситуацию (фактически — знание о ситуации) как набор суждений вида $a : A$: «(в ситуации) имеется объект a типа A». *Сигнатурой* называется набор суждений вида $l : A$, где l — имя (называемое также меткой; неформально, оно обозначает «роль» объекта в ситуации), а A — тип. Мы считаем, что порядок суждений в сигнатуре не имеет значения. Сигнатура задаёт тип ситуаций, таких, что для каждого суждения сигнатуры $l : A$, в них существует объект с именем l типа A. Будем говорить, что такая ситуация *выполняет* данную сигнатуру. При этом сигнатура даёт необходимую, но не достаточную информацию о ситуации: в последней может быть произвольное количество других объектов, не указанных в сигнатуре. Кроме того, ситуация может также содержать информацию другого рода, такую как определения или способы построения самих объектов. Будем далее записывать ситуацию как $[l_1 : A_1, l_2 : A_2, \ldots]$. Здесь l_i — имена (метки) объектов ситуации, а A_i — типы, к которым они должны относиться.

К особому роду относятся ситуации, в которых какие-то из типов являются синглетонами, то есть типами, состоящими из одного объекта. Это, очевидно, означает, что все выполняющие такую сигнатуру ситуации ситуации содержат эти объекты. Эти случаи будем записывать в виде $[\ldots, l_i = a_i : A_i, \ldots]$, что означает, что объект с именем l_i равен a_i. В этой записи будем также опускать A_i, если это не приводит к неясностям.

Будем, далее, предполагать, что на типах может быть задано отношение подтипа. Обозначение $A <: B$ означает «A является подтипом B». Тип A является подтипом типа B, если всякий объект типа A является одновременно объектом типа B (более подробно о подтипах см. [Pierce, 2002, pp. 181 sqq.]). Ситуации допускают два вида подтипов: подтипирование в ширину (width subtyping) и подтипирование в глубину (depth subtyping) [Pierce, 2002, p. 183]. Подтипирование в ширину возникает при добавлении суждений к сигнатуре. Действительно, такое добавление сужает список ситуаций, выполняющих сигнатуру, поскольку на них накладывается дополнительное условие. Подтипирование в глубину возникает, когда мы заменяем какие-то типы сигнатуры на их подтипы. Действительно, такая замена также приводит к сужению набора ситуаций, выполняющих сигнатуру. Нетрудно видеть, что ситуации с синглетонами являются частным случаем подтипа, поскольку синглетон с объектом типа A является подтипом типа A. Это аналогично пониманию элемента множества в теории категорий, где элемент определяется как морфизм из синглетона (точнее, конечного объекта), то есть как подмножество, состоящее из одного элемента. Таким образом, даже конкретные ситуации мы понимаем как тип (по-существу, тип синглетона). С учётом сказанного, мы будем понимать ситуации $[x : A, \ldots]$ как «имеется некоторый (неуказанный) объект x типа A,…», а ситуации $[x = a : A, \ldots]$ — как «имеется (определённый) объект a типа A,…». Например, $[l_1 = a_1, l_2 : A_2, l_3 = a_3, \ldots]$ соответствует «конкретизации» ситуации $[l_1 : A_1, l_2 : A_2, l_3 : A_3, \ldots]$, такой, что в ней имеется известный объект a_1, неизвестный объект типа A_2, известный объект a_3 и т. д.

Семантика естественного языка

Как мы видели, ситуационная семантика описывает «информационный» аспект языка: под ситуацией мы понимаем знание о ситуации. Соответственно, ситуационная семантика естественного языка должна переводить языковые конструкции в ситуации, описывающие знание, выраженное этими конструкциями. Рассмотрим в качестве примера простой фрагмент естествен-

S	::=	NP VP	*предложение*
NP	::=		*группа существительного*
	|	PN	*имя собственное*
	|	‘a’ CN	*неопределённый артикль*
	|	‘the’ CN	*определённый артикль*
	|	‘every’ CN	*«каждый»*
VP	::=		*группа глагола*
	|	VI	*непереходный глагол*
	|	VT NP	*переходный глагол*

Рис. 1. Грамматика фрагмента языка

ного языка (английского) и его интерпретацию в терминах нашей семантики. За основу положено теоретико-типовое построение грамматики А. Ранта [Ranta, 1994].

