Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»
Факультет Санкт-Петербургская школа физико-математических и
компьютерных наук
Департамент информатики
Основная образовательная программа
Прикладная математика и информатика

Кайсин Илья Сергеевич

Задача обитаемости в системах типов низкого ранга

Выпускная квалификационная работа

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

Научный руководитель:
к. ф.-м. н., профессор Москвин Д. Н.

Рецензент:
Пеленицын А. М

Санкт-Петербург
2019

Оглавление

Аннотация

Типы являются важной абстракцией в языках программирования. Ограничивая множество корректных программ, они позволяют программистам писать более безопасный код и избегать ошибок. Теория типов – математический аппарат, формализующий данную область. Одним из базовых вопросов теории типов является *задача обитаемости*: «Существует ли терм (выражение в языке) заданного типа?». Задача обитаемости может быть поставлена в различных *системах типов* (моделях, описывающих часть языка программирования, относящуюся к типам). Цель данной работы – решить проблему обитаемости для некоторой определённой системы типов, а именно для просто типизированного лямбда исчисления с пересечениями второго ранга и подтипизацией. Недавние работы показывают, что задача обитаемости неразрешима в системах без ограничения на ранг [18] и EXPSPACE-полной в системе с пересечениями второго ранга, но без подтипизации [12,19]. Однако система и с пересечениями второго ранга, и с подтипизацией до сих пор не была изучена. В данной работе мы показываем, что задача обитаемости разрешима в этой системе. Для этого мы приводим алгоритм, решающий её, доказывая его корректность, полноту и завершаемость. Помимо этого, работа закрывает некоторые неточности предыдущих статей и предлагает новые подходы к реализации описанных в них алгоритмов.

Ключевые слова: теория типов, лямбда исчисление, типы-пересечения, ранг типа, подтипизация, проблема обитаемости.

Types are an important abstraction in programming languages. They enable to program in a safer way and avoid bugs setting restriction on programs and distinguishing correct and incorrect ones. Type theory is the mathematical apparatus for this domain, one of whose basic questions is the inhabitation, usually defined as follows: «does there exist a term (some expression in the language) of a given type?». The question could be raised in different type systems (models of type-related programming languages parts), the goal of this project is to solve this question in some specific systems. Namely, simply typed lambda calculus with type intersection of rank two (type system restriction) and subtyping. Recently presented works show that the problem is undecidable in type systems without rank limitation [18] and EXPSPACE-complete without subtyping [12, 19], but the system with both of them is not studied yet. The approaches include an adaptation of an existing in similar type systems inhabitation algorithm, proof of its correctness and soundness, implementing and testing the algorithm in Haskell programming language. The preliminary results also cover several flaws in the previous works and suggest that the inhabitation algorithm scheme could be improved in terms of efficiency.

Keywords: type theory, lambda calculus, intersection types, type rank, subtyping, type inhabitation problem.

Введение

Почти во всех языках программирования в том или ином виде представлены типы. Чем сильнее система типов языка, тем больше ограничений она накладывает на корректные программы и тем больше ошибок может быть найдено на *этапе компиляции*, что позволяет уменьшить число потенциальных ошибок на *этапе исполнения*, когда программа работает в реальной среде, и цена каждой ошибки велика. Изучением систем типов и их свойств занимается теория типов.

Помимо языков программирования, теория типов применяется в системах интерактивных доказательств – средствах, позволяющих математикам полуавтоматически формулировать и доказывать теоремы на некотором формальном языке; используется для описания оснований математики как альтернатива наивной теории множеств.

Наиболее развитую систему типов имеют функциональные языки программирования. Основанием этой системы является типизированное *лямбда исчисление*.

Лямбда исчисление было впервые введено Алонзо Чёрчем в 1930-х годах и на данный момент имеет множество модификаций. Самая базовое из типизированных лямбда исчислений – просто типизированное лямбда исчисление. Типы в этой системе конструируются из двух составных частей: типовых переменных (например, `Int` – тип целых чисел или `String` – тип строк) и операции (типового конструктора) «$\to$». «$\to$» представляет функциональный тип. Например, тип «`Int` $\to$ `String`» – это тип функций, отображающих целые числа в строки.

Система типов, рассматриваемая в данной работе, содержит ещё один типовой конструктор – «$\wedge$», представляющий пересечение двух типов. Лямбда исчисление с пересечениями известно с 1970-х. Значимость этой системы подтверждается многочисленными работами, посвященными ей: модели лямбда исчисления [1, 4], проблемы нормализации и оптимальной редукции [14, 15], вывод типов и компиляция [10, 20].

Другим важным аспектом систем типов с пересечениями является их логическая интерпретация. Изоморфизм Карри-Говарда ставит в соответствие типам логические высказывания, а термам – доказательства этих высказываний. Благодаря такой интерпретации, можно считать, что решая задачу обитаемости, мы по высказыванию изучаем его доказательства – ищем их или проверяем их наличие. Просто типизированное лямбда исчисление с пересечениями при этом соответствует пропозициональной логике с пересечениями. Неразрешимость задачи обитаемости в этой системе [17] говорит о её большой выразительной силе. Поэтому, в частности, важно разграничить случаи разрешимости и неразрешимости.

Объектом исследования данной работы является проблема обитаемости в просто типизированном лямбда исчисление с пересечениями и подтипизацией. Интересующий нас вопрос – разрешима ли эта проблема в случае, если система ослабляется определёнными ограничениями на типы. А именно, для типов с пересечениями в [13]

вводится понятие *ранга* – глубины вложенности «$\wedge$» в тип в качестве левого аргумента «$\rightarrow$». Оказывается, что если ограничить ранг типов двумя, то в системе без подтипизации проблема разрешима, то есть существует алгоритм, по типу ранга $\leqslant 2$ предъявляющий терм заданного типа. Добавление в систему подтипов изменяет систему, значительно расширяя множество допустимых типизаций. Однако данная работа показывает, что задача обитаемости разрешима и в системе с подтипами.

1. Обзор предметной области

1.1. Обзор литературы

Впервые типы были введены в логику Бертраном Расселом в начале XX века. Алонзо Чёрч ввёл их в лямбда исчисление в 1940 [3], построив так называемое просто типизированное лямбда исчисление. Альтернативная версия этой системы была предложена Хаскелем Карри в 1969 [6]. Системы Карри и Чёрча отличаются тем, что в системе Чёрча тип «встроен» в лямбда-терм. Это означает, что любой терм типизируется уникальным образом. В системе Карри один терм может иметь несколько типов, если соответствующие выводимы из аксиом и правил. Для задачи обитаемости различия этих двух систем не играют большой роли, поскольку тип всегда известен.

