Васильев С.Н.,

г.н.с, ИПУ РАН им. В.А. Трапезникова

Смирнова Н.В.,

н.с., ИПУ РАН им. В.А. Трапезникова

Суконнова А.А.,

ст. инж., ИПУ РАН им. В.А. Трапезникова

Душкин Д.Н.,

инж.-прогр, ИПУ РАН им. В.А. Трапезникова

Абраменков А.Н.,

ст. инж.-прогр.; ИПУ РАН им. В.А. Трапезникова
snv@ipu.ru

Методы интеллектуализации обучающих систем

Аннотация

Предлагаются методы интеллектуализации следящих систем обучения предметам естественнонаучного цикла. Подробно описываются логические методы автоматизации решения задач и планирования действий, а также эвристические методы проверки решений задач с привлечением инструментов символьных вычислений.

1.Введение

В данной работе рассматриваются методы интеллектуализации обучающих систем, хотя ряд рассматриваемых здесь методов имеет более широкую применимость. Одним из наиболее перспективных видов обучающих систем представляются так называемые следящие ИОС, т.е. такие обучающие системы, которые сравнивают шаги найденного решения, найденного самой системой, с шагами решения, получаемого обучаемым, для проверки хода решения обучаемого на завершенность и правильность.

К следящим ИОС относится система «Волга», разрабатываемая в Институте проблем управления им. В.А.Трапезникова РАН в сотрудничестве с Казанским государственным техническим университетом им. А.Н.Туполева и психологическим факультетом Московского государственного университета им. М.В.Ломоносова [1]. В решателях задач ИОС, способных отыскивать решения автоматически, могут задействоваться разные методы. В случае, когда это методы автоматического доказательства теорем (АДТ), в частности, дедукции или абдукции, ИОС именуется как система, основанная на АДТ (см., например, [2]). Существуют решатели, использующие более широкий спектр методов и эвристик, которые изначально разрабатывались психологами для

исследования процессов человеческого мышления, называемые когнитивными. Соответствующие ИОС именуются системами, основанными на когнитивных архитектурах (см., например, [3]).

ИОС «Волга» относится к гибридным следящим системам, поскольку в ней для проверки решений студентов используются не только логические решатели, но и ряд эвристических методов, задействующих библиотеку символьных вычислений SymPy[4]. В п. 2-4 статьи описывается первая версия подсистемы проверки решений, позволяющая не только установить правильность введенного студентом фрагмента решения, но также определить, какое из заложенных в системе решений наиболее близко к решению студента, является ли решение студента завершенным и достаточно развернутым. В п. 5 рассматриваются методы АДТ, автоматического планирования действий и автоматического поиска недостающих средств достижения текущей подцели. Областью применения методов АДТ является автоматическая генерация возможных решений учебных задач и планирование действий при управлении учебным процессом.

2. Обзор результатов, полученных другими коллективами.

Одной из наиболее развитых систем, в которой реализована поддержка ввода и проверка решений задач в достаточно свободной форме, является ANDES Tutor[5]. Эта система спроектирована для обучения вводному университетскому курсу физики, рассчитанному на два семестра. Как отмечают разработчики ANDES [6], в ранних версиях ANDES правильность уравнений, вводимых студентами, определялась по тому, является ли эквивалентной формула студента одной из формул из некоторого списка.

Данный список генерировался на этапе настройки системы путем применения набора правил, задающих возможные алгебраические манипуляции, к набору основных, или «канонических», уравнений. Этот метод требовал создания широкого списка возможных комбинаций формул из набора «канонических» уравнений. Даже для простых учебных задач в базу данных обучающей системы требовалось поместить слишком много возможных решений, поэтому разработчики ANDES решили сконструировать другую подсистему (описание этой подсистемы см. в [6]). Мы считаем, что вышеописанной проблемы можно избежать (см. п. 4). Что касается известных проблем алгоритмической неразрешимости массовой задачи эквивалентности термов и формул той или иной рассматриваемой теории, то еще в 1970-ых годах (В.М. Глушков и др.) отмечалась полезность понятия практической разрешимости, учитывающей эвристики и вычислительную сложность задач, в сравнении с зачастую практической неразрешимостью ряда теоретически разрешимых задач. Поэтому для ИОС «Волга» была разработана подсистема, в которой правильность вводимых студентами формул определяется путем сравнения их на эквивалентность с формулами, присутствующими в возможных решениях учебной задачи.

Эквивалентность формул в ИОС «Волга» определяется с помощью библиотеки символьных вычислений SymPy. Идея привлечения инструментов символьных вычислений к проверке ответов студентов путем сравнения выражений на эквивалентность не нова. По-видимому, самыми широко известными обучающими системами, в которых используется данный подход, являются ActiveMath[7] и STACK[8]. Между тем, в этих системах в качестве решения задания допускаются только только «одношаговые» задачи.

К сожалению, в отличие от системы компьютерной алгебры Maxima, которая используется в STACK, библиотека SymPy не позволяет отключать некоторые аксиомы при проверке на эквивалентность формул. Отключение аксиом коммутативности и ассоциативности полезно при определении того, соответствует ли формула студента требованиям преподавателя – например, является ли она достаточно упрощенной, обладает ли требуемым порядком слагаемых. Преимуществами SymPy являются высокий уровень интероперабельности и надежность работы в условиях многопользовательского веб-приложения. Поэтому в архитектуру ИОС «Волга» была встроена библиотека SymPy.

