УДК 160.1, 165.0, 510.20
DOI: 10.17223/1998863X/46/6

Л.Д. Ламберов

ПОНЯТИЕ ДОКАЗАТЕЛЬСТВА В КОНТЕКСТЕ ТЕОРЕТИКО-ТИПОВОГО ПОДХОДА, I: ДОКАЗАТЕЛЬСТВО ПРОГРАММ [1]

Рассматривается теоретико-типовое понятие доказательства корректности компьютерных программ. Исследуются особенности этого понятия, а также ряд связанных с ним теоретико-познавательных проблем. Особое внимание уделяется проблеме обозримости и связи априорного и апостериорного. Настоящая статья является первой в серии о понятии теоретико-типового доказательства.
Ключевые слова: *основания математики, доказательство, теория типов, компьютерные науки, корректность.*

При рассмотрении понятия доказательства в контексте теоретико-типового подхода в дальнейшем будут выделены три основных момента. Во-первых, это доказательство корректности программ, чему посвящена настоящая статья. Во-вторых, это доказательство математических результатов с помощью специальных компьютерных средств (например, *Coq* и *Agda*), что будет рассматриваться в следующей статье. В-третьих, это доказательство с точки зрения современного теоретико-типового подхода в рамках унивалентных оснований математики, что является предметом уже третьей статьи. Для начала следует отметить, что все три темы тесным образом связаны. Так, для доказательства корректности программ используются достаточно выразительные системы типов (например, с зависимыми типами), с помощью которых на формальном языке можно выразить спецификацию программы (её тип), а затем с помощью компилятора проверить, соответствует ли программа своей спецификации. Далее, благодаря соответствию Карри – Говарда, согласно которому, грубо говоря, тип соответствует математическому утверждению, а объект данного типа (например, функция) – доказательству этого утверждения (предъявлению или построению искомого объекта), математические результаты могут быть записаны и доказаны с помощью формального языка (достаточно выразительной) теории типов, а затем проверены компилятором точно так же, как проверяется корректность программ. Последнее разительным образом отличает применение вычислительных машин для получения математических результатов с помощью различных реализаций исчислений типов от доказательств методом перебора (как, например, в случае с доказательством теоремы о четырёх красках 1977 г.).

Собственно, мотивация понятия доказательства программ достаточно ясна. 5 октября 1960 г. из-за ошибки [1. P. 23–24]в работе радарной установки в Каанааке (Гренландия) BMEWS система раннего обнаружения баллистических ракет, проработавшая на тот момент всего четыре дня, сообщила о пя-

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-011-00301.

той, наивысшей степени угрозы. Этот уровень угрозы предполагал, что с территории СССР с вероятностью 99,9% по территории США произведён массовый запуск ракет, наиболее вероятно снаряжённых атомными боеприпасами. На принятие решения о дальнейших действиях у военного руководства было около 20 минут. Маршал авиации Рой Слемон, ответственный в то время за соответствующие решения, догадался уточнить у главы разведки NORAD, Командования воздушно-космической обороны Северной Америки, Харриса Халла текущее местонахождение Первого секретаря ЦК КПСС Никиты Хрущёва. Как оказалось, последний находился в Нью-Йорке, где всего через несколько дней сделал своё знаменитое выступление «с ботинком» по поводу венгерского вопроса. Из обстоятельств, сообщённых разведкой, Слемон заключил, что тревога (скорее всего) является ошибочной. «Это было типичное *человеческое* умозаключение: оно могло быть ошибочным, но оно *убедило* (курсив везде мой. – *Л. Л.*) Слемона» [1. P. 24].

В настоящее время вычислительные устройства окружают нас почти везде и встречаются практически во всякой человеческой деятельности, от многих из них зависит множество человеческих жизней. Все мы заинтересованы в том, чтобы в работе наших вычислительных машин ошибки возникали как можно реже, а в лучшем случае не возникали вообще. К этому можно приблизиться двумя способами, один из которых (тестирование) больше похож на методы естественных наук, а другой (формальное доказательство) – на методы математики. В силу сложности и многоуровневости вычислительных машин ни тот ни другой метод не даёт абсолютных гарантий, тем не менее если абстрагироваться от возможных проблем с оборудованием и операционной системой и сконцентрироваться исключительно на разработке программного обеспечения, то следует отметить следующие особенности указанных методов. Тестирование предполагает перебор некоторых типичных, а также некоторых крайних, предельных случаев, которые, естественно, не может покрыть все случаи использования ПО, что делает этот метод схожим с вероятностными методами естественных наук. Никто не может гарантировать, что тесты покрывают какой-нибудь случай, который произойдёт при реальном использовании программы, поэтому они изначально «неполны» и не могут гарантировать строгой определённости. С другой стороны, формальное доказательство с помощью вывода типов (в особенности построенное с использованием зависимых типов) даёт определённость при условии, что используемая при доказательстве формальная система корректна сама по себе и корректно реализована в виде языка программирования. Кроме того, благодаря соответствию Карри – Говарда вывод типов может использоваться и для доказательства конструктивных математических теорем (теорема представляет собой тип, а доказательство – объект данного типа). В дальнейшем внимание будет сосредоточено именно на формальных доказательствах.