Системы типов с операцией пересечения получили своё развитие в начале 1980-х [4, 5]. Позднее Хиндли исследовал свойства этой системы и формально описал её семантику [8]. Кроме того, в этой статье было введено понятие *подтипизации* ($\leqslant$), означающее с теоретико-множественной точки зрения «включение» одного типа в другой, и доказаны некоторые его свойства. Подтипизация и его свойства активно используются в настоящей работе.

Если говорить о проблеме обитаемости, первая работа, имеющая отношение к интересующей нас системе типов, появилась в 1979 году [16]. В этой статье доказывается, что для просто типизированного лямбда исчисления (без пересечений) задача обитаемости разрешима и является PSPACE-полной. В 1981 была доказана неразрешимость задачи выполнимости в пропозициональной интуиционистской логике второго порядка [7]. В соответствии с изоморфизмом Карри-Говарда, это означает неразрешимость задачи обитаемости в полиморфном лямбда исчислении (System F), где помимо типового конструктора «$\to$» вводится конструктор полиморфного типа «$\forall$».

Полиморфизм просто типизированного лямбда исчисления с пересечениями более сильный: «$\forall$» можно рассматривать как бесконечное пересечение по всем возможным типам. В [15] было показано, что в этой системе множество типизируемых термов и множество нормализуемых (имеющих нормальную форму) совпадают.

Неразрешимость задачи обитаемости для системы с пересечениями была показана в [17] (1999). Это означает, что данная система является слишком богатой в смысле выразительной мощности, и для существования алгоритмического решения задачи обитаемости необходимо ввести на систему некоторые ограничения. В [13] вводится понятие ранга типа – меры его сложности. Конструкция, приведённая в [17], к населению которой сводится проблема останова, содержит типы ранга $\leqslant 4$. Поэтому ограничения «ранг $\leqslant 4$» недостаточно, чтобы сделать проблему разрешимой.

В [11] было показана разрешимость задачи для другой версии системы с пересечениями и η – правилом (эквивалентно подтипизации). В этой системе нет ограничения

на ранг типов; она ослабляется за счёт того, что в нём запрещено применять одно из правил вывода, а именно так называемое правило введения пересечения 1

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau} \ (I\wedge) \tag{1}$$

1.2. Просто типизированное лямбда исчисление

1.2.1. Лямбда термы

1.2.2. Типизация

2. Система типов $\lambda_{\wedge\eta}$

Система типов $\lambda_{\wedge\eta}$ отличается от просто типизированного лямбда исчисления новым типовым конструктором: $\wedge$, соответствующим пересечению типов. Эту операцию можно понимать в теоретико-множественном смысле: типом $\sigma \wedge \tau$ типизируются такие и только такие термы, которые типизируются и σ, и τ. Правила вывода, соответствующие этому поведению, обозначаются $(I\wedge)$ и $(E\wedge)$ (введение пересечения и элиминация пересечения соответственно).

Кроме того, вводится ещё одно дополнительное правило (η), позволяющее проводить эта-экспансию.

Таким образом, система состоит из следующих правил вывода: Ax, $I \to$, $E \to$, $I\wedge$, $E\wedge$, η.

$$
\begin{aligned}
\frac{}{\Gamma, x\colon \tau \vdash x\colon \tau}\ (Ax) \\
\frac{\Gamma, x\colon \sigma \vdash M\colon \tau}{\Gamma \vdash (\lambda x.M)\colon \sigma \to \tau}\ (I \to) \\
\frac{\Gamma \vdash M\colon \sigma \to \tau \quad \Gamma \vdash N\colon \sigma}{\Gamma \vdash (MN)\colon \tau}\ (E \to) \\
\frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash M\colon \tau}{\Gamma \vdash M\colon \sigma \wedge \tau}\ (I\wedge) \\
\frac{\Gamma \vdash M\colon \sigma \wedge \tau}{\Gamma \vdash M\colon \sigma}\ (E\wedge) \\
\frac{\Gamma \vdash M\colon \sigma}{\Gamma \vdash (\lambda x.Mx)\colon \sigma}\ (\eta)
\end{aligned}
\tag{2}
$$

2.1. Подтипизация

Определим на типах отношение подтипизации. С теоретико-множественной точки зрения подтпизация соответствует отношению «быть подмножеством»: если терм типизируется σ, то он типизируется всеми надтипами σ, то есть такими τ, что $\sigma \leqslant \tau$. Отношение определим следующими аксиомами и правилами, аналогично определе-

нию из [8].

$$\frac{}{\sigma \leqslant \sigma}\ (A1)$$

$$\frac{}{\sigma \leqslant \sigma \wedge \sigma}\ (A2)$$

$$\frac{}{\sigma \wedge \tau \leqslant \sigma}\ (A3)$$

$$\frac{}{\sigma \wedge \tau \leqslant \tau}\ (A4)$$

$$\frac{}{(\sigma \to \tau_1) \wedge (\sigma \to \tau_2) \leqslant \sigma \to (\tau_1 \wedge \tau_2)}\ (A5) \qquad (3)$$

$$\frac{\sigma \leqslant \sigma' \quad \tau \leqslant \tau'}{\sigma \wedge \tau \leqslant \sigma' \wedge \tau'}\ (R1)$$

$$\frac{\sigma \leqslant \sigma' \quad \tau \leqslant \tau'}{\sigma' \to \tau \leqslant \sigma \to \tau'}\ (R2)$$

$$\frac{\tau_1 \leqslant \tau_2 \quad \tau_2 \leqslant \tau_3}{\tau_1 \leqslant \tau_3}\ (R3)$$

В [9] показано, что правило (η) может быть заменено следующим правилом:

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M : \tau}\ (\leqslant)$$

На самом деле, поскольку $\sigma \wedge \tau \leqslant \sigma$ и $\sigma \wedge \tau \leqslant \tau$, правило $(E\wedge)$ также избыточно. Таким образом, правила в этой системе следующие:

$$\frac{}{\Gamma, x\colon \tau \vdash x\colon \tau}\ (Ax)$$

$$\frac{\Gamma, x\colon \sigma \vdash M\colon \tau}{\Gamma \vdash (\lambda x.M)\colon \sigma \to \tau}\ (I \to)$$

$$\frac{\Gamma \vdash M\colon \sigma \to \tau \quad \Gamma \vdash N\colon \sigma}{\Gamma \vdash (MN)\colon \tau}\ (E \to) \qquad (4)$$

$$\frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash M\colon \tau}{\Gamma \vdash M\colon \sigma \wedge \tau}\ (I\wedge)$$

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M : \tau}\ (\leqslant)$$

Полученную систему будем называть $\lambda_{\wedge\leqslant}$. Она эквивалентна $\lambda_{\wedge\eta}$ в смысле типизации: утверждение типизации, верное в системе $\lambda_{\wedge\leqslant}$, верно в $\lambda_{\wedge\eta}$ и наоборот. Поэтому в контексте нашей задачи эти две системы полностью взаимозаменяемы.

Определим отношение эквивалентности на типах следующим образом.

Определение 1. $\sigma \sim \tau \iff \sigma \leqslant \tau \ и\ \tau \leqslant \sigma$

Благодаря правилу $(\leqslant)$, для эквивалентных типов верно следующее утверждение:

Лемма 2.1. *Множества термов, тпизируемых эквивалентными типами, в точности совпадают.*

В частности, это означает, что переход к эквивалентному типу не влияет на его обитаемость.

Легко видеть, что отношение «$\sim$» является отношением конгруэнтности (в смысле [2]) а именно, верна следующая Лемма:

Лемма 2.2. *Пусть $\alpha, \beta, \gamma, \delta$ – типы. Если $\alpha \sim \beta$ и $\gamma \sim \delta$, то $(\alpha \to \gamma) \sim (\beta \to \delta)$, а также $(\alpha \wedge \gamma) \sim (\beta \wedge \delta)$.*

Для удобства введём следующее обозначение:

Определение 2.

$$\hat{\tau} = \{\tau_i \mid \tau_1 \wedge \cdots \wedge \tau_k = \tau \text{ и } \tau_i \text{ не является пересечением ни для какого } i\}$$

То есть $\hat{\tau}$ – множество, полученное разделением всех пересечений на верхнем уровне τ.

В [8] показана следующая экивалентность:

Лемма 2.3. $\alpha \to \beta \wedge \gamma \sim (\alpha \to \beta) \wedge (\alpha \to \gamma)$

Применяя её многократно, мы можем привести тип к нормальной форме. А именно, введём операцию «*» следующим образом:

Определение 3.

$$\begin{aligned} \alpha^* &= \alpha \quad \text{если } \alpha \text{ — типовая переменная} \\ (\sigma \wedge \tau)^* &= \sigma^* \wedge \tau^* \\ (\sigma \to \tau)^* &= \bigwedge_{\varphi \in \hat{\tau^*}} \sigma \to \varphi \end{aligned}$$

Например, $(\alpha \to \gamma \wedge (\beta \to \gamma \wedge \delta))^* = (\alpha \to \beta \to \gamma) \wedge (\alpha \to \beta \to \delta) \wedge (\alpha \to \gamma)$.

Операция «*» переводит тип в эквивалентный.

Лемма 2.4. *Для любого типа τ: $\tau \sim \tau^*$*

Определение 4. *Тип τ находится в нормальной форме, если $\tau^* = \tau$.*

Общий вид типа в нормальной форме после применения «*» описывается следующей леммой:

Лемма 2.5. *Для любого типа τ: $\tau^* = \bigwedge_i (\varphi_{i1} \to \cdots \to \varphi_{iki} \to \alpha_i)$, где α_i - типовая переменная.*

Следствие 2.5.1. *Типы в нормальной форме без пересечений на верхнем уровне — в точности типы вида* $\cdots \to \alpha$*, где* α *– некоторая типовая переменная.*

В [8] показано, что отношение ($\leqslant$) рекурсивно и существует алгоритм, позволяющий определить для двух типов α и β верно ли, что $\alpha \leqslant \beta$:

Алгоритм 1.

$$\alpha \leqslant \beta = (\alpha \texttt{ == } \beta)$$
$$\textit{если } \alpha \textit{ и } \beta \textit{ — типовые переменные}$$
$$\sigma \leqslant \tau = False$$
$$\textit{если один из типов — переменная, а другой тип стрелочный}$$
$$(\sigma_1 \to \tau_1) \leqslant (\sigma_2 \to \tau_2) = (\tau_1 \leqslant \tau_2) \texttt{ AND } (\sigma_2 \leqslant \sigma_1)$$
$$\sigma \leqslant \tau = \forall \tau_i \in \hat{\tau^*} \colon \exists \sigma_j \in \hat{\sigma^*} \colon \sigma_j \leqslant \tau_i$$

Введём ещё несколько обозначений.

Определение 5. *Обозначим множество всех подтипов* τ *через* $\underline{\tau}$*, а множество всех его надтипов через* $\bar{\tau}$*. То есть* $\underline{\tau} = \{\sigma \mid \sigma \leqslant \tau\}$, $\bar{\tau} = \{\sigma \mid \tau \leqslant \sigma\}$.

Далее будем считать, что надтипы и подтипы могут использоваться везде, где могут использоваться типы. При этом операции над типами «поднимаются» на уровень множеств. Так, например, выражение $\alpha \to \bar{\beta} \to \underline{\gamma}$ следует понимать как множество $\{\alpha \to \beta' \to \gamma' \mid \beta' \in \bar{\beta}, \gamma' \in \underline{\gamma}\}$.

Утверждение о типизации, в котором фигурируют надтипы или подтипы, будем считать выводимым, если оно выводимо *для некоторых* элементов соответствующих можеств надтипов и подтипов.

Запись о вхождении типизации в контекст $(x \colon T) \in \Gamma$ в случае, если в T фигурируют надтипы или подтипы, означает, что $(x \colon \tau) \in \Gamma$ *для некоторого* $\tau \in T$.

Запись вида $T_1 \leqslant T_2$, где в T_i могут фигурировать подтипы и надтипы, означает, что $\forall \tau_1 \in T_1, \tau_2 \in T_2 : \tau_1 \leqslant \tau_2$.