3. Организация ввода и проверки решений студентов в ИОС «Волга».

В ИОС «Волга» фрагменты решения вводятся в систему с помощью специального элемента интерфейса (см. рис. 1). Фрагмент решения, введенный с помощью этого элемента интерфейса, называется шагом студента. Для студентов предусмотрено два вида интерфейса ввода шага: «трудный» и «легкий». Далее будет описан только «трудный» интерфейс. Интерфейс ввода шага включает текстовое поле ввода и область, в которой отображается визуальное представление вводимой формулы, динамически формируемое с помощью скрипта Mathjax. Правила ввода формул похожи на синтаксис LaTeX.

Возможные решения учебных задач представляются посредством коллекции экземпляров класса «Этап». Правильность шага определяется путем проверки того, является ли формула студента эквивалентной формулам из некоторого списка (об этом подробнее далее). Эквивалентность формул проверяется с помощью функции *simplify* библиотеки SymPy.

Перед проверкой на эквивалентность выполняется предобработка сравниваемых формул. Поскольку синтаксис Mathjax немного отличается от синтаксиса Sympy, к тому же правила ввода формул были несколько упрощены, предобработка включает замену некоторых символьных последовательностей на другие (например, «^» на «**»). Кроме того, во время предобработки все обозначения задачи в сравниваемых формулах заменяются на соответствующие символы «x1y», …, «xny» (при вводе формул студент может использовать только обозначения из определенного списка, отображаемого в окне пользовательского интерфейса решения

задачи). Такая замена позволяет не только избежать ситуаций, в которых присутствие обозначений, содержащих не поддерживаемые SymPy символы (например, «p(a,b)») вызывает исключения во время проверки эквивалентности, но и эффективно вставлять в формулу студента забытые им знаки умножения.

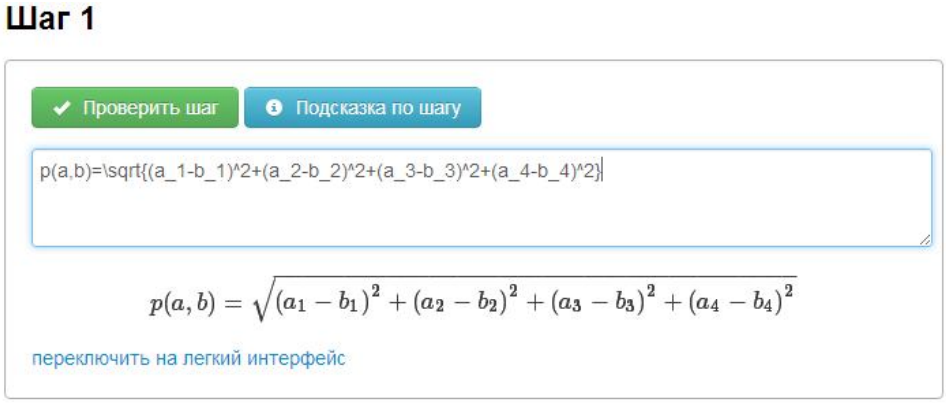

Рис. 1. *Пример шага студента.*

Заметим, что в некоторых случаях в дополнение к стандартной процедуре проверки на эквивалентность двух выражений запускаются дополнительные эвристики. Если часть формулы этапа содержит префикс «#almost#», запускается эвристика, проверяющая путем сравнения количеств операций сложения, умножения, вычитания и возведения в степень, является ли соответствующая часть формулы студента настолько же развернутой. В случае присутствия префикса «#fixed#» запускается эвристика, проверяющая, является ли часть формулы студента такой же, как и часть формулы этапа в «строгом» смысле. Эта эвристика проверяет на точное совпадение две строки, соответствующие сравниваемым частям формул этапа и студента с предварительно удаленными из них пробелами. Например, когда формула этапа «#fixed# x + y = #almost# a + b», то тогда, если формула студента

x + y = a + b, шаг студента введен правильно,

y + x = a + b, шаг студента введен неправильно,

x + y = b + a, шаг студента введен правильно,

x + y = a + b + 1 – 1, шаг студента введен неправильно.

Впервые необходимость запуска дополнительных эвристик после стандартной проверки на эквивалентность возникла при анализе решений студентов на достаточную развернутость (для того, чтобы предотвратить списывание, преподаватели могут требовать от студентов со слабой успеваемостью предоставления более развернутых решений) Рассмотрим следующий пример.

Условия задачи: Вычислить расстояние между векторами $a=\begin{pmatrix}1\\2\end{pmatrix}$ и $b=\begin{pmatrix}5\\6\end{pmatrix}$

Одно из возможных решений:

$$p(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \quad (1)$$

$$p(a,b) = \sqrt{(1-5)^2 + (2-6)^2} \quad (2)$$

$$p(a,b) = \sqrt{(-4)^2 + (-4)^2} \quad (3)$$

$$p(a,b) = \sqrt{32} \quad (4)$$

Формулы (2)-(4) соответствуют последовательному упрощению формулы расстояния между векторами, в которую подставили конкретные значения координат. Заметим, что для функции *simplify* формулы (2)-(4) являются попарно эквивалентными, и, соответственно, неразличимыми. Так, например, если преподаватель потребует, чтобы в решении студентов обязательно присутствовала формула (2), а решение студента будет состоять только из формулы (4), то проверяющим алгоритмом не будет установлено, что решение студента не соответствует требованиям преподавателя. Для решения этой проблемы в конфигурацию ИОС «Волга» были внесены описываемые ниже изменения.