В 1968 г. на конференции в Гармише был провозглашён «кризис программного обеспечения», для решения которого была предложена «инженерия разработки ПО», состоящая из двух компонент: практических дисциплин из области инженерии и теоретических оснований, выстраивающихся на основе математики и логики. По словам одного из основоположников компьютерных наук Э. Дейкстры, конференция в Гармише положила конец тёмному Средневековью в этой области исследований и открыла эпоху Просвещения.

Одной из отличительных черт этого процесса стало развитие и использование формализованных «математизированных» доказательств для демонстрации того, что та или иная программа соответствует своей спецификации, т.е. выполняет именно ту работу, для которой она задумывалась. К 1960-м гг. ещё не были построены строго определённые математические интерпретации инструкций большинства языков программирования. Языки программирования описывались по большей части исключительно синтаксически, описание же семантики было далеко не тривиальной задачей. Не будет преувеличением сказать, что развитие семантики языков программирования было связано в первую очередь с развитием идеи формальной верификации программ. Безусловно, некоторые черты семантического подхода можно обнаружить, допустим, в работах Дж. фон Неймана и А. Тьюринга. Тем не менее активная работа в этом направлении началась в 1960-х гг., в частности с указания на важность семантического описания Дж. Маккарти, который в статье «По направлению к математической науке вычислений» [2] призывал использовать математические методы для доказательства того, что те или иные процедуры позволяют решать те или иные проблемы, т.е., по словам Дж. Маккарти, можно было бы избавиться от процесса отладки и передать заботу о проверке доказательства самой вычислительной машине. В конечном счёте это привело к построению различного рода операционных и денотационных семантик языков программирования, возникновению логики Хоара, бурному развитию и применению теории типов, а также реализации различных исчислений в виде таких языков программирования и средств интерактивного доказательства теорем, как *Coq*, *Agda*, *Isabelle*, *Mizar* и мн. др. В настоящее время эти средства используются как для написания верифицированных программ (т.е. программ, содержащих «математическое» доказательство своего соответствия спецификации), так и для написания формальных доказательств математических теорем.

В действительности доказательства программ и доказательства теорем оказываются теснейшим образом связанными не только концептуально (о чём речь пойдёт ниже), но и исторически. Компьютерное доказательство теоремы о четырёх красках было опубликовано[1] К. Аппелем и В. Хакеном в 1977 г. За публикацией доказательства теоремы о четырёх красках последовала реакция со стороны как математического, так и философского сообщества. Однако высказанная критика, можно сказать, «витала в воздухе», поскольку в самом начале 1977 г. была опубликована совместная статья Р. Демилло, Р. Липтона и А. Перлиса [5], в которой особое внимание обращалось на социальные процессы легитимации математических результатов и, собственно, во многом предвосхищалась критика компьютерных доказательств математических теорем. Работая над формальным подходом к доказательству программ, Р. Демилло и Р. Липтон в одном из частных обсуждений[2] около 1974 г. задались вопросом, чем же является доказательство, ответ на который Р. Липтон сформулировал в духе натурализации философии математики: «Доказательства представляют собой не больше и не меньше, чем то, что делают математики».

[1] Подробное изложение даётся в серии из двух статьей, разделённых с учётом авторства: [3, 4]. Краткое изложение дано в [5].

[2] См.: [1. P. 201–202].