Лемма 2.6. *Пусть* $\sigma \to \tau$ *– тип в нормальной форме. Тогда* $\underline{\sigma \to \tau} = \varphi \wedge (\bar{\sigma} \to \underline{\tau})$*, для некоторого (возможно, пустого)* φ*.*

Доказательство. Включение $\varphi \wedge (\bar{\sigma} \to \underline{\tau}) \subseteq \underline{\sigma \to \tau}$ почти очевидно:

$$\dfrac{\dfrac{\sigma \leqslant \bar{\sigma} \quad \underline{\tau} \leqslant \tau}{\bar{\sigma} \to \underline{\tau}}\,(R2)}{\varphi \wedge (\bar{\sigma} \to \underline{\tau}) \leqslant \sigma \to \tau}\,(A4)$$

Для включения $\underline{\sigma \to \tau} \subseteq \varphi \wedge (\bar{\sigma} \to \underline{\tau})$ достаточно посмотреть на Алгоритм 1. Подтип $\sigma \to \tau$ — это либо тип из $(\bar{\sigma} \to \underline{\tau})$ (третий случай алгоритма), либо тип-пересечение, один из элементов которого — подтип $\sigma \to \tau$; остальные элементы можно «переместить» в φ перестановкой. $\square$

2.2. Существенность η правила

Насколько добавление η - правило меняет систему? Система $\lambda_\wedge$ существенно слабее $\lambda_{\wedge\eta}$, а именно, есть тип, необитаемый в системе без эта-правила и обитаемый в системе с ним.

Лемма 2.7. *Утверждение о типизации $x : \alpha \to \beta \wedge \gamma \vdash x : \alpha \to \beta$ верно в $\lambda_{\wedge\eta}$ но неверно в $\lambda_\wedge$.*

Доказательство. Докажем выводимость в системе $\lambda_{\wedge\leqslant}$:

$$\dfrac{\dfrac{}{x\colon \alpha \to \beta \wedge \gamma \vdash x\colon \alpha \to \beta \wedge \gamma} \quad \alpha \to \beta \wedge \gamma \leqslant \alpha \to \beta}{x\colon \alpha \to \beta \wedge \gamma \vdash x\colon \alpha \to \beta}$$

Чтобы доказать, что утверждение неверно в системе $\lambda_\wedge$, предположим обратное. Рассмотрим последнее правило вывода, которое могло быть применено, чтобы получить данное утверждение. $(I \to)$ и $(E \to)$ не могли быть применены, поскольку они типизируют абстракцию и аппликацию соответственно, а x — типовая переменная. Правило (Ax) требует наличия соответствующей типизации в контексте, правило $(I\wedge)$ приписывает терму тип-пересечение, а не стрелочный тип.

Таким образом, единственным возможным правилом могло быть $(E\wedge)$:

$$\dfrac{\begin{array}{c}\vdots\\ x\colon \alpha \to \beta \wedge \gamma \vdash x\colon (\alpha \to \beta) \wedge \sigma\end{array}}{x\colon \alpha \to \beta \wedge \gamma \vdash x\colon \alpha \to \beta}$$

Какие правила вывода могли привести к данному утверждению о типизации? Только $(I \to)$ и $(E \to)$. Однако легко видеть, что их «обратное» применение порождает утверждение вида $x\colon \alpha \to \beta \wedge \gamma \vdash x\colon (\alpha \to \beta) \wedge \sigma$ для некоторого (возможно, пустого) σ, но утверждение такого вида уже встречалось ранее, и могло быть получено лишь с помощью $(I \to)$ и $(E \to)$. Значит, дерева вывода не существует.

□

Лемма 2.8. *Тип $\delta \wedge (\alpha \to \beta \wedge \gamma) \to \delta \wedge (\alpha \to \beta)$ пуст в $\lambda_\wedge$, но содержит терм* `id` *в $\lambda_{\wedge\eta}$.*

Доказательство. Анализом возможных деревьев вывода, аналогичным рассуждениям в предыдущей лемме, получаем, что для существования типизации $\vdash M\colon \delta \wedge (\alpha \to \beta \wedge \gamma) \to \delta \wedge (\alpha \to \beta)$ необходима и достаточна типизация $x\colon \alpha \to \beta \wedge \gamma \vdash x\colon \alpha \to \beta$.

□

Замечание 2.8.1. *Тип $(\alpha \to \beta \wedge \gamma) \to \alpha \to \beta$ обитаем в обеих системах. В $\lambda_\wedge$ он содержит $\lambda ab.ab$, но не $\lambda a.a$.*

2.3. Выводы и результаты по главе

В данной главе была рассмотрена система типов $\lambda_{\wedge\eta}$ и эквивалентная ей $\lambda_{\wedge\leqslant}$, для которых в следующих разделах будет приведён и доказан населяющий алгоритм. Также было введено отношение подтипизации «$\leqslant$»; отношение эквивалентности «$\sim$»; операции $\bar{\tau}$ и $\underline{\tau}$, ставящие в соответствие типу множество его надтипов и подтипов соответственно; операция «$*$», приводящая тип к нормальной форме. Алгоритм приведения к нормальной форме является важной составной частью населяющего алгоритма, описанию которого посвящена следующая глава.

3. Населяющий алгоритм

В [12] был представлен населяющий алгоритм для системы $\lambda_\wedge$. Однако этот алгоритм описан недостаточно полно, что выяснилось при его реализации. Кроме того, алгоритм и доказательство его корректности содержат ошибку. Данная работа устраняет эти неточности и ошибки. Алгоритм для $\lambda_{\wedge\leqslant}$ является модификацией алгоритма из [12], поэтому сначала рассмотрим его (в исправленном варианте).

