СЕМАНТИЧЕСКОЕ МОДЕЛИРОВАНИЕ И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

академик РАН С.С.Гончаров, д.ф.-м.н. Д.И.Свириденко

(Институт математики СО РАН)

Ключевые слова: итеративное программирование, декларативное программирование, функциональное программирование, логическое программирование, вычислимость, аксиоматический подход, теоретико-модельный подход, семантическое моделирование, искусственный интеллект, «объяснительный» искусственный интеллект

Аннотация: Обсуждается проблема решения задач с помощью компьютеров. Анализируются достоинства и недостатки декларативного программирования, в том числе, функционального и логического программирования, выступающих как инструментарий искусственного интеллекта. Описывается альтернативный, теоретико-модельный подход к решению задач, носящий название семантического моделирования. Анализируются достоинства предлагаемой концепции, в том числе, возможность сочетания аксиоматического и теоретико-модельного подхода к решению задач в рамках единой подхода, а также возможность интеграции на базе семантического моделирования методов непрерывной и дискретной математики. Описывается возможность построения на основе семантического моделирования нового, «объяснительного искусственного интеллекта, свободного от недостатков, присущих традиционному искусственному интеллекта

Человек в живой природе занимает особое место и одно из основных качеств, выделяющих его в ней – это умение ставить задачи, решать их и, что очень важно, при необходимости не только ***объяснить*** найденное решение, но и ***обосновать*** его. Длительное время человек достаточно успешно пользовался этим умением, не прибегая к помощи каких-либо внешних инструментов, пока не столкнулся с необходимостью научиться решать сложные и масштабные задачи. Это вынудило его искать соответствующий инструментарий, и он придумал компьютер – пожалуй, один из чрезвычайно перспективных и весьма эффективных инструментов.

При решении человеком задач с помощью компьютера стандартная последовательность действий выглядит следующим образом: формулируется задача, ищется алгоритм ее решения, формальное или формализованное описание алгоритма передается программисту для написания кода на том или ином языке программирования, понятного используемому компьютеру, пишется и отлаживается программа, планируются и осуществляются вычисления, порождаемые написанной программой, полученные данные интерпретируются Заказчиком и если они удовлетворительны с точки зрения некоторого критерия, то принимается решение, что задача решена. Если же результат неудовлетворителен, то производится анализ предпринятых действий и в них вносятся соответствующие изменения. Заметим, что такие изменения могут коснуться практически всех этапов процесса решения задачи, вплоть до этапа ее постановки. При этом, ошибки, нечеткости или неточности, допущенные при формулировке задачи, могут повлечь за собой весьма «дорогостоящие» последствия, что налагает на постановщика задачи особую ответственность.

В связи с таким «экономическим» подходом к проблеме компьютерного решения задач имеет смысл вспомнить известный афоризм: «правильно поставленная задача – наполовину решенная задача». Сразу же возникает вопрос – а существует ли ситуация, когда правильно поставленная задача фактически означает ее полноценное решение? Поскольку речь идет о решении человеком задач с помощью компьютеров, то этот вопрос можно переформулировать иначе – существует ли возможность формулировать задачу

таким образом, чтобы из этой формулировки компьютер мог бы автоматически «извлечь» требуемое решение? И не только извлечь решение, но и ***объяснить*** его?

Заметим, что этот вопрос не оригинален – решением похожей проблемы занимается ***искусственный интеллект (ИИ)*** [1-3]**.** В частности, такое направление ИИ как машинное обучение (смотри, например, [4]) решает эту проблему, пытаясь с помощью математико-статистических методов и технологии нейронных сетей путем их обучения (методом «натаскивания») синтезировать решение той или иной массовой задачи. Например, по радужной оболочка человеческого глаза или по тепловому полю поставить пациенту диагноз, распознать человеческое лицо в толпе идущих людей и т.д.

Следует отметить, что технология нейронных сетей имеет много удачных и вполне успешных приложений и активно развивается, особенно в последнее время. Но она же обладает и серьезным изъяном – синтезируемые нейронными сетями решения являются плохо или даже вообще необоснованными, поскольку никто не знает, как нейронная сеть приходит к своим решениям (так называемый эффект "черного ящика"). Кроме этого, системы искусственного интеллекта, базирующиеся на применении технологии традиционных нейронных сетей, являются нестабильными, демонстрируя непредсказуемое поведение в непредусмотренных и нестандартных условиях. Оба этих обстоятельства – необоснованность и нестабильность, делают использование традиционных нейронных сетей проблематичным и даже зачастую вообще неприемлемым в таких областях, как медицина, оборона, логистика, управление сложными человеко-машинными системами и т.п. В чем же причина таких негативных эффектов?