Для шагов решения, соответствующих упрощению одного и того же выражения, создается только один объект «Этап решения». Если существует необходимость наложить условия на развернутость решения, то преподаватель, помимо основной формулы, в качестве дополнительных свойств этапа вводит в систему формулы, которые должны обязательно присутствовать в решении студента («обязательные» формулы). Для рассматриваемого примера имеем:

Этап 1.

Основная формула: (1).

Этап 2.

Основная формула: (2), «обязательные» формулы: (2)-(4)[20].

После стандартной процедуры проверки шага в случае его правильности, если шаг студента соответствует этапу с непустым множеством «обязательных» формул, формула шага студента «строго» сравнивается с каждой из «обязательных» формул этапа (т.е. при сравнении на эквивалентность считается, что каждая «обязательная» формула содержит префикс «#fixed#»). Если находится «обязательная» формула, «строго» совпадающая с формулой шага студента, то в системе сохраняется информация о том, что данный шаг студента реализует именно эту «обязательную» формулу. Решение студента считается достаточно развернутым, если в нем реализованы все «обязательные» формулы этапов решения, наиболее близкого к решению студента.

Решение, наиболее близкое к решению студента, характеризуется максимумом совпадений между множеством присутствующих в нем этапов

20 В зависимости от пожеланий преподавателя совокупность «обязательных» формул может быть другой.

и множеством различных[21] этапов, присутствующих в решении студента. Если студент еще не ввел правильных шагов, то наиболее близким считается решение, рекомендованное преподавателем в качестве опорного.

Этап, соответствующий шагу студента, определяется автоматически во время проверки шага. Для этого сначала проверяется, эквивалентна ли формула студента, каждой из формул, соответствующих этапам возможных решений задачи. Если оказывается, что формула студента эквивалентна нескольким формулам, тогда формулы дополнительно сравниваются на наличие эквивалентности в «строгом» смысле (т.е. с использованием префикса «#fixed#»). Если же формула студента эквивалентна (в обычном или «строгом» смысле) только одной из формул, соответствующих этапам возможных решений, тогда этот этап соответствует шагу студента.

4. Проблема множественных комбинаций формул, генерируемых студентами при решениях учебных задач

Рассмотрим эту проблему на примере задачи вычисления расстояния между векторами *a, b* в евклидовом пространстве.

Для решения этой задачи необходимо использовать следующие формулы:

$$p(a,b) = |c| \quad (5)$$

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} a_1 - b_1 \\ a_2 - b_2 \end{pmatrix} \quad (6)$$

$$|c| = \sqrt{(c,c)} \quad (7)$$

$$(c,c) = c_1^2 + c_2^2 \quad (8)$$

Комбинация формул сводится к замене величин в одной формуле алгебраическими выражениями, полученными из других формул, участвующих в комбинации. Например, комбинация формул (5) и (7) имеет вид: $p(a,b) = \sqrt{(c,c)}$, комбинация же формул (5), (7), (8), такова: $p(a,b) = \sqrt{c_1^2 + c_2^2}$.

Нетрудно придумать алгоритм, который бы сгенерировал все возможные решения студента, включающий все возможные комбинации данного набора формул. В данном случае было сгенерировано 16 решений. В решениях произвольных учебных задач может присутствовать несколько таких «базовых» наборов формул. Например, если возможные решения задачи можно описать 5-ю «базовыми» наборами формул, то преподавателю на этапе настройки системы придется заложить в базу данных 80 возможных решений. Рассмотрим возможный способ преодоления этой проблемы.

Возможности комбинации формул удобно представлять в виде графа, каждая вершина которого соответствует одной или нескольким величинам, а каждое ребро - одной формуле (см. рис. 2). Если в графе вершины a и b соединены ребром $a \leftarrow b$, это означает, что переменная a может быть вычислена по формуле, которая содержит величину b (значений других

21 имеется в виду термин языка SQL, см. опцию DISTINCT выражения SELECT.

величин при этом не требуется).

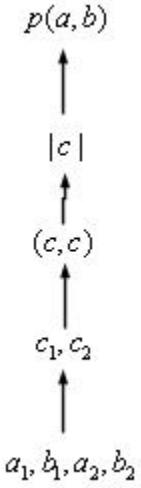

Рис. 2. *Возможности комбинации формул для задачи вычисления расстояния между векторами a и b евклидова пространства*

Если добавить в модель базы данных класс «Комбинаторный этап», который бы содержал информацию об этом графе, тогда процедура проверки решений может включать следующие шаги:

1. если существует вершина графа, величины которой совпадают с величинами в левой части уравнения студента, запомнить эту вершину (обозначим ее a),
2. если существует вершина графа, величины которой совпадают с величинами в правой части уравнения студента, запомнить эту вершину (обозначим ее b),
3. сформировать формулу, соответствующую «комбинаторному этапу», путем последовательной комбинации формул, двигаясь по графу от вершины a до вершины b,
4. применить стандартную проверку на эквивалентность формулы студента формуле «комбинаторного этапа».

Рассмотрим следующий пример. Пусть студент ввел формулу $|c|=c_1+c_2$. В левой части формулы студента находится $|c|$, в правой части - c_1, c_2. В этом случае формула «комбинаторного этапа» получается как комбинация формул (7) и (8) и имеет вид: $|c|=\sqrt{c_1^2+c_2^2}$.