3.1. Населяющий алгоритм для $\lambda_\wedge$

Алгоритм 2. *В процессе алгоритма решается система из нескольких задач:* $[\Gamma_1 \vdash X : \tau_1, \ldots, \Gamma_n \vdash X : \tau_n]$. *Решением системы является терм* X, *удовлетворяющий всем утверждениям типизации одновременно. При этом поддерживается инвариант: все задачи системы имеют общий набор переменных в контексте (при этом одной и той же типовой переменной могут соответствовать разные типы в разных контекстах). Алгоритм заключается в применении следующих преобразований до тех пор, пока это возможно:*

- 1. *Один из типов* τ_i *– пересечение* ($\tau_i = \sigma \wedge \rho$). *Тогда* i*-я задача* $\Gamma_i \vdash X : \tau_i$ *заменяется двумя:* $\Gamma_i \vdash X : \sigma$ *и* $\Gamma_i \vdash X : \rho$. *Таким образом, размер системы увеличивается*

- 2. *Все типы* τ_i *стрелочные* ($\tau_i = \sigma_i \to \rho_i$, $i = 1 \ldots n$). *Тогда решение системы –* $X = \lambda x.X'$, *где* X' *– решение новой системы* $[(\Gamma_1, x : \sigma_1 \vdash X' : \rho_1), \ldots, (\Gamma_n, x : \sigma_n \vdash X' : \rho_n)]$. *То есть в этом случае во всех типах редуцируется первый аргумент.*

- $3_\wedge$. *Один из типов* τ_i *– это типовая переменная. Тогда* X *не может быть абстракцией и должен быть (возможно пустой) аппликацией некоторой головной переменной, взятой из контекста, к термам. В этом случае необходимо недетерминированно выбрать из контекста головную переменную* x *и число* $k \geqslant 0$ *таким образом, чтобы* $\Gamma_i \vdash \lambda z_1 \ldots z_k.xz_1 \ldots z_k : \rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$ *для всех* $i = 1 \ldots n$. *Тогда решение –* $X = xZ^1 \ldots Z^k$, *где* Z^i *– решение системы* $[(\Gamma_1 \vdash Z^i : \rho_1^i), \ldots, (\Gamma_n \vdash Z^i : \rho_n^i)]$. *Здесь* k *систем независимы и могут быть решены параллельно.*

Если ни одно из преобразований применить невозможно, то система не имеет решения. «Точка выхода» из алгоритма – тот случай в преобразовании 3, при котором $k = 0$, *при этом решением является переменная.*

В данном алгоритме не совсем ясным является преобразование $3_\wedge$: как именно выбираются ρ_i^j. Типизация $\Gamma_i \vdash \lambda z_1 \ldots z_k.xz_1 \ldots z_k : \rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$ не гарантирует того, что тип x в контексте Γ_i обязан быть $\rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$.

3.2. Населяющий алгоритм для $\lambda_{\wedge\eta}$

Алгоритм для системы с η-правилом во многом повторяет алгоритм для $\lambda_\wedge$. Принципиальное отличие заключается в преобразовании 3.

***Алгоритм* 3.** *Поддерживается система из нескольких задач:* $\Gamma_1 \vdash X : \tau_1, \ldots, \Gamma_n \vdash X : \tau_n$. *До тех пор, пока это возможно, выполняется одно из четырёх преобразований:*

- 0. *Один из типов* τ_i *не находится в нормальной форме. Тогда заменим* τ_i *на* τ_i^*.
- *Шаги* 1 *и* 2 *аналогичны Алгоритму 2.*
- $3_{\wedge\eta}$. *Один из типов* τ_i *– это типовая переменная. Тогда* X *не может быть абстракцией и должен быть (возможно, пустой) аппликацией некоторой головной переменной, взятой из контекста, к термам. В этом случае необходимо недетерминированно выбрать переменную* x *и число* $k \geqslant 0$ *таким образом, чтобы* $\Gamma_i \vdash x : \rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$ *для всех* $i = 1 \ldots n$. *Тогда решение –* $X = xZ^1 \ldots Z^k$, *где* Z^i *– решение системы* $(\Gamma_1 \vdash Z^i : \rho_1^i), \ldots, (\Gamma_n \vdash Z^i : \rho_n^i)$. *Здесь* k *систем независимы и могут быть решены параллельно.*

  Следствие 4.1.3 описывает, как именно устроен недетерминированный выбор x, k *и* ρ_i^j.

Аналогично алгоритму 2, если ни одно из преобразований применить невозможно, система не имеет решения.

3.3. Отличия алгоритмов

В [12] в качестве населяющего алгоритма для системы $\lambda_\wedge$ приведён алгоритм, почти совпадающий с Алгоритмом 3, описанным выше. Однако на следующем примере можно убедиться, что данный алгоритм не является полным в системе $\lambda_\wedge$.

Пусть задача – $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash X\colon \beta$. Алгоритм должен произвести преобразование $3_{\wedge\eta}$, но в контексте нет такого x, что $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash x\colon \cdots \to \beta$, поскольку, согласно Лемме 2.7, утверждение о типизации $p\colon \alpha \to \beta \wedge \gamma \vdash p\colon \alpha \to \beta$ неверно в $\lambda_\wedge$. Поэтому алгоритм завершится, не найдя решение $X = pq$. Но преобразование $3_\wedge$ применить можно: $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash \lambda z.pz\colon \alpha \to \beta$. Здесь λ позволяет применить $(I \to)$ и удалить $\to$.

3.4. Выводы и результаты по главе

В данной главе был рассмотрен населяющий Алгоритм 2 для системы $\lambda_\wedge$, описанный ранее в [12], а также приведён оригинальный населяющий Алгоритм 3 для

системы $\lambda_{\wedge\eta}$. Ключевым отличием алгоритмов является преобразование 3, некоторые подробности которого для Алгоритма 3 описывает следствие 4.1.3.

Довольно очевидным является тот факт, что результат, выдаваемый алгоритмами корректен: если на входе τ алгоритм выдаёт терм x, то $\emptyset \vdash x\colon \tau$. Однако кроме корректности нас интересует полнота (если тип обитаем, то алгоритм находит его обитателей) и завершаемость (если тип необитаем, алгоритм завершится за конечное число операций). Оказывается, завершаемость в таком виде доказать невозможно, поскольку в общем случае задача населённости для этой системы типов неразрешима [17]. Но, при условии некоторых ограничений на входные типы, алгоритм всё же гарантированно останавливается за конечное время. Описанию этих ограничений, а также доказательству корректности, полноты и завершаемости посвящена следующая глава.

4. Свойства алгоритма

В этом разделе будут доказаны свойства Алгоритма 3 для системы $\lambda_{\wedge\eta}$. А именно, его корректность (soundness), полнота (completeness), завершаемость (termination) на типах ранга $\leqslant 2$.

4.1. Корректность

Корректность алгоритма означает, что ответ, данный им, удовлетворяет входным условиям. То есть если некий терм был найден, то он действительно типизируется заданным типом.