На взгляд авторов этой заметки главной причиной наличия указанных выше недостатоков применяемых технологий ИИ является изолированность и обособленность используемых ими приемов и инструментов от исходного *смысла (семантики)* задачи, т.е. от тех знаний о проблемной области, которые используются при ее постановке и формулировке. Такая изолированность и обособленность усугубляется еще и тем, что при написании компьютерных программ используются, как правило, технологии и языки программирования, носящие *императивный* характер, что автоматически влечет за собой разделение данных и операций над данными [5]. Напомним, что при описании задач и при их решении в виде алгоритмов мы всегда имеем дело с двумя видами сущностей:

- *данными*, т.е. с тем, что мы хотим обрабатывать, преобразовывать, управлять,
- *операциями*, т.е. с действиями над данными.

Очевидно, что любой язык формулирования задач и любой язык, в терминах которого мы записываем ее решение, должен предоставлять возможность описывать эти сущности. Для этого такие языки обычно содержат, с одной стороны, элементарные выражения, представляющие исходные примитивные данные и операции над ними (базис языка), а, с другой стороны, набор специальных средств конструирования из этих простых объектов более сложных конструкций, которые могли бы рассматриваться как единые сущности, что позволяло бы манипулировать этими конструкциями с помощью средств этого же языка. И то, что при императивном программировании мы вынуждены ***разделять*** данные и операции является одной из главных причин разрушения исходной семантики задачи. Помимо всего прочего разрушение и, тем самым, неполный учет семантических аспектов задачи приводит к таким проблемам, как неструктурированность больших данных (Big Data), с которой, что интересно, предлагается бороться, используя … те же традиционные технологии ИИ, пытаясь придать с их помощью некий смысл получаемым решениям и, тем самым, как бы восстановить семантику решаемых задач. Возникает порочный круг – вначале разрушаем исходную семантику задачи с тем, чтобы потом попытаться ее восстановить!

Помимо императивных языков программирования ИИ активно пользуется так называемыми *декларативными языками* [6], каковыми являются функциональные [7-9] и

логические [10-13] языки программирования. Но и это не спасает ситуацию. Остановимся чуть подробнее на этом моменте.

На этапе формулирования человеком задачи, решение которой предусматривает использование компьютера и потому формулировка задачи должна быть свободной от неточностей и двусмысленностей, стараются применять по возможности строгие, формальные конструкции. Человеческий опыт подсказывает, что для человека, формулирующего задачу, предпочтительнее, чтобы используемые языковые конструкции имели вид математических утверждений, носящих повествовательный, декларативный характер. Правда иногда, когда задача изначально формулируется как необходимость осуществить совокупность некоторых действий, используется форма представления задачи и в виде последовательности инструкций, носящих уже повелительный, т.е императивный характер. Очевидно, что императивными по своей природе являются и описания решений задач в виде алгоритмов. И уж совсем отчетливый императивный характер имеют представляющие эти алгоритмы команды компьютерной программы.

Справедливости ради стоит отметить, что императивные языки программирования высокого уровня, как правило, имеют достаточно развитые средства написания повествовательных, декларативных конструкций. Используя такие средства можно в отдельных случаях, когда уже заранее имеется некая компьютерная программа решения целого класса задач (шаблон), формулировать конкретную задачу из данного класса задач не как последовательность инструкций, а как совокупность декларативных условий, позволяющих параметрически настроить этот шаблон на решение именно этой конкретной задачи и автоматически получать это решение.

Понятно, что такой автоматизированный способ решения задачи выглядит намного привлекательнее, поскольку вместо указаний о том, *как* решать задачу мы формулируем лишь описание того, *что* следует решать. Поэтому вполне закономерным является вопрос: а можно ли создать декларативный язык постановки задач, который бы:

- опирался на общепризнанную и развитую математическую базу,
- максимально был приближен к постановщику задачи, был удобен и комфортабелен для формулировок задач и способен был адекватно отражать в постановке исходную семантику используемых знаний,
- был пригоден для формулировки по возможности максимально широкого класса практических задач,
- допускал бы эффективную интерпретацию своих конструкций как набор команд для компьютера?

Заметим, что в предельном случае этот вопрос звучит как вопрос о возможности создания некого универсального декларативного языка с добротной математической базой и пригодного для декларативного описания любой(!) задачи, имеющей алгоритмическое решение.