Студент ввел неправильный шаг, поскольку формулы $|c|=\sqrt{c_1^2+c_2^2}$ и $|c|=c_1+c_2$ не эквивалентны.

Улучшение вышеописанной процедуры – тема следующих публикаций. Целью данного раздела статьи было показать преодолимость проблемы, возникающей при использовании ранее предложенного метода проверки решений студентов.

5. Логические методы интеллектуализации

Известны разные логические исчисления и методы поиска выводов,

ориентированные на компьютерную реализацию. В их числе – метод резолюций для представления и обработки знаний в языке дизъюнктов [9-12], метод представления и обработки знаний в языке по-формул (ПОФ-метод) [13] и другие. Хотя метод резолюций, как и ПОФ-метод, машинно-ориентирован, ПОФ-метод обладает рядом преимуществ, которые являются ключевыми для интеллектуализации компьютерных средств обучения на основе логического подхода. В частности, не уступая другим методам автоматического доказательства теорем в свойствах выразительности языка, а также корректности и полноты стратегий выводов, языковые и дедуктивные средства исчисления по-формул (по-формализм) характеризуются меньшей комбинаторностью пространства поиска выводов и лучше совместимы с эвристиками предметной области из-за того, что в отличие от языка дизъюнктов, используемого в методе резолюций, в языке L по-формул эвристическая структура знания не разрушается. Более полный перечень особенностей этого метода содержится в [13].

Язык L по-формул является полным языком первого порядка, формулы которого представляются как деревья. Каждый узел есть т.н. позитивный квантор (ПК), состоящий из знака квантора $(\forall,\exists)$, вектора связываемых этим квантором переменных и условия, накладываемого на значения этих и ранее квантифицированных переменных в виде конъюнкта, где под конъюнктом понимается множество (конъюнкция) атомов. Ветвления после ПК всеобщности (существования) понимаются как дизъюнкции (соответственно конъюнкции). ПК всеобщности (узлы-посылки) и существования (узлы-факты) чередуются вдоль каждой ветви формульного дерева. Без ограничения общности можно считать, что 1) корнем формульного дерева является ПК $\forall : True$ (набор переменных отсутствует, $True$ - тождественно истинный предикат и, если за корнем следует ветвление, то этот ПК понимается просто как дизъюнкция), 2) листьями дерева являются ПК существования и, в частности, $\exists : False$ ($False$ - тождественно ложный предикат), 3) ни один ПК, кроме листьев, не содержит $False$. Подробнее язык L описан в [13], а здесь рассмотрим лишь его пропозициональный фрагмент. Поэтому в ПК существования (узлах-фактах) кванторы $\exists$ не пишем, а в ПК всеобщности (узлах-посылках) вместо символа $\forall$ пишем знак вопроса.

Семантика по-формулы F прозрачна: она совпадает с теоретико-модельной семантикой соответствующего образа $F^{ИВ}$ (формулы F) в классическом исчислении высказываний (или образа $F^{ИП}$ в первопорядковом варианте языка L, см. [13]). Язык по-формул полон относительно классических выразительных возможностей [13].

Далее будет рассмотрен пропозициональный фрагмент исчисления по-формул J [13] с такой модификацией построения выводов, которая предложена в [14,15] и позволяет осуществлять распараллеливание вывода

с целью его ускорения.

Пусть по-формула F имеет вид, представленный на рис. 3, где в эллипсах – нераскрытые части формулы. Для определения правила ω будем считать, что A не совпадает с $False$ и обведенное пунктиром -непусто, хотя подформулы $\Phi, \Psi, \Sigma_1, ..., \Sigma_l$ могут быть пустыми. Содержание $\Psi, \Sigma_1, ..., \Sigma_l$ может быть разным, но их ветви должны начинаться с узлов-посылок, т.е. с вопросов, а ветви в Φ должны начинаться с узлов-фактов. Пусть в общем случае узел-посылка (вопрос) B имеет альтернативы (т.е. дизъюнктивное ветвление) как на рис. 3, где l i 1.

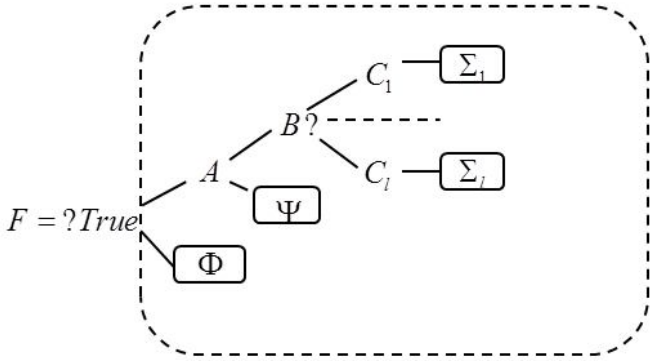

Рис. 3. *Общий вид по-формулы.*

Пусть содержимое вопроса B подтверждено в узле-факте A, т. е. $B \subseteq A$. Тогда по определению применения правила ω к вопросу B приводит к по-формуле, представленной на рис. 4. В соответствии с методом доказательства от противного, выводы в исчислении J ориентированы на опровержение отрицания доказываемого, поэтому в качестве аксиомы выбрана по-формула $True?-False.$ Конечная последовательность применения ω, приводящая к этой аксиоме, называется выводом.