Теорема 1 (Soundness)**.** *Если алгоритм находит терм M для входного типа τ, то* $\vdash M : \tau$

Доказательство. Усилим утверждение: докажем, что алгоритм корректно решает систему задач. Доказательство – индукция по количеству шагов алгоритма. В базе индукции используется аксиома (Ax). В переходах индукции в шагах 1, 2 и $3_{\wedge\eta}$ используются правила вывода $(I\wedge)$, $(I\to)$ и $(E\to)$ соответственно. □

4.2. Полнота

Полнота – гораздо более содержательное свойство. Оно означает, что если существует терм заданного типа, то алгоритм обязательно найдёт или его, или другой терм этого типа. В нашем случае это будет терм в так называемой длинной форме. Далее будет доказан ряд лемм, имеющих отношение к преобразованию $3_{\wedge\eta}$ и проясняющих его смысл.

Лемма 4.1**.** *Пусть x – переменная, M_i – термы, τ – тип в нормальной форме без пересечений на верхнем уровне (см. следствие 2.5.1). Тогда равносильно:*

$$\Gamma \vdash xM_1 \dots M_k : \tau$$

$$\Longleftrightarrow$$

$$\Gamma \vdash x : \alpha_1 \to \dots \to \alpha_k \to \tau \text{ и } \Gamma \vdash M_i : \alpha_i \text{ для } i = 1 \dots k$$

Доказательство. Докажем равносильность в обе стороны.

$\Leftarrow$) применяя правило $(E\to)$ k раз, получаем в точности необходимое утверждение о типизации.

$\Rightarrow$) Если $k = 0$, то утверждение леммы очевидно. Иначе докажем индукцией по размеру дерева вывода (анализируя дерево с конца, аналогично доказательству

Леммы 2.7). Рассуждения будет удобнее провести в системе $\lambda_{\wedge\leqslant}$ (4), поскольку в ней меньше правил вывода.

Какое правило вывода могло быть применено последним, чтобы получить утверждение $\Gamma \vdash xM_1 \dots M_k \colon \tau$? Это не могло быть (Ax), так как оно типизирует переменную; это не могло быть $(I \to)$, так как оно типизирует лямбда абстракцию; это не могло быть $(I\wedge)$, поскольку в τ нет пересечений на верхнем уровне. Таким образом, могли быть применены только два правила: $(\leqslant)$ или $(E \to)$.

Согласно Лемме 2.5.1, τ имеет вид $\rho_1 \to \dots \to \rho_m \to \beta$, где β — типовая переменная. По Лемме 2.6, подтип τ тогда обязан иметь вид $\varphi_1 \wedge (\overline{\rho_1} \to \varphi_2 \wedge (\overline{\rho_2} \to \varphi_3 \wedge \dots (\overline{\rho_k} \to \beta) \dots))$. К такой типизации можно или снова применить $(\leqslant)$, получив нечто такой же формы (поскольку $\underline{\tau} = \underline{\tau}$) или применить $(I\wedge)$, получив в одной из веток нечто той же формы. Так или иначе, в определённый момент будет применено $(E \to)$ к $\underline{\tau}$.

Таким образом, в общем случае дерево вывода имеет следующий вид:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash xM_1 \dots M_{k-1} \colon \alpha_k \to \underline{\tau}\end{array} \quad \begin{array}{c}\vdots\\ \Gamma \vdash M_k \colon \alpha_k\end{array}}{\Gamma \vdash xM_1 \dots M_k \colon \underline{\tau}}\,(E \to)}{\begin{array}{c}\vdots\,(\leqslant), (I\wedge)\\ \Gamma \vdash xM_1 \dots M_k \colon \tau\end{array}}$$

Поскольку $\alpha_k \to \underline{\tau} \leqslant \alpha_k \to \tau$, верно утверждение о типизации $\Gamma \vdash xM_1 \dots M_{k-1} \colon \alpha_k \to \tau$, которое подходит под условия леммы. Поэтому, применив те же рассуждения ещё $k-1$ раз, получим $\Gamma \vdash x \colon \alpha_1 \to \underline{\alpha_2 \to \dots \to \alpha_k \to \tau}$, а также $\Gamma \vdash M_i \colon \alpha_i$ для $i = 1 \dots k$.

Легко видеть, что $\alpha_1 \to \underline{\alpha_2 \to \dots \to \alpha_k \to \tau} \leqslant \alpha_1 \to \dots \to \alpha_k \to \tau$, поэтому мы можем применить $(\leqslant)$ и получить необходимую типизацию.

□

Замечание 4.1.1. *Условие о том, что τ – тип без пересечений на верхнем уровне, существенно. Так, например, для $\Gamma = \{x \colon (\alpha \to \beta)\wedge(\gamma \to \delta), M \colon (\alpha\wedge\gamma)\}$, $\Gamma \vdash xM \colon \beta\wedge\gamma$, но при этом $\Gamma \nvdash x \colon \dots \to \beta \wedge \gamma$.*

Замечание 4.1.2. *Условие о том, что τ – тип в нормальной форме, существенно. Так, например, для $\Gamma = \{x \colon (\alpha \to \varphi \to \beta) \wedge (\gamma \to \varphi \to \delta), M \colon (\alpha \wedge \gamma)\}$, $\Gamma \vdash xM \colon \varphi \to (\beta \wedge \gamma)$, но при этом $\Gamma \nvdash x \colon \dots \to \varphi \to (\beta \wedge \gamma)$.*

Следствие 4.1.1. *В условиях предыдущей леммы, при выполнении любой из равносильных частей, $\Gamma \ni x \colon \underline{\alpha_1 \to \dots \to \alpha_k \to \tau}$,*

Доказательство. Для доказательства достаточно провести ещё один шаг в рассуждениях леммы, за тем лишь исключением, что вместо $(E \to)$ должно быть применено (Ax), что и гарантирует наличие нужной типизации в контексте. □

Следствие 4.1.2. *В условиях предыдущей леммы, при выполнении любой из равносильных частей, в контексте* Γ *есть типизация* $x\colon \sigma$ *такая, что* $\hat{\sigma^*} \ni \overline{\alpha_1} \to \cdots \to \overline{\alpha_k} \to \underline{\tau}$.

В частности, если τ *— типовая переменная, то* $\hat{\sigma^*} \ni \overline{\alpha_1} \to \cdots \to \overline{\alpha_k} \to \tau$.