Поскольку в формулировке всех этих вопросов речь идет о задачах, допускающих решение с помощью компьютера в виде набора неких вычислительных конструкций, то поиск ответов на поставленные вопросы естественно начать с поиска математической основы такого языка, т.е. с поиска и анализа существующих в дискретной математике моделей, формализующих понятие ***«вычислимость».***

Напомним, что наиболее распространенными в математике являются *язык множеств* и *язык функций*. Поэтому вполне естественно, что изучение понятия вычислимости в основном осуществлялось и продолжает осуществляться в терминах именно этих языков - «вычислимая функция», «вычислимое множество». Однако не все созданные математиками модели вычислимости для данных понятий оказались приемлемыми и перспективными при компьютерном решении задач. Так, например, в парадигме

функционального программирования (ФП) используются в основном только три модели, формализующих понятие «вычислимость» в терминах вычислимых функций:

- классическая теория рекурсивных функций;
- ламбда-исчисление (типовое и бестиповое);
- комбинаторная логика.

Так, например, *модель рекурсивных функций* составляет основу языка *РЕФАЛ* (РЕкурсивных Функций АЛгоритмический язык), придуманного еще в Советстком Союзе в начале 60-х годов прошлого столетия. Другая функциональная модель вычислимости - *лямбда-исчисление* или *исчисление лямбда-конверсий*, наиболее популярная в ФП модель, была создана в 1934 г. А.Черчем. Предложенный им *ламбда-язык*, несмотря на его кажущуюся простоту, оказался весьма мощным по своей выразительности, поскольку, как выяснилось, позволяет (согласно знаменитому *Тезису Черча*) описывать все(!) интуитивно вычислимые функции (отметим, что справедливость Тезиса Черча была подкреплена теми математическими фактами, что в ламбда-языке оказались выразимы все функции, определяемые в других альтернативных моделях вычислимости и наоборот). Близкую к ламбда-исчислению модель вычислимых функций, носящую название *комбинаторной логики,* в 1940 г. создал Х. Карри. Обе эти модели вычислимости некоторое время развивались как чисто математические теории и только в начале 1960-х гг. Д.Маккарти создал на основе языка ламбда-исчисления язык функционального программирования *ЛИСП* (LISP — LISt Processing), а Д. Тернер на базе комбинаторной логики реализовал низкоуровневый язык для описания трансляторов языков функционального программирования. В последствии на идеях лямбда-исчисления и комбинаторной логики было создано еще несколько языков, в том числе и функциональные системы программирования (FP-системы), позволяющие работать с функциями высших порядков.

Многообразие созданных к 90-м годам прошлого столетия языков функционального программирования послужило причиной реализации проекта создания функционального языка, в котором были объеденены наиболее интересные идеи ФП. Так в начале 90-х гг. появился функциональный язык программирования, названный *HASKELL* в честь Хаскелла Карри. Тем не менее, следует признать, что основной вклад в развитие парадигмы ФП внес язык *ЛИСП*, который на протяжении многих лет оставался наиболее популярным функциональным языком программирования. Этот язык был использован в большом числе исследований и разработок в области ИИ при решении таких задач, как обработка естественного языка, создание экспертных систем, распознавание образов и многих других. Однако заметим, что, строго говоря, и *ЛИСП,* и HASKELL не являются чисто декларативными языками, поскольку в них разрешается использовать императивные конструкции, оправдывая это тем, что они позволяют создавать весьма эффективные практические приложения. На практике же это приводит к излишней сложности используемых конструкций, поскольку исчезает их первичная декларативная ясность. Кстати, аналогичный эффект имеет место и при добавлении в императивные языки функциональных средств. Заметим, что попытки создания языка, совмещающего в себе сильные стороны функционального и императивного программирования, продолжаются до сих пор.

Итак, что же предлагает нам современное ФП с позиций декларативного описания задач и процессов их решения. В ФП действия представляются в виде функций, которые одни данные (аргументы) преобразуют в другие данные (значения). При этом, в отличие от императивных языков, значения функций однозначно определяются их аргументами и не зависят как от истории вычислительного процесса, так и от внешних по отношению к вычислительному процессу сигналов и состояний. В то же время имеется и важное отличие программно реализованной функции от ее математического аналога – «программистская» функция должна быть определена *явно*, т.е. таким образом, чтобы

всегда была возможность извлечь из ее описания способ ее вычисления. Отсюда, в ФП исходные условия задачи описываются в виде конечного набора явных определений функций, (база данных), а цель задачи задается в виде запроса на вычисление значений так называемого целевого выражения с заданными параметрами и возможно содержащего обращения как к базовым функциям, так и к функциям, определенным в базе данных. Таким образом, общая схема формулирования и решения задачи в ФП сводится к:

- функциональной спецификации задачи, т.е. к написанию ее исходных условий в виде конечного набора определений функций,
- к написанию запроса и заданию его параметров,
- вычислению значений данного запроса.