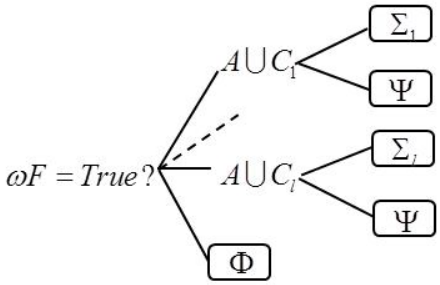

Рис. 4. *Результат однократного применения правила* ω *к* F.

Каждый шаг вывода формализует не столько специальный (локальный) переход от условий к действию (как, например, при использовании инструктивных знаний «если-то» в экспертных системах продукционного типа, выражающий более или менее очевидным образом отдельный шаг приближения к цели), сколько интуитивное представление человека о правильности (логичности) умозаключений, т.е. имеет более универсальный характер.

Применение правила вывода к узлу вопросу B с дизъюнктивным ветвлением привело к размножению узла-факта A. Обработку каждой

подформулы, начинающейся с узла $A \cup C_i, i = \overline{1,l},$ можно осуществлять независимым образом параллельно, например, на разных процессорах (естественный ИЛИ-параллелизм). В отличие от [13], в [14,15] предложено применение правила вывода ω и к вопросам, входящим в третий, пятый и более глубокие (нечетные) уровни вопросов формулы $F,$ например, на рис. 3 входившим в Φ, Ψ и $\Sigma_i, i = \overline{1,l}.$ Поясним это на примере (рис. 5).

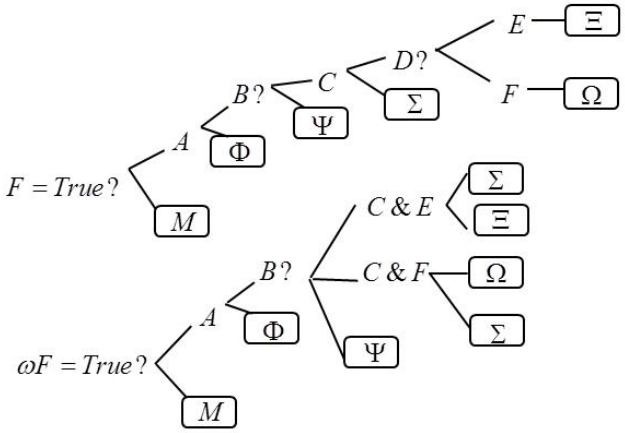

Рис. 5. *К расширению применимости правила ω для распараллеливания вывода.*

Пусть требуется ответить на вопрос D формулы $F.$ Тогда условием применимости правила ω к вопросу D будет вложимость конъюнкта D в объединение конъюнктов $A \cup B \cup C.$ Результатом применения ω к D будет формула $\omega F,$ представленная на рис. 5.

Теорема 1. F логически эквивалентно $\omega F.$

Если на каждом шаге применения правила ω каждая новая формула следует из предыдущей (хотя они даже эквивалентны), то вывод противоречия в конце цепочки применений правила ω означает противоречивость исходной формулы, что и требуется (по методу от противного).

Утверждение 1 (необходимый критерий опровержимости). Для противоречивости формулы F необходимо, чтобы хотя бы один лист древовидного представления F совпадал с $False.$

Доказательство очевидно. Действительно, т.к. ни один узел, отличный от листьев, не содержит предиката $False,$ то, если хотя бы в одном листе тоже нет этого предиката, то мы никогда в узле-факте 1-го уровня не получим противоречия.

Разумеется, наличие листьев с противоречиями не означает опровержимости по-формулы (т.е. указанный критерий не является достаточным, он только – необходимый). Например, следующая формула, очевидно, неопровержима, $True? - \text{A} - \text{B}? - False,$ если B не является подмножеством множества атомов A (не применимо правило вывода), что необходимо для проникновения $False$ в $A.$

При решении задач с неполной информацией (учебных или

управленческих в модуле управления) необходим механизм дооснащения дополнительными средствами, в том числе конструктивными средствами вычислимости в виде недостающих формул, средствами построения планов действий модуля управления и т.п. Это дооснащение реализуемо на основе сочетания правила ω с некоторым правилом абдуктивного типа. Такое сочетание правил вывода осуществляется в излагаемом ниже механизме решения логических уравнений (вначале – в классической семантике).

Рассмотрим пропозициональное уравнение $X \to A,$ где $(\neg A)^L$ – известная произвольная хорновская по-формула, т.е. без дизъюнктивных ветвлений, а X – неизвестная формула, подлежащая отысканию для обеспечения выводимости в исчислении высказываний (ИВ) формулы $X \to A,$ т.е. $\underset{\text{ив}}{\to} X \quad A$.

Рассмотрим следующую процедуру построения X по A с применением средств исчисления J. Поскольку обоснование $X \to A$ равносильно опровержению $(X \,\&\, \neg A)^L$, то к формуле $(\neg A)^L$ добавляется неизвестная V, т.е. рассматривается по-формула $\Omega = True?\ \ True\ \left((\text{ШИ})^L, V\right)$. Здесь и далее используется скобочная структура формул для представления ветвлений и других структурных элементов (вместо геометрического, как на рис. 3-5). По-формула $(\neg A)^L$ имеет общую структуру набора формул $B_1?(\Phi_1),...,B_m?(\Phi_m)$, где $m \geq 1$. Поначалу ω неприменимо (узел-факт первого уровня – пустой), поэтому в порядке конкретизации V формируем начальный фрагмент будущего решения $X = \left(\tilde{V}\right)^{\text{ив}}$:

$$\tilde{V} = True?\left(B_1(V_1),...,B_m(V_m)\right),$$

где V_i – новые неизвестные (подлежащие конкретизации).