Пусть язык содержит синтаксические категории S (предложение), CN (имя нарицательное), PN (имя собственное), VI (непереходный глагол), VT (переходный глагол), артикли *a*, *the* и слово *every*. Правила грамматики изображены на рис. 1.

Будем интерпретировать языковые выражения как сигнатуры, то есть типы ситуаций. Тогда CN соответствует типу в смысле теории типов, то есть сигнатуре вида $[A : \mathit{Type}]$. Например, слово «cow» описывает ситуации, в которых известно, что такое корова. При этом сигнатура не определяет, имеется ли в ситуации хотя бы одна корова; ситуации «There is a cow» имеют сигнатуру $[\mathrm{cow} : \mathit{Type}, x : \mathrm{cow}]$. Далее, PN соответствует ситуации с одним объектом, то есть $[A : \mathit{Type}, l = a : A]$. Глаголы VI соответствуют сигнатуре $[A : \mathit{Type}, v : A \to \mathit{Type}, x : A, \alpha : v(x)]$. Иначе говоря, ситуация VI содержит тип A, зависимый от него тип $v : A \to \mathit{Type}$, объект $x : A$ и доказательство того, что $v(x)$. В более традиционных терминах эта сигнатура описывает пропозицию, основанную на зависимом типе v. Её можно понимать как параметризованную A и v функцию, сопоставляющую каждому $x : A$ доказательство того, что $v(x)$. Например, глагол «run» (зависящий от «animal») имеет сигнатуру $[\mathrm{animal} : \mathit{Type}, \mathrm{run} : \mathrm{animal} \to \mathit{Type}, x : \mathrm{animal}, \alpha : \mathrm{run}(x)]$ и описывает ситуации типа «An animal runs». Аналогично, переходные глаголы VT интерпретируются как ситуации вида $[A : \mathit{Type}, B : \mathit{Type}, v : A \to B \to \mathit{Type}, x : A, y : B, \alpha : v(x, y)]$.

Введём обозначения:

$$\mathtt{CN}(A) = [A : \mathit{Type}],$$
$$\mathtt{PN}(A, a) = [A : \mathit{Type}, x = a : A],$$
$$\mathtt{VI}(A, v, x, \alpha) = [A : \mathit{Type}, v : A \to \mathit{Type}, x : A, \alpha : v(x)],$$
$$\mathtt{VT}(A, B, v, x, y, \alpha) = [A : \mathit{Type}, B : \mathit{Type}, v : A \to B \to \mathit{Type}, x : A, y : B, \alpha : v(x, y)].$$

Для синтаксической категории $\mathtt{C}(\dots, x, \dots) = [\dots, x : A, \dots]$ будем писать $\mathtt{C}\langle \dots, a, \dots \rangle$ для конкретизации $[\dots, x = a, \dots]$ и $\mathtt{C}\langle \dots, \_, \dots \rangle$ для конкретизации $[\dots, x : A, \dots]$. Будем также для простоты опускать конечные знаки подчёркивания. Например, для $\mathtt{C}(A, x, y) = [A : \mathit{Type}, x : A, y : A]$ выражение $\mathtt{C}\langle \_, a \rangle$ обозначает $[A : \mathit{Type}, x = a, y : A]$.

Рассмотрим предложения с непереходными глаголами, то есть предложения вида NP VI.