При этом процесс вычисления с математической точки зрения представляет собой процесс исполнения так называемых правил переписывания, который можно интерпретировать как логический вывод искомого результата из аксиоматической теории, в роли которой выступает набор определений функций.

Похоже, но несколько иначе, осуществляется постановка и решение задач в другой парадигме декларативного программирования – в так называемом логическом программировании (ЛП). Его математическую основу составляет акиоматическая модель вычислимости, базирующаяся на понятии вычислимого множества. Логическая программа представляет собой формульное описание вычислимого отношения между параметрами в терминах формального логического языка. При этом разница между параметрами, объявленными входными, и параметрами, рассматриваемыми как выходные, в логической программе достаточно условна, что и отличает парадигму логического программирования от функционального. Дело в том, что вычисление функции всегда является строго направленным процессом – мы подаем на вход значения аргументов функции и на выходе должны получить результат, т.е. вычисленные значения функции. В логических же языках допускается ситуации, когда вначале указывается значение выходного параметра и ищется такое значение входного, при котором имеет место заданное отношение. Другое важное отличие логического программирования от функционального – это неоднозначность логических вычислений, поскольку удовлетворяющих исходному запросу значений переменных у логической программы может оказаться несколько.

В логическом программировании используется так называемое формульное описание вычислимых отношений с использованием достаточно узкого подмножества языка исчисления предикатов первого порядка. Заметим, что для логической программы, как и для функциональной, также имеют место понятия «база данных» и «запрос». При этом база данных (т.е. исходные условия задачи) в ЛП задается конечным набором логических правил и фактов, записанных в виде логических формул узкого подмножества языка исчисления предикатов, которые рассматриваются как аксиомы предметной области. Запрос также представляет собой логическую формулу этого языка, которая может содержать переменные, а целью решения задачи, специфицированной в виде логической базы данных и формулы-запроса, является попытка доказать, что запрос есть либо *логическое следствие* аксиом, т.е. является *теоремой* данной системы аксиом (с определением желательно всех вариантов означивания переменных в запросе), либо убедиться в том, что запрос не является логическим следствием этой аксиоматической теории.

Первые попытки реализовать отдельные идеи логического программирования делались еще в начале 60-х годов прошлого столетия. Значительный вклад в становление ЛП внес Дж. Робинсон, который для языка исчисления предикатов первого порядка в 1965 г. разработал алгоритм унификации и на его основе создал метод резолюций. Используя этот метод в 1971 г. А.Колмероз и Ф.Руссел создали первую версию языка ПРОЛОГ

(Prolog — PROgrammation en LOGique), написанную на ФОРТРАНе. Сразу же отметим, что ПРОЛОГ, как и ЛИСП, является квазидекларативным языком, поскольку в нем также достаточно активно используются императивные конструкции. Основополагающей для становления ЛП как новой парадигмы декларативного программирования стала публикация Р.Ковальским в 1974 г. статьи, в которой он показал, что для целей ЛП достаточно ограничиться использованием лишь *хорновских дизъюнктов,* т.е. формул вида P_1 & ... &P_n -> Q.

Пик популярности идей ЛП пришелся на 80-е годы прошлого столетия, свидетельством чему явился японский проект создания ЭВМ 5-го поколения. В предшествующие этому проекту и последующие годы наметилось несколько направлений развития ЛП, среди которых следует выделить следующие:

- *совершенствование и модификация* языка ПРОЛОГ (расширение базы языка, использование более мощных логических средств, постулирование модульности логических программ и т.п.);
- *параллельное* логическое программирование (заметим, что логические программы по своей природе являются параллельными);
- *комбинирование* логического стиля программирования с функциональным.