После ответа на синтезированный вопрос $True$ формула Ω примет вид

$$True?(B_1\left(B_1?(\Phi_1),...,B_m?(\Phi_m), V_1\right),...,B_m(B_1?(\Phi_1),...,B_m?(\Phi_m),V_m)) \qquad (9)$$

где каждая подформула, начинающаяся с узла-факта B_i, является объектом независимого опровержения.

Каждая такая подформула после, по меньшей мере, одного применения ω (в том числе к вопросу B_i) примет один из следующих трех видов: либо 1) $False$, либо 2) $C(V_i)$, либо 3) $C\left(C_1?(\Psi_1),...,C_k?(\Psi_k),\ V_i\right), k$ і 1, с неприменимым более правилом ω.

В первом случае i-я ветвь в $\tilde{V}$ опровергнута и по критерию получения логически наислабейшей импликации формулы $\tilde{V}$ полагаем $\tilde{V}_i = \Lambda$, где Λ – пустое выражение.

Во втором случае полагаем $\tilde{V}_i = C?\,False$, после чего i-я ветвь снова опровергается с указанным результатом $\tilde{V}_i$. В третьем случае формируем

$\tilde{V}_i = C?\left(C_1\left(V_{i1}\right),...,C_k\left(V_{ik}\right)\right)$

и после ответа на этот вопрос узел-факт C снова размножится и каждый новый узел-факт $C \text{ И} C_l$ будет началом ветви

$C \text{ И} C_l\left(C_1?\left(\Psi_1\right),...,C_k\ ?\left(\Psi_k\right),\ \ V_{il}\right),\ \ l=\overline{1,k}$.

Дальше процесс продолжается аналогично описанному для (9). Правило синтеза решения $\tilde{V}$ назовём правилом α, а весь процесс – ограниченным (ω,α)-процессом, поскольку синтез очередного фрагмента решения $\tilde{V}$ осуществляется лишь в случае неприменимости правила ω.

Теорема 2. *Пусть дано пропозициональное логическое уравнение* $X \to A$ *, где* $(\neg A)^L$ *– известная хорновская по-формула,* X *– неизвестно. Тогда ограниченный* (ω,α)*-процесс* (*синтеза по-формулы* $\tilde{V}$) *конечен и приводит к необходимому и достаточному условию* $X = \left(\tilde{V}\right)^{\text{ИВ}}$ *, т. е.*

$$\square_{\text{ИВ}}\left(\tilde{V}\right)^{\text{ИВ}} \leftrightarrow A$$

Таким образом, при возникновении трудностей с доказательством A (ω,α)- процесс преодолевает их путем формирования и принятия некоторых дополнительных предположений X. Этот подход представляется характерным для содержательных рассуждений. Под трудностями доказательства понимаются не только собственно исчерпание ресурсов, но и, например, появление тех или иных признаков бесперспективности или невозможности дальнейшего доказательства (зацикливание, неприменимость правил вывода и т. п.). Даже при отсутствии ограничений на ресурсы в неразрешимых и полуразрешимых теориях возникает проблема принятия решения в случае, если доказываемая формула невыводима, а признаков этого не обнаруживается ни сразу, ни в процессе доказательства. При этом учет ресурсных ограничений оказывается полезным сам по себе для построения решающих правил, в частности, прерывающих процесс поиска вывода.

По своей постановке задача разработки алгоритмического метода синтеза гипотез X для выводимости формул $X \to A$ напоминает работы по автоматическому синтезу теорем [16], решению пропозициональных и первопорядковых логических уравнений [17], индуктивному логическому программированию [18], абдуктивному логическому программированию [19]. Однако (ω,α)- процесс по своему содержанию и сфере возможных приложений отличается от указанных работ. Некоторая первопорядковая версия (ω,α)- процесса описана в [20].

С логической точки зрения многие учебные задачи могут иметь

неклассическую семантику («доказать, что истинно…»), а конструктивную («найти…», «вычислить…», «построить…»). При этом должна быть возможность извлечения из логического вывода утверждения о достижимости цели искомого плана действий (плана отыскания, вычисления или построения). Примером конструктивной задачи является задача, цель которой специфицируется формулой $A \supset B$, а логический вывод должен обеспечить распознавание, какой конкретно из этих двух случаев имеет место. Так, классический вывод специфицируемой формулой $A \supset A$ цели, где, например, A - утверждение «Великая теорема Ферма верна» (как и при других A, например, когда A - утверждение «Данные два треугольника равны» и т.п.), ничего не дает и тривиален.

Поэтому нужны логики для решения задач в конструктивной семантике, широко возникающих не только на уровне отыскания решений учебных задач, но и для планирования действий самой системы.

В [13] выделен конструктивный фрагмент исчисления J (см. Приложение). В проекции на пропозициональный язык из Теорем 2.8 – 2.10 и Следствия 2.1 [13] очевидным образом вытекает следующее утверждение.