Доказательство. Воспользуемся предыдущим следствием и тем наблюдением, что $\underline{\alpha_1 \to \cdots \to \alpha_k \to \tau} = \varphi_1 \wedge (\overline{\alpha_1} \to \varphi_2 \wedge (\overline{\alpha_2} \to \varphi_3 \wedge \ldots (\overline{\alpha_k} \to \underline{\tau}) \ldots))$ □

Следующее следствие проясняет, как с алгоритмической точки зрения устроено преобразование $3_{\wedge\eta}$ в Алгоритме 3. А именно, как выбрать x, k и ρ_i^j.

Следствие 4.1.3. *Пусть в шаге* $3_{\wedge\eta}$ *Алгоритма 3,* τ_i *– типовая переменная. Тогда для того, чтобы выбрать* x, k *и* ρ_i^j $(j = 1 \ldots k)$, *достаточно перебрать все элементы* $(x\colon \sigma)$ *контекста такие, что* $\hat{\sigma^*}$ *содержит тип, заканчивающийся на* τ_i, *то есть имеющий вид* $\rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$.

Для того, чтобы проверить, что выбранный x *удовлетворяет всем остальным задачам, а также чтобы подобрать* ρ_t^j *(для всех* $t \neq i$ *и* $j = 1 \ldots k$), *достаточно найти среди элементов* $\hat{\sigma^*}$ *(где* $\Gamma_t \ni (x\colon \sigma)$*) типы, у которых после «отщепления»* k *аргументов остаётся подтип* τ_t, *то есть имеющие вид* $\rho_t^1 \to \cdots \to \rho_t^k \to \tau_t'$, *где* $\tau_t' \leqslant \tau_t$.

Теорема 2 (Completeness). *Если у системы задач существует решение, то Алгоритм 3 найдёт его или другое решение.*

Доказательство. Доказательство – индукция по размеру наибольшего типа в задачах системы. Пусть дана система $T = [\Gamma_1 \vdash X\colon \tau_1, \ldots, \Gamma_n \vdash X\colon \tau_n]$, а M — её решение.

Если один из τ_i – тип-пересечение, алгоритм разобьёт его на два. При этом решение M останется решением системы, но уменьшится размер задач системы, поэтому можно применить индукционное предположение. Аналогичные рассуждения можно провести в случае, если один из τ_i не находится в нормальной форме (при этом, чтобы размеры типов уменьшились, нужно снять пересечения, сделав несколько раз преобразование 1).

Пусть ни один из типов τ_i не является пересечением. Рассмотрим, как может быть устроено M.

- $M = \lambda x.M'$.

  Тогда все типы в задачах должны быть стрелочные: $\tau_i = \alpha_i \to \beta_i$. Алгоритм перейдёт к системе $T = [\Gamma_1, x\colon \alpha_1 \vdash X'\colon \beta_1, \ldots, \Gamma_n, x\colon \alpha_n \vdash X'\colon \beta_n]$, у которой существует решение M'.

- $M = xM_1M_2 \dots M_k$.

  – Пусть все типы в задачах стрелочные: $\tau_i = \alpha_i \to \beta_i$. Тогда алгоритм, аналогично предыдущему пункту, перейдёт к системе $T = [\Gamma_1, y\colon \alpha_1 \vdash X'\colon \beta_1, \dots, \Gamma_n, y\colon \alpha_n \vdash X'\colon \beta_n]$. решение которой — $xM_1M_2 \dots M_k y$.

  – Пусть один из типов в задачах – переменная: $\tau_i = \beta$. По Лемме 4.1, верно следующее: $\Gamma_i \vdash x\colon \alpha_i^1 \to \dots \to \alpha_i^k \to \tau_i$ и $\Gamma_i \vdash M_j\colon \alpha_i^j$ для $i = 1 \dots n, j = 1 \dots k$. Поэтому, выбрав k и x, алгоритм перейдёт к системам $T_j = [(\Gamma_1 \vdash X_j : \overline{\alpha_1^j}), \dots, (\Gamma_n \vdash X_j : \overline{\alpha_n^j})]$. M_j является решением j-й системы.

□

4.3. Завершаемость

В предыдущих разделах было доказано, что Алгоритм 3 выдаёт корректное решение, если оно существует. Однако от алгоритма требуется, чтобы он завершался за конечное число шагов, даже если тип пустой. В текущей системе типов такие гарантии получить невозможно: задача обитаемости неразрешима [17]. Для обеспечения завершаемости можно ограничить множество входных типов, а именно ввести ограничение на их ранг.

Понятие ранга вводится в [13] и определяется как максимальная глубина вложенности «∧» в качестве левого аргумента «→». Более формальное определение следующее:

Определение 6**.**

$$
\begin{aligned}
rank(\tau) &= 0 \\
&\qquad \textit{если } \tau \textit{ – тип без пересечений} \\
rank(\sigma \wedge \tau) &= max(1, rank(\sigma), rank(\tau)) \\
rank(\sigma \to \tau) &= max(1 + rank(\sigma), rank(\tau)) \\
&\qquad \textit{если } rank(\sigma) > 0 \textit{ или } rank(\tau) > 0
\end{aligned}
$$

Так, например, $rank(\varphi \wedge (\tau \to (\alpha \wedge \beta) \to \gamma)) = 1$

Лемма 4.2**.** *Если* $rank(\rho_1 \to \dots \to \rho_k \to \tau) = r > 0$*, то* $rank(\rho_i) < r$ *для всех* $i = 1 \dots k$*.*

Следующее доказательство во многом повторяет доказательство завершаемости из [12].

Теорема 3 (Termination)**.** *Пусть* τ *– тип с рангом не больше двух* ($rank(\tau) \leqslant 2$)*. Тогда Алгоритм 3 завершается на входе* τ*.*

Доказательство. Пусть $rank(\tau) = 0$, что равносильно тому, что в τ нет пересечений. Тогда алгоритм завершится после линейного числа преобразований 2 (и затем одного $3_{\wedge\eta}$).

Проследим за рангами целевых типов τ_i и типов в контекстах Γ_i.

В преобразованиях 0 и 1 контексты не изменяются, а ранги типов в задачах не увеличиваются.