Остановимся на последнем направлении. Следует отметить, что уже с момента появления логического и функционального языков программирования неоднократно предпринимались попытки объединить эти две декларативные парадигмы. Как правило, главной целью всех предлагаемых вариантов объединения являлась возможность использования более широкого набора стратегий управления вычислениями, например, эффективной стратегией ленивых вычислений, характерной для функционального программирования, или стратегией параллельных вычислений, изначально свойственных логическим программам. Однако, серьезной проблемой при подобных объединениях оказалась проблема единообразия используемых структур данных, а также решение проблемы их типизации. Именно по этой причине в объединенных декларативных языках наиболее распространенными структурами данных оказались списки и деревья. Что же касается проблемы типизации данных, то в декларативном программировании реализуются оба возможных варианта ее решения – либо полное игнорирование понятия типа данных, либо, наоборот, следование строгой и жесткой дисциплины их определения и использования. Кстати, заметим, что отсутствие в декларативном языке понятия типа данных (так называемые *бестиповые языки*) имеет свое преимущество, поскольку в таких языках на множество всех его объектов (данные, действия, программы) можно смотреть как на *единое пространство*. Яркими примерами таких языков являются *ЛИСП* и *ПРОЛОГ*, у которых единственной формой представления объектов являются символьные выражения, что и позволяет программе обрабатывать и преобразовывать другие декларативные программы и даже саму себя -- знаменитая *проблема самоприменимости*. Отметим, что эта особенность декларативных бестиповых языков активно используется в ИИ и именно поэтому эти языки так популярны в ИИ. Кстати, из-за бестиповости декларативных языков серьезной математической задачей в свое время оказалась задача построения строгой математической семантики для таких языков. Эта проблема была практически одновременно решена в 70-е годы известными математиками – в России академиком Ю.Л.Ершовым (теория А-пространств), в Англии - Дана Скоттом (теория информационных систем).

Итак, подведем итог вышесказанному. Ключевой идеей декларативного программирования является то обстоятельство, что программа в нем является ***теорией***, а вычисления представляют собой ***вывод*** в этой теории. Понятно, что чем меньше ограничений накладывается на допустимые способы построения таких теорий, тем проще пользователю специфицировать задачу. Но, к сожалению, имеет место и другое

наблюдение – чем меньше ограничений, тем менее эффективным будет алгоритм интерпретации теорий и, следовательно, алгоритм организации поиска вывода. Данное противоречие заставляет сторонников и апологетов декларативного программирования искать разумный компромисс между удобством и выразительностью существующих декларативных языков и их эффективностью. Такой поиск может осуществляться либо в рамках существующих парадигм функционального и логического программирования, либо в их критическом осмысливании и, возможно, создании новой парадигмы.

Как известно, фундаментальным изучением понятия вычислимости в математике занимается математическая логика в рамках двух основных подходов: *аксиоматическом* и *теоретико-модельном*. Как было замечено выше, именно аксиоматический подход составляет математическую основу декларативного программирования. Но ведь в качестве концептуальной основы технологии компьютерного решения задач можно выбрать и теоретико-модельный подход, взяв, скажем, в качестве математической базы возможность *формульной определимости вычислимости на конструктивных моделях*, а сам процесс вычислимости мыслить, как процесс проверки *истинности формулы* на конструктивной модели. Эта идея и была положена в основу концепции ***семантического моделирования***, выдвинутой в 80-х годах прошлого столетия (авторы в то время использовали термин «семантическое программирование») сотрудниками ИМ СО РАН академиками РАН С.С.Гончаровым, Ю.Л.Ершовым и д.ф.-м.н. Свириденко Д.И. [14-20]. Коротко, суть этой концепции заключаться в следующем:

1. Предполагается наличие исходной базовой модели, выступающей как ядро системы, средствами которой будут специфицироваться и решаться задачи. При этом предполагается, что все объекты данной модели (элементы, функции, предикаты) и операции над ними являются *конструктивными*, т.е. вычислимыми. Заметим, что в семантическом моделировании данная модель рассматривается вместе со своей *списочной надстройкой*, состоящей из наследственно-конечных списков, порожденных элементами базовой модели. Допускается интерпретация некоторых предикатов базовой модели как *оракулов*;
2. Исходным условиям задачи сопоставляется смешанное формульно-термальное, т.е. фактически логико-функциональное определение новой модели с использованием так называемых $\sum$-*формул* и $\sum$-*термов* языка исчисления предикатов первого порядка сигнатуры исходной базовой модели. При этом отметим, что термальная часть используемого $\sum$-языка значительно расширена путем допущения *условных* и *рекурсивных* термов [21,22]. Допускается также рекурсивное описание не только новых функций, но и определяемых предикатов;
3. В качестве ответа на формулу-запрос или терм-запрос, записанные либо как $\sum$-формула, либо как $\sum$-терм, алгоритмически ищутся в первом случае такие конкретизации свободных переменных формулы, для которых данная $\sum$является ***истиной,*** или вычисляется значение $\sum$-терма-запроса.