Утверждение 2. Если по-формула F - произвольная спецификация конструктивных средств достижения цели (т.е. каждому $\supset$-ветвлению сопоставлена упомянутая процедура распознавания), а по-формула G - спецификация цели из класса формул вида $D?(E_1,...,E_k)$, то всякий (классический) вывод в J формулы $((F)^{ив} \& \neg (G)^{ив})^L$ конструктивен в том смысле, что каждому $\vee$-ветвлению в G можно сопоставить процедуру распознавания в виде композиции процедур для F.

Нетрудно проверить, что из Утверждения 2 и Теоремы 2 вытекает следующее утверждение.

Утверждение 3. Пусть задача A имеет вид $F \to G$, где $(F)^L$ – хорновская по-формула, $(G)^L$ – из класса $B?(B_1,...,B_n)$. Тогда всякий ограниченный (ω, α)- процесс синтеза условия $\tilde{V}$ выводимости формулы $(F \& \neg G)^L$ конечен и конструктивен, а синтезируемое решение $\tilde{V}$ – спецификация искомого, логически минимального, конструктивного дооснащения.

Рассмотрим простейший пример применения ограниченного (ω, α)-процесса в задаче планирования действий в условиях недостатка конструктивных средств достижения цели.

Пример.

«В треугольнике известна длина a основания и площадь S. Высота h, опущенная на основание, совпадает с боковой стороной b. Найти длину второй боковой стороны c».

Пусть база знаний предметной области (планиметрии) не содержит формул, связывающих между собой величины a, h, S, дающих возможность конструктивного отыскания любой из них по двум другим, но есть формула

вычисления длины гипотенузы c через длины катетов a и b. Тогда, добавив к спецификации $b\&a\to c$ (спецификация минимальной базы знаний, выражающая вычислимость c через b и a) условия задачи $S\&a$, а также $(b\to h)\&(h\to b)$, что означает вычислимость b и h друг через друга (тривиальную, поскольку b и h совпадают), утверждение о вычислимости c в указанной вычислительной обстановке примет в ИВ вид

$A=(S\&a\ \&(b\&a\to c)\&(b\to h)\&(h\to b)\to c)$,

а отрицание этого утверждения в языке L

$$(\neg A)^{L}=(True?\{S,a\},\ \Phi),$$

где $\Phi=(\{b,a\}?c,\ \{b\}?\{h\},\ \{h\}?\{b\},\ \{c\}?False)$.

После применения ω к $\Omega=True?\ True\ \left((\text{Ш})^{L},V\right)$ получим $\omega\,\Omega=True?\ \{S,a\}\ (\Phi,V)$, где V – неизвестная, вводимая впрок - на случай неполноты средств для разрешимости задачи. Такой случай как раз здесь имеет место (данных хватает, а формульных зависимостей не хватает для вычислимости c). Поскольку теперь ω неприменимо, синтезируем спецификацию дооснащения

$\tilde{V}=\{S,a\}?\ \ (\{b,a\}(V_1),\ \ \{b\}(V_2),\ \ \{h\}(V_3),\ \ \{c\}(V_4))$.

Далее применяем α

$$\alpha\,\omega\Omega=True?\ \ \{S,a\}\ \ \left(\Phi,\tilde{V}\right)$$

и снова ω

$\omega\alpha\omega\Omega=True?\ \ (\{S,a,b\}(\Phi,V_1),\{S,a,b\}(\Phi,V_2),\{S,a,h\}(\Phi,V_3),\{S,a,c\}(\Phi,V_4))$.

Каждый из первых двух узлов-фактов (второго уровня) опровергается минимум за два применения правила ω, третий – за минимум три применения ω, а четвёртый – за минимум одно применение ω. Поэтому полагаем $V_i=\Lambda,\forall i=\overline{1,4},$ и окончательно получим решение

$\tilde{V}=\{S,a\}?\ \ (\{b,a\},\ \ \{b\},\ \ \{h\},\ \ \{c\})$,

означающее спецификацию недостающего средства вычисления по S,a либо 1) b, либо 2) h (что равносильно b), либо 3) c (что означает прямое вычисление c по S,a, т.е. путем применения явной формульной зависимости c от S,a).

Соответственно из вывода извлекается план решения задачи: ввести данные S,a (первое применение ω); убедившись в неприменимости более имеющихся знаний (данных) и конструктивных средств, синтезировать начальный (он же окажется и полным) фрагмент спецификации недостающих средств; ввести недостающие средства (применение α); использовать все средства (девять применений ω) с получением ответа c.

В частности, достаточным для «проталкивания» задачи над указанной «бедной» базой знаний является дооснащение вычислительным средством, способным по S и a вычислять h (по известной формуле

$h = 2S / a$).

Встраивание описанного (ω, α)-процесса как процедуры дооснащения для решения задачи может быть достаточно разнообразным, в том числе в управлении учебным процессом, например, для автоматического синтеза сценария диалога ИОС с обучаемым.

6. Заключение

В п. 2-4 статьи описана подсистема эвристической проверки решений обучаемого (студента), основанная на символьных преобразованиях. Она позволяет не только определить правильность введенного им фрагмента решения, но также определить, какое из априорно заложенных в систему решений наиболее близко к решению студента, является ли решение студента завершенным и достаточно развернутым. Качество работы предложенного способа проверки решений во многом зависит от возможностей функции *simplify* библиотеки SymPy. Возникает вопрос о том, как можно описать класс выражений, для которых функция *simplify* всегда выдает результат, т.е. выделить область полноты разработанной системы. На этот вопрос нельзя ответить, поскольку *simplify* – это эвристический метод, который постоянно совершенствуется. Так, если, например, выражение действительно тождественно 0, но не упрощается до него, то это обычно происходит в силу одной из следующих причин: 1) требуемое упрощение очень сложное, 2) упрощение не применимо для некоторых значений переменных.