Предложения вида PN VI интерпретируются следующим образом:

$$[\![\texttt{PN}\langle A, a\rangle\ \texttt{VI}\langle A, v\rangle]\!] = \texttt{VI}\langle A, v, a\rangle = [A = A, v = v, x = a, \alpha : v(a)]$$

(обратите внимание, что здесь в записи $A = A$ слева стоит имя или метка, а справа — объект, в данном случае, множество A; я надеюсь, что это не введёт читателей в заблуждение). Например, предложение «Ortcutt runs» соответствует типу ситуаций, в которых имеются тип «man», одноместный предикат «run» и объект «Ortcutt» типа «man», а также (неспецифицированное) доказательство того, что Орткут бежит. Заметим, что в правильно построенном предложении типы A и имена x в PN и VI должны совпадать (с точностью до подтипа).

Предложения с неопределённым артиклем интерпретируются следующим образом:

$$[\![\text{A}\ \texttt{CN}\langle A\rangle\ \texttt{VI}\langle A, v\rangle]\!] = \texttt{VI}\langle A, v\rangle = [A = A, v = v, x : A, \alpha : v(x)].$$

Это тип ситуаций, содержащих известные тип A и предикат v вместе с неуказанными объектом $x : A$ и доказательством того, что $v(x)$. Упрощённо говоря, это тип пар, состоящих из объекта и доказательства того, что для этого объекта истинен предикат v. Например, предложение «A man runs» соответствует типу ситуаций, в которых имеется (известен) тип «man», одноместный предикат «run», а также неуказанные объект типа «man» и доказательство того, что этот объект бежит.

Предложения с определённым артиклем интерпретируются следующим образом:

$$[\![\text{The}\ \texttt{CN}\langle A\rangle\ \texttt{VI}\langle A, v\rangle]\!] = \texttt{VI}\langle A, v, e\rangle = [A = A, v = v, x = e, \alpha : v(e)],$$

где e — объект, известный из контекста, на который указывает определённый артикль.

Наконец, универсальные предложения интерпретируются как функции

$$[\![\text{Every}\ \texttt{CN}\langle A\rangle\ \texttt{VI}\langle A, v\rangle]\!] = \lambda(x : A).[A = A, v = v, x = x, \alpha : v(x)].$$

Таким образом, это функция, которая для каждого $x : A$ задаёт ситуацию с известными типом A и предикатом v, а также объектом x и неуказанным доказательством того, что $v(x)$. Например, предложение «Every man runs» интерпретируется как функция, позволяющая для каждого человека построить ситуацию, в которой этот человек бежит.

Предложения с переходными глаголами интерпретируются путём построения глагольных групп VP. Например, для глагола вида $\texttt{VT} = [A : \mathit{Type}, B : \mathit{Type}, v : A \to B \to \mathit{Type}, x : A, y : B, \alpha : v(x, y)]$ имеем, в частности:

$$\texttt{VP} = \texttt{VT}\langle A, B, v\rangle\ \texttt{PN}\langle B, b\rangle = [A = A, B = B, v = v, x : A, y = b, \alpha : v(x, b)].$$

Как видно, это выражение имеет форму непереходного глагола $[A = A, v = v', x : A, \alpha : v'(x)]$ при $v'(x) = v(x, b)$, поэтому дальнейшая интерпретация проводится по правилам для VI.

Как и в семантике Монтегю, интерпретация артикля *a* и слова *every* соответствует экзистенциальному и универсальному квантору. Действительно, зависимый тип $B(x)$, зависящий от типа A, можно понимать как тип ситуаций

$$[A : \mathit{Type}, B : A \to \mathit{Type}, x : A, \alpha : B(x)].$$

Тогда экзистенциальные и универсальные кванторы (Σ- и Π-типы) интерпретируются следующим образом:

$$\Sigma(x : A)B = [A = A, B = B, x : x, \alpha : B(x)]$$
$$\Pi(x : A)B = \lambda(x : A).[A = A, B = B, x = x, \alpha : B(x)].$$

Непереходные глаголы `VI` соответствуют, как видно, зависимым типам (ср. с грамматикой А. Ранта [Ranta, 1994]).