Сразу же отметим, чтг на практике мы предлагаем ограничиться классом так называемых Δ_0-*формул* и Δ_0-*термов*. Напомним, что отличие Δ_0-формул от $\sum$-формул заключается в том, что при написании Δ_0-формул разрешается использовать только *ограниченные кванторы* существования и всеобщности. Причина такого ограничения заключается в желании с самого начала ограничиться эффективными вычислениями, т.е. вычислениями заданной, например, полиномиальной или автоматной сложности.

Итак, в предлагаемом подходе главными выступают процесс формульно-термального описания некой проблемной области в виде конструктивной модели и процедура либо проверки истинности $\sum$-формулы-запроса на этой модели, либо вычисления $\sum$-терма в этой модели. Практика показала, что такой модельный подход позволяет вполне адекватно отразить и полностью сохранить *исходную семантику задачи*. Именно поэтому он называется ***семантическим моделированием***.

Концепция семантического моделирования получила свое воплощение в целой серии теоретических и прикладных работ. В настоящее время продолжается дальнейшее развитие ее теоретических и прикладных аспектов, в том числе, в направлении создания практических технологий семантического моделирования и их адаптации применительно к особенностям различных конкретных проблемных областей -- ритейл, финтех, медицина, техника, умные контракты и кошельки, криптоэкономика, ТРИЗ и другие [23-30].

Одним из главных направлений развития семантического моделирования в настоящее время является его применение и к проблемам ИИ. В результате исследований в этом напралении возникла альтернативная традиционному искусственному интеллекту концепция ***«объясняющего» искусственного интеллекта (Explanable Artificial Intelligence)***, целью которого является создание таких технологий ИИ, в том числе и технологий обучения компьютеров, которые базировались бы на исходной семантике решаемой задачи и позволяли бы автоматически синтезировать семантически обусловленное и понятное человеку ее решение, при этом автоматически адаптирующиеся к изменению условий решения этой задачи**.** И такие технологии, основанные на оригинальных математических результатах, обобщающих логико-математический и вероятностный анализ формальных понятий, были созданы. При этом полученные математические результаты позволили нам не только создавать эффективные методы решения задач ИИ, свободные от указанных ранее недостатков традиционных технологий ИИ, но и, что очень важно для дальнейшего развития ИИ, разработать уникальный подход к решению проблемы ***самообучения*** систем ИИ**.**

Что касается практического применения концепции «объясняющего» ИИ, то успешными примерами здесь могут служить работы по автоматизации и глубокой роботизации бизнес-отношений, а также работы в области умных контрактов и умных кошельков нового поколения, осуществляемых ИМ СО РАН совместно с иркутскими коллегами из Иркутского Государственного Университета и группой компаний АЙЛАЙН (смотри, например, проекты **Bsystem** (www.inbox.io) и **Kirik** (www.kirik.io)). С помощью семантического моделирования и методов «объяснительного» искусственного интеллекта в настоящее время решается также широкий круг и других теоретических и прикладных проблем, например, решение совместно с НГУ проблем анализа сложных и масштабных коммуникационных сетей и платежных систем.

Важно отметить, что семантический подход позволяет, в отличие от аксиоматического варианта, осуществлять эффективный синтез решений задач заранее оговоренной сложности, например, полиномиальной или автоматной. Более того, в рамках семантического моделирования можно ставить и решать задачу создания «почти везде» эффективных алгоритмических решений, скажем, алгоритмов автоматной или полиномиальной сложности на «почти всех» исходных данных. Эти исследования, как и многие другие носят уникальный характер.