В преобразовании 2 ранг типов в задачах не увеличивается. А в контексты попадают типы σ_i, находящиеся слева от стрелки в типе $\tau_i = \sigma_i \to \rho_i$, а значит, имеющие или нулевой ранг, или хотя бы на 1 меньший ранг, чем τ_i (см. Лемму 4.2).

В преобразовании $3_{\wedge\eta}$ контексты не изменяются, а в целевыми становятся типы α_i^j, являющиеся аргументами в типах из контекстов, а значит, имеющие на 1 меньший ранг (или нулевой).

Из вышеизложенных соображений следует, что типы в контекстах всегда имеют ранг $\leqslant 1$, поэтому преобразование $3_{\wedge\eta}$ порождает задачи, в которых целевые типы имеют нулевой ранг. Эти задачи решаются за линейное число шагов. Преобразования $0 - 2$ структурно уменьшают целевые типы, поэтому применяются лишь конечное число раз.

□

4.4. Выводы и результаты по главе

В данной главе для Алгоритма 3 были показаны корректность (Теорема 1) и полнота (Теорема 2). Кроме того, была доказана важная Лемма 4.1, использующаяся в доказательстве полноты, и Следствия (4.1.1, 4.1.2 и 4.1.3) из неё. Эти следствия позволяют реализовать преобразование $3_{\wedge\eta}$, явным образом перебирая головную переменную в контексте. Также было введено понятие ранга типа и доказана завершаемость алгоритма для типов, имеющих ранг $\leqslant 2$.

Таким образом, в главе доказано, что задача обитаемости является разрешимой в системе $\lambda_{\wedge\eta}$ для типов ранга $\leqslant 2$.

Список литературы

[1] Alessi Fabio, Barbanera Franco, Dezani-Ciancaglini Mariangiola. Intersection Types and Lambda Models. — Vol. 355, no. 2. — P. 108–126. — Access mode: https://doi.org/10.1016/j.tcs.2006.01.004.

[2] Barendregt Henk, Dekkers Wil, Statman Richard. Lambda Calculus with Types. — New York, NY, USA : Cambridge University Press. — ISBN: 0-521-76614-1 978-0-521-76614-2.

[3] Church Alonzo. A Formulation of the Simple Theory of Types. — Vol. 5, no. 2. — P. 56–68. — Access mode: https://doi.org/10.2307/2266170.

[4] Coppo Mario, Dezani-Ciancaglini Mariangiola. An Extension of the Basic Functionality Theory for the \(\lambda\)-Calculus. — Vol. 21, no. 4. — P. 685–693. — Access mode: https://doi.org/10.1305/ndjfl/1093883253.

[5] Coppo Mario, Dezani-Ciancaglini Mariangiola, Venneri Betti. Functional Characters of Solvable Terms. — Vol. 27, no. 2-6. — P. 45–58. — Access mode: https://doi.org/10.1002/malq.19810270205.

[6] Curry Haskeil B. Modified Basic Functionality in Combinatory Logic. — Vol. 23, no. 2. — P. 83–92. — jstor : 42968456.

[7] Gabbay Dov. Semantical Investigations in Heyting's Intuitionistic Logic. — Dordrecht, Holland Boston Hingham, MA : D. Reidel Pub. Co. Distributed in the U.S.A. and Canada by Kluwer Boston. — ISBN: 978-94-017-2977-2.

[8] Hindley J. R. The Simple Semantics for Coppo-Dezani-Sallé Types // International Symposium on Programming / Ed. by Mariangiola Dezani-Ciancaglini, Ugo Montanari. — Berlin, Heidelberg : Springer Berlin Heidelberg. — P. 212–226.

[9] Hindley J. Roger. Types with Intersection: An Introduction. — Vol. 4, no. 5. — P. 470–486. — Access mode: https://doi.org/10.1007/BF01211394.

[10] Kfoury A. J., Wells J. B. Principality and Type Inference for Intersection Types Using Expansion Variables. — Vol. 311, no. 1-3. — P. 1–70. — Access mode: https://doi.org/10.1016/j.tcs.2003.10.032.

[11] Kurata Toshihiko, Takahashi Masako. Decidable Properties of Intersection Type Systems // Typed Lambda Calculi and Applications / Ed. by Mariangiola Dezani-Ciancaglini, Gordon Plotkin. — Berlin, Heidelberg : Springer Berlin Heidelberg. — P. 297–311.

[12] Kuśmierek Dariusz. The Inhabitation Problem for Rank Two Intersection Types // Typed Lambda Calculi and Applications / Ed. by Simona Ronchi Della Rocca. — Berlin, Heidelberg : Springer Berlin Heidelberg. — P. 240–254.

[13] Leivant Daniel. Polymorphic Type Inference // Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. — POPL '83. — New York, NY, USA : ACM. — P. 88–98. — Access mode: http://doi.acm.org/10.1145/567067.567077.

[14] Neergaard Peter Møller, Mairson Harry G. Types, Potency, and Idempotency: Why Nonlinearity and Amnesia Make a Type System Work // Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004. — P. 138–149. — Access mode: https://doi.org/10.1145/1016850.1016871.

[15] Pottinger Garrel. A Type Assignment for the Strongly Normalizable \(\lambda\)-Terms // J.R. Hindley; J.P. Seldin (Eds.), To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism. — London : Academic Press. — P. 561–577.

[16] Statman Richard. Intuitionistic Propositional Logic Is Polynomial-Space Complete. — Vol. 9. — P. 67–72. — Access mode: https://doi.org/10.1016/0304-3975(79)90006-9.

[17] Urzyczyn Pawel. The Emptiness Problem for Intersection Types. — Vol. 64, no. 3. — P. 1195–1215. — jstor : 2586625.

[18] Urzyczyn Pawel. Inhabitation in Typed Lambda-Calculi (A Syntactic Approach) // Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings / Ed. by Philippe de Groote. — Vol. 1210 of Lecture Notes in Computer Science. — Springer. — P. 373–389.

[19] Urzyczyn Pawe\l. Inhabitation of Low-Rank Intersection Types // Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications. — TLCA '09. — Berlin, Heidelberg : Springer-Verlag. — P. 356–370. — Access mode: http://dx.doi.org/10.1007/978-3-642-02273-9_26.

[20] Wells Joe B., Haack Christian. Branching Types // Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. — P. 115–132. — Access mode: https://doi.org/10.1007/3-540-45927-8_9.