Статья Р. Демилло, Р. Липтона и А. Перлиса открывается замечательной цитатой знаменитого логика XX в. Дж. Россера (вообще каждый раздел статьи начинается с подобного эпиграфа или даже нескольких): «Мне хотелось бы задать тот же вопрос, что задал Декарт. Вы собираетесь дать точное определение логической корректности, которое должно совпасть с моим смутным интуитивным ощущением логической корректности. Но как вы намерены доказать, что они совпадают? Средний математик не должен забывать, что интуиция является последним авторитетом». Согласно авторам и целью верификации программ, и целью математической деятельности является увеличение степени уверенности – уверенности в том, что программа корректна, или в том, что теорема истинна. Для построения доказательства в математике может использоваться логический вывод, представляющий собой цепочку элементарных шагов, каждый из которых является либо аксиомой, либо получен из предыдущих с помощью обоснованных правил вывода. Тем не менее это далеко не всё, что составляет доказательство. Помимо самого доказательства, которое может быть либо выражено формальным способом, либо представлено в формальном виде лишь частично, важна роль сообщества. Следует обратить внимание на то, что одна из целей, для которых доказательство строится, – повышение степени уверенности. Другими словами, доказательство представляет собой аргумент, призванный убеждать. Кроме того, необходимо отметить, что ошибки математиков – это не что-то из ряда вон выходящее. Математики, как и все люди, совершают ошибки, но эти ошибки не остаются незамеченными, они обнаруживаются или самим авторами доказательств, или другими математиками.

Однако даже несмотря на то, что целью доказательства является убеждение, доказательство само по себе ещё никакого убеждения не даёт. Кроме того, возникает вопрос, во что же мы верим, когда верим в математические результаты. Мы верим в (возможно, принципиальное) существование чистого формального доказательства либо в то, что результат имеет место с некоторой вероятностью (как например, в случае с откровениями Рамануджана или вероятностным методом определения простоты М. Рабина). Доказательство представляет собой своего рода «элементарный шаг» процесса развития математики, а потому само понимание того, чем является математика, зависит от понимания этого «элементарного шага». В случае, если доказательство понимается первым из указанных способов, природа математики оказывается кардинально отличной от природы остальных наук. Если же доказательство даёт лишь вероятностную уверенность, то математика представляет собой нечто подобно естественным наукам.

Используя метафору Р. Демилло, Р. Липтона и А. Перлиса [6. P. 273], можно сказать, что доказательство математического результата представляет собой «сообщение», а не некоторый идеальный объект, удостоверившись в существовании которого исследователь может спокойно отдохнуть. И наоборот, с построения доказательства ещё только начитается его «жизнь». Доказательство математического результата в дальнейшем проходит сложный путь: оно изучается сообществом, рецензируется, публикуется, обсуждается, интернализируется и перефразируется (т.е. участники математического сообщества стремятся согласовать новое доказательство со своими собственными взглядами и вписать его в свою собственную «математическую картину

мира»), обобщается, используются и связывается с другими результатами, входит в доказательства других результатов, в том числе и в других областях математики. Даже известный математический результат открывается множество раз и порой разными способами. Указанные обстоятельства, представляющие собой скорее социальную жизнь математических результатов, значительно усиливают убеждённость доказательства для всего математического сообщества. Фактически такая обработка математического результата приводит к стабилизации ключевых понятий, после чего он в полной мере становится частью признанного математического знания, а убеждённость в нём может оказаться достаточной, чтобы верить в существование чистого формального доказательства, даже несмотря на то, что изначально результат мог восприниматься как вероятностный.

Описанный социальный процесс имеет и прагматические мотивы, а именно: обычное желание изучать чёткие и ясные проблемы, работать с простыми понятиями и проверять простые доказательства[1].

Рассмотрим некий идеализированный процесс по разработке программного обеспечения и соответствующего доказательства корректности. Разработка начинается с формулировки требований, предъявляемых к ПО, и описания работы итоговой программы. По своей природе эти требования и описание являются неформальными, поэтому для начала требуются определённые (и порой весьма серьёзные) усилия как для их формулировки, так и для формулировки формализованных требований и описания. После этого разрабатывается программное обеспечение, для которого затем строится доказательство его корректности, где под корректностью понимается, во-первых, соответствие формализованной спецификации, а во-вторых, отсутствие ошибок, связанных со смешением типов данных в программе. Было бы весьма странно требовать, чтобы указанное доказательство строилось непосредственно людьми, поскольку малейшее изменение исходного кода программы требует переработать по меньшей мере соответствующие фрагменты доказательства. Такие доказательства строятся разработчиками ПО лишь частично через определение типов данных подходящим для проверки образом, подробное описание спецификации каждой отдельно взятой функции, а также через определение специальных вспомогательных функций. В дальнейшем построение доказательства для отдельно взятой программы выполняется компилятором. Такой подход к разработке ПО оказывается весьма затратным по времени и в настоящее время распространён очень мало. В первую очередь это связано с тем, что полноценные доказательства даже для относительно простых программ занимают довольно много строк кода[2].