Предлагаемый нами семантический подход не отрицает использование в нем и элементов аксиоматического подхода. Более того, мы активно пользуемся воможностью эффективного синтеза обоих подходов. Так, например, при проведении исследований по объяснительному ИИ выяснилось, что результаты, полученные в рамках предлагаемого нами синтетического подхода, дают нам значительное преимущество перед чисто математико-статистическим или чисто аксиоматическим подходами, поскольку позволяют вместо традиционного машинного (глубокого) обучения типа «черный ящик» предложить принципиально другой подход к проблеме машинного обучения, основанный на сочетании математико-статистического и логического подходов (смотр, например, [24-28]). Такое сочетание позволяет строить иерархию логико-вероятностных классов, которые, с одной стороны, легко ***понимаемы*** человеком как совокупность логико-вероятностных утверждений, а, с другой стороны, более точно, на основе самообучения, формируют

иерархию «естественных» классов, тем самым, теоретически, по своим последствиям, намного превосходящих глубокое обучение, основанное на традиционных нейронных сетях. Проведенные предварительные эксперименты полностью подтвердили этот теоретический вывод. Более того, эти эксперименты показали также, что такое семантически-«прозрачное» машинное (глубокое) обучение адекватно формализует не только «естественную» классификацию, но и даже процесс реального восприятия, описанный в психологической литературе.

Последнее обстоятельство позволяет нам вплотную подойти к решению труднейшей проблемы ИИ – проблемы ***саморефлексии*** систем ИИ. Заметим, что успешное решение этой проблемы даст возможность говорить о достижении искусственным интеллектом следующего уровня развития, который можно условно назвать ***искусственным интеллектом 3.0*** (если считать, что развиваемый нами «объясняющий» ИИ - это ***искусственный интеллект уровня 2.0***). Именно на уровне 3.0 развития технологий ИИ появится возможность создавать системы ИИ, умеющие не только ставить перед собой задачи и осуществлять поиск их решения, но и, что принципиально важно, ***объяснять*** найденные решения. Отдельные элементы ИИ 3.0 в рамках семанического подхода создаются нами уже сейчас. Мы надеемся, что уже в течение ближайших 10-15 лет будет получено решение таких важнейших проблем, как проблема *самоидентификации* и *самоприменимости*, включая решение проблемы *целеполагания,* ведущих к удовлетворительному решению проблемы ***саморефлексии.***

ЛИТЕРАТУРА:

[1] Логический подход к искусственному интеллекту: от классической логики к логическому программированию. Пер. с франц./ Тейз А., Грибомон П., Луи Ж. и др.- М.; Мир, 1990. – 432 с.

[2] Стюард Рассел, Питер Норвиг. Искусственный интеллект. Современный подход. 2-е изд.: Пер.с англ –М.; Издательский дом «Вильямс», 2007. – 1408 с.

[3] Mariusz Flasinski. Introduction to Artificial Intelligence. Springer International Publishing, Switzerland. 2016. – 321 p.

[4] Miroslav Kubat. An Introduction to Machine Learning. 2 Edition; Springer International Publishing AG. 2017. – 348 p.

[5] Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация. 4-е изд. СПб.:Питер, 2003

[6] Зюзысов В. М. Математическое введение в декларативное программирование: учебное пособие. — Томск: ТГУ, 2003. — 83 с.

[7] Хендерсон П. Функциональное программирование. Применение и реализация: Пер. с англ. -М.: Мир, 1983. -349с.

[8] Хювенен Э.,Сепянен И. Мир ЛИСПа в 2-х т. - М.:Мир,1990

[9] Филд А.,Харрисон П. Функциональное программирование: Пер. с англ. - М.:Мир, 1993

[10] Логическое программирование. Сборник статей. (Под ред. В. Н. Агафонова). — М.: Мир, 1988. — С. 368.

[11] Братко И. Программирование на языке ПРОЛОГ для искусственного интеллекта. – М.: Мир,1990

[12] Малпасс Д.Р. Реляционный язык ПРОЛОГ и его применение. – М.: Наука, 1990.

[13] Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ. – М.: Наука, 1990.

[14] S. S. Goncharov, D. I. Sviridenko, Σ-programming, Transl., II.Ser., Am. Math. Soc., Vol. 142, 1989, 101–121.

[15] S. S. Goncharov, D. I. Sviridenko, Theoretical aspects of Σprogramming, Lecture Notes in Computer Science, 215 (1986), 169–179.

[16] Yu. L. Ershov, S. S. Goncharov, D. I. Sviridenko, Semantic programming, Information processing, Proc. IFIP 10 th World Comput. Congress, Dublin, Vol. 10, 1986, 1113–1120.

[17] С. С. Гончаров, Д. И. Свириденко, Математические основы семантического программирования, Доклады АН СССР, 289:6 (1986), 1324-1328 (перевод S. S. Goncharov, D. I. Sviridenko, Mathematical principles of semantic programming, Doklady Akademii Nauk SSSR, 289:6 (1986), 1324–1328).

[18] Yu. L. Ershov, S. S. Goncharov, D. I. Sviridenko, Semantic foundations of programming, Lecture Notes in Computer Science, Vol. 278, 1987, 116–122.