Таким образом, выражения выделенного нами простого фрагмента естественного языка интерпретируются в языке ситуационной семантики. В частности, предложения (или пропозиции) оказываются типами ситуаций, а их доказательства — конкретными ситуациями (в нашем понимании — синглетонами). Аналогичным образом можно пытаться интерпретировать более полные фрагменты естественного языка, но это выходит за рамки настоящей статьи. В любом случае, имеющиеся разработки в области теоретико-типовой грамматики (прежде всего, Grammatical Framework А. Ранта [Ranta, 2011]) позволяют надеяться здесь на успех.

Контексты знания и ситуации

В этом разделе в качестве инструмента формализации мы будем использовать язык программирования Racket. Это функциональный язык семейства LISP, специально нацеленный на разработку языков предметных областей. Связь функциональных языков — которые являются расширенными вариантами λ-исчисления — с логикой и теорией типов установлена довольно давно (идея восходит к [Lambek and Scott, 1986]). В нашем случае их выбор для формализации обусловлен тем, что они, как правило, обладают развитыми средствами для разбиения программ на независимые блоки (модули) с регулировкой видимости содержимого этих блоков из других частей программы. Модуль является, вероятно, наиболее близким к ситуации понятием компьютерных наук. Модуль это независимый фрагмент программы, содержащий определения, декларации, вычисления и пр. Будучи таковым, он естественным образом подходит для формализации систем мнений и контекстов. Во многих языках программирования имеются развитые средства для работы с модулями, уже позволяющие формализовать эффекты непрозрачности (вероятно, наиболее разработанный язык для работы с модулями содержится в языке ML, мы будем опираться на его идеи). Как правило, содержимое модуля не видно из других частей программы без специального «открытия» этого содержимого. При открытии идентификаторы модуля могут быть переименованы, модуль может быть открыт частично и т. д., так что «видимость» содержимого модуля может настраиваться в широких пределах, причём быть различной в различных частях программы (в частности, в других модулях). Однако модули разработаны прежде всего для нужд программирования, в частности, для организации модульной сборки программ. Многие эффекты, вызванные непрозрачностью референции в контекстах при этом просто не встречаются. Поэтому при разработке формальной семантики соответствующие средства работы с модулями должны быть добавлены. Ниже мы рассмотрим, как это может выглядеть в языке Racket. В оставшейся части статьи описан прототип языка `SitSem`, опирающийся, в свою очередь, на `Cur`, язык с зависимыми типами, являющийся, в целом, исчислением индуктивных конструкций (CIC — Calculus of Inductive Constructions [Paulin-Mohring, 2015], основной язык системы Coq). Формализацию можно найти по адресу `https://github.com/odomanov/ttsemantics/tree/master/Racket/SitSem`.

Итак, мы формализуем ситуации как модули. Модули являются стандартной конструкцией языка Racket и в упрощённом виде задаются командой

```
(module S
  (provide a,...)
  body),
```

где `S` — имя модуля, после `provide` перечисляются имена объектов модуля, которые могут быть открыты как видимые снаружи модуля, а `body` — набор определений, деклараций и пр., составляющих содержание модуля. Ситуации могут комбинироваться, образуя сложные ситуации. Мы выделим комбинирование двух видов. При первом из них всё содержимое одной ситуации добавляется к другой, как будто оно изначально в ней присутствовало — в этом смысле добавляемая ситуация «прозрачна». При втором виде добавляемая ситуация тем или иным образом не согласована с исходной, что требует специального указания способа, каким её содержимое должно включаться (она, в этом смысле «непрозрачна»). Например, в известном примере Куайна [Quine, 1956] Ральф опознаёт одного и того же человека (которого мы знаем под именем Орткут) в разное время как разных людей, причём в одном случае считает его шпионом, а в другом — нет. В ситуации Ральфа, представляющей систему его убеждений, имеется два человека *mh* и *mb*, тогда как в нашей системе убеждений мы знаем, что это один и тот же человек. Мы не можем просто присоединить убеждения Ральфа к нашим, поскольку они существенным образом отличаются от них. Так происходит во всех случаях, когда мы работаем с контекстами знания, отличного от нашего, с которыми мы не можем просто согласиться и присоединить к нашему знанию.