Следует обратить внимание на то, что само по себе «доказательство» программы не имеет социального измерения. Так, доказательство в полном смысле (вместе с непосредственной проверкой компилятором) не интересует никого, за исключением, пожалуй, узких специалистов (и то в редких случаях отладки собственно компилятора). Тем не менее понятие доказательства программ сталкивается с двумя более фундаментальными затруднениями. Во-

[1] О роли простоты доказательства см.: [7].

[2] Например, можно сравнить реализацию алгоритма быстрой сортировки, которая на языке Haskell занимает три строки кода, а на языке Agda вместе с доказательством соответствия кода спецификации занимает уже 240 строк. Реализацию и доказательство на Agda см.: [8].

первых, доказательство является формальным, но в нём доказывается соответствие неформальным требованиями. В этом случае необходимо объяснить, каким образом неформальное описание желательной работы программы может быть сделано формальным. Эта ситуация подобна ситуации с тезисом Чёрча [9, 10]. Во-вторых, доказательство неотделимо от собственно программы, поскольку программы имеют свойство меняться, переписываться, в них исправляются ошибки и добавляются новые возможности. Таким образом, практически при каждом изменении меняется и доказательство. В настоящее время это уже не является такой уж серьёзной проблемой и имеет чисто техническое решение. Однако спецификация какой-либо нетривиальной программы с учётом всех входных и выходных данных может занимать огромный объём, что требует наличия развитых инструментов программирования, отсутствующих в настоящий момент. Таким образом, спецификация и собственно программы отделены друг от друга и для того, чтобы доказательство было адекватным, не должны смешиваться, что обеспечить в ходе разработки весьма затруднительно[1]. Следует указать, что работа над спецификацией и над доказательством не может быть отдана на откуп автоматам, поскольку в этом случае в цепочке звеньев, каждое из которых требует веры в свою надёжность, появится ещё одно звено.

Допустим, исследователям удалось построить систему для автоматического доказательства программ. Что в таком случае будет знать программист, когда получит автоматическое доказательство?[2] В этом случае программист будет находиться в ситуации, которая мало чем отличается от ситуации с предсказаниями Рамануджана, которому результаты диктовала богиня Наматжири, или математического авторитета марсианина Саймона [11]. В конечном счёте прагматическая цель разработки программ состоит в том, чтобы они были в первую очередь надёжными, а лишь во вторую (или третью, четвёртую) – проверяемыми.

Действительно, доказательство выполняет определённую социальную роль, а потому обладает социальным характером. При этом само по себе доказательство является скорее цепочкой логических рассуждений. Вычислительная машина вполне справляется с построением формального доказательства как такой цепочки рассуждений и может построить то, что можно назвать «полным доказательством» в отличие от его наброска [12]. Согласно Дж. Фетцеру полное доказательство относится в области априорного знания, а потому является абсолютно достоверным. Однако следует обратить особое внимание на то, что собой представляет компьютерная программа. Исполнение *любой* компьютерной программы приводит к изменению физического состояния[3] конкретной вычислительной машины. Таким образом, программа представляет собой не просто реализацию абстрактного алгоритма для абстрактной машины, а каузальную сущность. Обладаем ли мы и можем ли в принципе обладать априорным знанием о каузальных сущностях? Другими словами, могут ли утверждения о физических объектах и причинно-

[1] К примеру, в ходе разработки программа «подгоняется» под спецификацию либо спецификация меняется в случае обнаружения каких-либо ранее неучтённых технических аспектов.

[2] Необходимо ещё раз отметить, что размер доказательства значительно превышает размер программы.

[3] Это так, даже если исключить программы, которые управляют, допустим, сервоприводами или другими устройствами.

следственных связях между ними быть настолько же достоверными и обладать таким же необходимым характером, как утверждения чистой математики? Представляется, что нет, а потому и сама идея доказательства программ, по мнению Дж. Фетцера, является категориальной ошибкой.