[19] Yu. L. Ershov, The principle of Σ-enumeration. Soviet. Math. Dokl., Vol. 27, 1983, 670–672. [7] Yu. L. Ershov, Dynamic logic over admissible sets. Soviet. Math. Dokl., Vol. 28, 1983, 739–742.

[20] Yu. L. Ershov, Definability and Computability, Siberian School of Algebra and Logic, Plenum, New York, 1996, 264 p.

[21] Гончаров С.С. Условные термы в семантическом программировании, СМЖ, 2017, т. 58, № 5, стр. 1026-1034. (Goncharov S.S. Conditional terms in semantic programming, СМЖ, 2017, v. 58, № 5, p. 1026-1034)

[22] Гончаров С.С., Свириденко Д.И. Рекурсивные термы в семантическом программировании, СМЖ, 2018 (в печати),

[23] Малых А.А., Манцивода А.В. Документное моделирование, Известий ИГУ (серия «Математика»), 2017. (Malykh AA, Mantsivoda AV Document modeling, Izvestiya ISU (series "Mathematics"), 2017)

[24] Vitaliy V. Martynovich, Euvgeniy E. Vityaev. Recovering Noisy Contexts with Probabilistic Formal Concepts // Proceedings of the 2nd International Workshop on Soft Computing Applications and Knowledge Discovery (SCAKD 2016), Moscow, Russia, July 18, 2016. CEUR Workshop Proceedings, v. Vol-1687, pp. 24-35.

[25] E. E. Vityaev, V. V. Martinovich. Probabilistic Formal Concepts with Negation // A. Voronkov, I. Virbitskaite (Eds.): PCI 2014, LNCS 8974, 2015, pp.385-399.

[26] Витяев Е.Е., Неупокоев Н.В. Формальная модель восприятия и образа как неподвижной точки предвосхищений.Подходы к моделированию мышления. (сборник под ред. д.ф.-м.н. В.Г. Редько). УРСС Эдиториал, Москва, 2014г., стр. 155-172. (Vityaev E.E., Neupokoev N.V. Formal model of perception and image as a fixed point of anticipation. Approaches to the modeling of thinking. (a collection edited by Doctor of Physical and Mathematical Sciences VG Redko). URSS Editorial, Moscow, 2014, page 155-172.)

[27] Витяев Е.Е., Мартынович В.В. Формализация "естественной" классификации и систематики через неподвижные точки предсказаний // СИБИРСКИЕ ЭЛЕКТРОННЫЕ МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ (Siberian Electronic Mathematical Reports), Том 12, Институтом математики им. С. Л. Соболева СО РАН, 2015, стр. 1006-1031.(Vityaev EE, Martynovich VV Formalization of the "natural" classification and taxonomy through fixed points of predictions // Siberian Electronic Mathematical Reports, Vol. 12, S.L. Soboleva Institute of Mathematics of the SB RAS, 2015, pp. 1006-1031.)

[28] Е.Е. Витяев, В.В. Мартынович. Прозрачное глубокое обучение на основе вероятностных формальных понятий в задаче обработки естественного языка // Известия ИГУ. Серия математика. - 2017. - Том 22. С.31-49.(HER. Vityaev, V.V. Martynovich. Transparent profound learning based on probabilistic formal concepts in the task of processing natural language // Izvestiya IGU. A series of mathematics. - 2017. - Vol. 22. C.31-49.)

[29] Свириденко Д.И., Сибиряков В.Г. (Sviridenko D.I., Sibiryakov V.G.) ТРИЗ- теория решения инновационных задач: часть 1. Что такое инновационная задача. Сибирская финансовая школа. Менеджмент и инновации (2017 г.) №4/123, стр.21-37 (TRIZ-Theory of Innovation Solutions: Part 1. What is an Innovative Task. Siberian financial school. Management and Innovation (2017) №4 / 123, pp.21-37)

[30] Свириденко Д.И., Сибиряков В.Г. (Sviridenko D.I., Sibiryakov V.G.) ТРИЗ- теория решения инновационных задач: часть 2. Как решать инновационные задачи: разработка концепции инновации. Сибирская финансовая школа. Инновации (2017 г.). №3/122, стр.26-35 (TRIZ-Theory of Innovative Problems Solution: Part 2. How to solve innovative problems: the development of the concept of innovation. Siberian financial school. Innovations (2017). №3 / 122, pp.26-35)