В итоге, в `SitSem` предусмотрены три способа работы с включёнными ситуациями. Первый способ является стандартным в Racket:

```
(require 'S).
```

Эта команда добавляет содержимое ситуации (модуля) `S` к текущей ситуации так, как будто оно изначально в ней присутствовало (точнее, добавляются лишь определения имён, перечисленных в директиве `provide`, но мы опустим этот факт для простоты).

Во втором способе определена команда

```
(embed-sit 'S)
```

которая импортирует (открывает) содержимое ситуации `S`, добавляя ко всем идентификаторам префикс `S` с точкой, например, `S.x` вместо `x` и т. д. Тем самым, знание `S` содержится в нашем знании, будучи явно помеченным как знание `S`, и мы можем проводить вычисления из `S`, находясь в нашей ситуации. Этот способ, однако, не позволяет рассматривать случай описанного выше рассогласования (играющий, заметим, ключевую роль в возникновении семантических парадоксов и «загадок»).

Наконец, для работы третьим способом в `SitSem` определена команда

```
(with-link 'S ([a b] ...)
  body)
```

которая проводит вычисления «внутри» ситуации `S`, устанавливая связь идентификаторов нашей ситуации `a`, … с идентификаторами `b`, … ситуации `S`. Команда проводит вычисления, обозначенные здесь как `body`, и выдаёт в качестве значения результат последнего вычисленного выражения.

При этом все ссылки внутри `body` на перечисленные имена нашей ситуации заменяются на ссылки внутри `S`. Команда `with-link` допускает, чтобы одно и то же имя из нашей ситуации соответствовало нескольким именам в ситуации `S`. В этом случае команда вычисляет возможные комбинации замен и выдаёт в качестве результата не одно значение, а множество. Тем самым моделируется семантическая неоднозначность выражения.

Рассмотрим пример Куайна. Ситуация Ральфа (Ralph's belief) определяется как модуль, в котором определён тип `man` с двумя термами `mh` и `mb` (а также предикат «быть шпионом» и доказательства для него, но мы их для простоты опускаем):

```
(module RB "SitSem.rkt"
  (provide man mh mb)
  (define-datatype man : Type
    [mh : man]
    [mb : man]))
```

Внутри нашей ситуации тоже определён тип `man`, но только с одним термом `o`, обозначающим Орткута:

```
(define-datatype man : Type
  [o : man])
```

Действуя вторым из описанных способов и включив ситуацию `RB` в нашу ситуацию:

```
(embed-sit 'RB)
```

мы получим значения `RB.mh`, `RB.mb` в нашей ситуации, после чего можем в явном виде установить их связь с `o` (определив соответствующее отношение) и, например, установить, что можно вычислить два возможных ответа на вопрос «Считает ли Ральф Орткута шпионом?» (см. подробности в указанной выше формализации).

Действуя третьим способом, вычислим значение `o`, установив связь имён:

```
(with-link RB ([o mh] [o mb])
  o)
```

Это соответствует вычислению `o` «в контексте Ральфа», то есть отвечает на вопрос «Кто является Орткутом для Ральфа, с точки зрения Ральфа?». В результате вычисления мы получим два возможных значения, а именно `mh` и `mb`, для двух вариантов переноса Орткута в контекст Ральфа, как и требуется. Аналогично вычисляются более сложные выражения с использованием `o`.

Таким образом, модули позволяют нам моделировать эффекты прозрачности и непрозрачности контекстов, существенные для семантики естественного языка. В случае функциональных типизированных языков они также, как можно надеяться, предоставляют адекватный инструмент для представления ситуации в ситуационной семантике. Однако, будучи разработанными в рамках компьютерных наук, они ориентированы на потребности программирования и должны быть дополнены более эффективными средствами соотнесения модулей друг с другом, их объединения, сравнения и т. д. К сожалению, имеющиеся языки не в состоянии предоставить весь спектр необходимых свойств, и может быть поставлена задача разработки такого языка.