Однако, как кажется, ситуация с понятием доказательства программ не так однозначна. Дело в особых и довольно многомерных понятиях [13] абстракции и сокрытия информации в компьютерных науках. Коротко говоря, сокрытие информации предполагает отделение реализации, принадлежащей более низкому уровню, от интерфейса, доступного на определённом уровне для работы. Абстрагирование же предполагает обобщение и отбрасывание некоторой информации или особенностей с тем, чтобы иметь возможность «реализовывать» общие абстрактные структуры в разных конкретных сущностях. Такое понимание сокрытия информации и абстракции находится под серьёзным влиянием идей классической логики и не является единственным или таким уж однозначным в компьютерных науках. Как бы то ни было, эти понятия важно учитывать при рассмотрении доказательства программ, что поможет прояснить саму идею такого доказательства. Так, странно было бы предполагать, что доказательство программы, открывающей, допустим, DVD-привод, является формальным доказательством того, что при выполнении этой программы DVD-привод вычислительной машины *всегда* будет открываться. Очевидно, это не так. Дело в том, что доказательство такой программы представляет собой формальное доказательство корректной реализации некоторого алгоритма, доказательство соответствия программы её формально определённой спецификации. В этом случае имеется в виду, что программа будет выполнять определённые вычисления, требующиеся для взаимодействия с DVD-приводом через соответствующий интерфейс (например, через соответствующие функции операционной системы или драйверов устройства). Это доказательство, безусловно, не является математическим доказательством того, что некоторое данное физическое устройство перейдёт в строго определённое физическое состояние. Оно является формальным доказательством только на определённом уровне абстракции, при условии сокрытия и без учёта подлежащей реализации; в более же широком контексте оно уже не будет обладать искомыми характеристиками чистого формального доказательства. Таким образом, критика Дж. Фетцера представляется необоснованной, хотя следует признать, что она поднимает весьма интересную проблему.

Литература

1. *MacKenzie D.* Mechanizing Proof: Computing, Risk, and Trust. Cambridge, Mass. : The MIT Press, 2004. xi+428 p.

2. *McCarthy J.* Towards a Mathematical Science of Computation // Information Processing 1962: Proceedings of IFIP Congress 62 / ed. by M. Popplewell. Amsterdam : North Holland, 1963. P. 21–28.

3. *Appel K., Haken W.* Every Planar Map is Four Colorable. I. Discharging // Illinois Journal of Mathematics. 1977. Vol. 21, № 3. P. 429–490.

4. *Appel K., Haken W., Koch J.* Every Planar Map is Four Colorable. II. Reducibility // Illinois Journal of Mathematics. 1977. Vol. 21, № 3. P. 491–567.

5. *Appel K., Haken W.* The Solution of the Four-Color-Map Problem // Scientific American. 1977. Vol. 237, № 4. P. 108–121.

6. *DeMillo R., Lipton R., Perlis A.* Social Processes and Proofs of Theorems and Programs // Proceedings of the Fourth ACM Symposium on Principles of Programming Languages. 1977. P. 206–214.

7. *Черепанов Е.М.* Простота как критерий убедительности доказательства // Философия науки. 2010. № 1. С. 91–101.

8. *Верифицированный* QuickSort на Agda // Хабр. 2012. URL: https://habr.com/post/148769/ (дата обращения: 20.11.18).

9. *Целищев В.В.* Тезис Чёрча. Новосибирск: Параллель, 2008. 173 с.

10. *Бессонов А.В., Хлебалин А.В., Целищев В.В.* Можно ли доказать тезис Чёрча? // Философия науки. 2008. № 2. С. 44–61.

11. *Timoczko T.* The Four-Colour Problem and Its Philosophical Significance // Journal of Philosophy. 1979. Vol. 76, № 2. P. 57–83.

12. *Fetzer J.* Program Verification: The Very Idea // Communications of the ACM. 1988. Vol. 31, № 9. P. 1048–1063.

13. *Osternamm K., Giarrusso P.G., Kästner C., Rendel T.* Revisiting Information Hiding: Reflections on Classical and Nonclassical Modularity // ECOOP 2011 – Object-Oriented Programming. ECOOP 2011. Lecture Notes in Computer Science / ed. by M. Mezini. Berlin, Heidelberg : Springer, 2011. P. 155–178.

Lev D. Lamberov, Ural Federal University named after the first President of Russia B.N. Yeltsin (Yekaterinburg, Russian Federation).

E-mail: lev.lamberov@urfu.ru

Vestnik Tomskogo gosudarstvennogo universiteta. Filosofiya. Sotsiologiya. Politologiya – Tomsk State University Journal of Philosophy, Sociology and Political Science. 2018. 46. pp. 49–57.