Список литературы / References

Barwise, J. and Perry, J. (1983). *Situations and Attitudes*. MIT Press.

Bertot, Y. and Castéran, P. (2004). *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. XXV, 472 p. DOI: 10.1007/978-3-662-07964-5.

Cooper, R. (2005). Austinian Truth, Attitudes and Type Theory. *Research on Language and Computation*. Vol. 3. no. 2. pp. 333-362. DOI: 10.1007/s11168-006-0002-z.

Cooper, R. (2017). Adapting Type Theory with Records for Natural Language Semantics. In Chatzikyriakidis, S. and Luo, Z. (eds.). *Modern Perspectives in Type-Theoretical Semantics*. Studies in Linguistics and Philosophy 98. Springer International Publishing. pp. 71-94. DOI: 10.1007/978-3-319-50422-3_4.

Cooper, R. and Ginzburg, J. (2015). Type Theory with Records for Natural Language Semantics. In Lappin, S. and Fox, C. (eds.). *The Handbook of Contemporary Semantic Theory*. Blackwell Handbooks in Linguistics. Wiley. pp. 376-407. DOI: 10.1002/9781118882139.ch12.

Lambek, J. and Scott, P. J. (1986). *Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics 7. Cambridge. Cambridge University Press.

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.

Norell, U. (2009). Dependently Typed Programming in Agda. In Koopman, P., Plasmeijer, R., and Swierstra, D. (eds.). *Advanced Functional Programming: 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*. Lecture Notes in Computer Science 5832 : Theoretical Computer Science and General Issues. Berlin, Heidelberg. Springer-Verlag Berlin Heidelberg. pp. 230-266. DOI: 10.1007/978-3-642-04652-0_5.

Paulin-Mohring, C. (2015). Introduction to the Calculus of Inductive Constructions. In Paleo, B. W. and Delahaye, D. (eds.). *All about Proofs, Proofs for All*. Studies in Logic (Mathematical logic and foundations) 55. [Online]. Available at: https://hal.inria.fr/hal-01094195 (Accessed: 30 July 2021).

Pierce, B. C. (2002). *Types and Programming Languages*. Cambridge. The MIT Press. 648 p.

Quine, W. V. O. (1956). Quantifiers and Propositional Attitudes. *Journal of Philosophy*. Vol. 53. no. 5. pp. 177-187.

Ranta, A. (1994). *Type-theoretical grammar*. Indices 1. Clarendon Press. 226 p.

Ranta, A. (2011). *Grammatical Framework. Programming with Multilingual Grammars*. CSLI studies in computational linguistics. CSLI Publications, Center for the Study of Language and Information. XII, 331 p.

Rossberg, A. (Aug. 2015). 1ML – core and modules united (F-ing first-class modules). *ACM SIGPLAN Notices*. Vol. 50. pp. 35-47. DOI: 10.1145/2858949.2784738.

Salmon, N. U. (1986). *Frege's Puzzle*. Cambridge, Mass. Bradford Books/The MIT Press. 195 p.

Сведения об авторе / Information about the author

Доманов Олег Анатольевич — кандидат философских наук, доцент, старший научный сотрудник Института философии и права Сибирского отделения Российской академии наук, г. Новосибирск, Николаева, 8, e-mail: odomanov@gmail.com, http://orcid.org/0000-0003-0057-3901.

Статья поступила в редакцию 15.08.2021
После доработки 28.08.2021
Принята к публикации 10.09.2021

Oleg Domanov — Candidate of Philosophy, associate Professor, 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: odomanov@gmail.com, http://orcid.org/0000-0003-0057-3901.

The paper was submitted 15.08.2021
Received after reworking 28.08.2021
Accepted for publication 10.09.2021