DOI: 10.17223/1998863X/46/6

THE CONCEPT OF PROOF IN THE CONTEXT OF A TYPE-THEORETIC APPROACH, I: PROOF OF COMPUTER PROGRAM CORRECTNESS

Keywords: foundations of mathematics; proof; type theory; computer science; correctness.

When considering the concept of proof in the context of a type-theoretic approach, one should refer to the concepts of (1) proof of a computer program correctness, (2) proof of mathematical results by means of special computer tools, (3) proof from the point of view of the modern type-theoretical approach within the framework of univalent foundations of mathematics. This paper is devoted to the concept of proof of computer program correctness. Currently, computing devices surround us almost everywhere and are found in virtually every human activity. We are all interested in the correct functioning of our computers, so errors occur as rarely as possible, and at best do not occur at all. A formal proof using type inference gives certainty, provided that the formal system used in the proof is consistent and correctly implemented as a programming language. One of the purposes for which proof is formulated is to increase the degree of confidence. However, the proof itself does not give any conviction. If our belief in a proof is belief in the existence of pure formal proof, then the nature of mathematics is radically different from the nature of other sciences. If our belief implies a certain proportion of probability, then mathematics turns out to be similar to the natural sciences. The proof of the mathematical result is a “message”, and not some ideal object. The social aspect of mathematical results greatly enhances conviction. In fact, it leads to the stabilization of key concepts, after which the result becomes part of recognized mathematical knowledge, and conviction in it may be sufficient to belief in the existence of purely formal proof. Proof of the computer program correctness does not have this characteristic. Further, it is assumed that the execution of the program leads to a change in the physical state of a particular computer. Since we cannot have a priori proofs for the a posteriori phenomenon, a number of researchers suggest that the concept of proof of correctness is a categorical error. Nevertheless, it is necessary to take into account the peculiarities of the concepts of abstraction and information hiding in computer science. With a careful approach, the concept of proof of correctness is quite reasonable and does not contain a categorical error.

References

1. MacKenzie, D. (2004) *Mechanizing Proof: Computing, Risk, and Trust*. Cambridge, Mass.: MIT Press.

2. McCarthy, J. (1963) Towards a Mathematical Science of Computation. In: Popplewell, M. (ed.) *Information Processing 1962: Proceedings of IFIP Congress 62*. Amsterdam: North Holland. pp. 21–28.

3. Appel, K. & Haken, W. (1977) Every Planar Map is Four Colorable. I. Discharging. *Illinois Journal of Mathematics*. 21(3). pp. 429–490. DOI: 10.4236/jmf.2011.13014 4,016

4. Appel, K., Haken, W. & Koch, J. (1977) Every Planar Map is Four Colorable. II. Reducibility. *Illinois Journal of Mathematics*. 21(3). pp. 491–567. DOI: 10.1090/conm/098

5. Appel, K. & Haken, W. (1977) The Solution of the Four-Color-Map Problem. *Scientific American*. 237(4). pp. 108–121.

6. DeMillo, R., Lipton, R. & Perlis, A. (1977) Social Processes and Proofs of Theorems and Programs. *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*. pp. 206–214.

7. Cherepanov, E.M. (2010) Prostota kak kriteriy ubeditel'nosti dokazatel'stva [Simplicity as a criterion for convincing evidence]. *Filosofiya nauki – Philosophy of Sciences*. 1. pp. 91–101.

8. Verified QuickSort on Agda. (2012) [Online] Available from: https://habr.com/post/148769/. (Accessed: 20th November 2018).

9. Tselishchev, V.V. (2008) *Tezis Chercha* [Thesis of Church]. Novosibirsk: Parallel'.

10. Bessonov, A.V., Khlebalin, A.V. & Tselishchev, V.V. (2008) Mozhno li dokazat' tezis Chercha? [Is it possible to prove the thesis of Church?]. *Filosofiya nauki – Philosophy of Sciences*. 2. pp. 44–61.

11. Timoczko, T. (1979) The Four-Colour Problem and Its Philosophical Significance. *Journal of Philosophy*. 76(2). pp. 57–83. DOI: 10.2307/2025976

12. Fetzer, J. (1988) Program Verification: The Very Idea. *Communications of the ACM*. 31(9). pp. 1048–1063. DOI: 10.1145/48529.48530

13. Osternamm, K., Giarrusso, P.G., Kästner, C. & Rendel, T. (2011) Revisiting Information Hiding: Reflections on Classical and Nonclassical Modularity. In: Mezini, M. (ed.) *ECOOP 2011 – Object-Oriented Programming. ECOOP 2011. Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer. pp. 155–178.