В SymPy используется предположение о том, что все символы по умолчанию являются комплексными числами, и в процессе работы *simplify* не используются упрощения, которые не являются применимыми для всех комплексных чисел. Например, $\sqrt{x^2} == x$ верно только тогда, когда x – положительное число. Разработчики SymPy отмечают, что второй вышеобозначенной причины можно избежать путем дополнительных настроек функций упрощения выражений, вызываемых в ходе работы функции *simplify*.

Перечисленные особенности работы функции *simplify* позволяют надеяться на то, что решения не слишком сложных задач будут успешно проверяться предложенным выше способом. Данный способ был успешно апробирован при проверке задач из курса по линейной алгебре для студентов психологического факультета МГУ. При расширении обучающей системы на другие предметные области может потребоваться расширение и доработка как эвристики сравнения формулы шага студента с «обязательными» формулами, так и самой библиотеки SymPy.

Логические же методы интеллектуализации компьютерных систем в ИОС «Волга» при всей их абстрактности обладают рядом достоинств для их выделения в качестве базовых методов. Формализм позитивно-образованных формул, лежащий в основе логических методов, используемых в рамках проекта ИОС «Волга», характеризуется рядом

преимуществ указанных выше, и, в частности, компактностью представления и регулярностью структуры используемых формул, неразрушением исходной эвристической структуры знаний, что облегчает совместимость логики с эвристиками конкретного применения, большей эффективностью техники вывода в силу как меньшей размерности комбинаторного пространства поиска выводов, так и возможности распараллеливания вычислений в процессе вывода. На основе этих логических средств разрабатываются автоматические решатели задач для определенных предметов естественно-научного цикла (геометрии, математической статистики, качественной теории дифференциальных уравнений и др.). Такие решатели позволяют находить наиболее короткие решения задач или такие, которые не используют тех указанных априори разделов предмета, которые слабо освоены обучаемым.

Поскольку задачи планирования взаимодействия обучающей системы с обучаемым допускают ту или иную степень логической формализации, то применение логического (конструктивного) поиска выводов позволяет обеспечить интеллектуальность интерактивного взаимодействия системы с обучаемым: в случае когда обстоятельства, описывающие текущую ситуацию, имеют вид произвольных по-формул, а цель является квазихорновской формулой, любой вывод позволяет выстраивать цепочку действий как управление процессом обучения. Если та или иная рассматриваемая подцель недостижима ввиду неполноты средств ее достижения, то предложенный механизм (ω, α) -процесса обеспечивает дооснащение обстановки и планирование достижения подцели.

Описанные языки и исчисление приведены в пропозициональном варианте и допускают переформулировку на первопорядковый язык [13]. Вместе с тем вопросы работы с равенствами, поддержки индукции и некоторые другие логические вычисления пока остались неисследованными.

Литература

1. Vassilyev S. (et al.) Adaptive Approach to Developing Advanced Distributed E-learning Management System for Manufacturing / S.N.Vassilyev, G.L. Degtyarev, V.V.Kozlov, N.N.Malivanov, S.R.Sabitov, R.A.Sabitov, R.D.Sirazetdinov // Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing. (FR-C86). Moscow. 2009. pp. 2198-2203.

2. Makatchev M. (et al.) Abductive Theorem Proving for Analyzing Student Explanations to Guide FeedBack in Intelligent Tutoring Systems / M. Makatchev, P. W. Jordan, K.VanLehn // Journal of Automated Reasoning. 2004. Vol 32, № 3. pp 187-126.

3. Koedinger K. R. Cognitive tutors: Technology bringing learning sciences to the classroom / K.R. Koedinger, A.T.Corbett // The Cambridge handbook of the learning sciences. NY: Cambridge University Press. 2006. pp 137.

4. Sympy Development Team. SymPy: Python library for symbolic mathematics [Электронный ресурс] // URL: http://www.sympy.org (Дата обращения: 06.10.2012)

5. VanLehn K. (et al.) The Andes Physics Tutoring System: Lessons Learned / K. VanLehn, C. Lynch, K. Schulze, J. A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, M.

Wintersgill // International Journal of Artificial Intelligence in Education. 2005. Vol 15, № 3. pp 147-204.

6. J.A. Shapiro. An Algebra Subsystem for Diagnosing Students' Input in a Physics Tutoring System / Shapiro J.A. // International Journal of Artificial Intelligence in Education (IJAIED), 2005. № 15. pp. 205-228

7. Melis E. ActiveMath: An Intelligent Tutoring System for Mathematics / E.Melis, J.Siekmann // Artificial Intelligence and Soft Computing – ICAISC 2004. Springer Berlin Heidelberg, 2004. Vol. 3070. pp. 91-101.

8. Bradford R. (et al.) A Comparison of Equality in Computer Algebra and Correctness in Mathematical Pedagogy / R. Bradford, J.H. Davenport, C. Sangwin // International Journal for Technology in Mathematics Education, 2009. Vol.16, № 1.

9. Рассел С. Искусственный интеллект. Современный подход / С.Рассел, П.Норвиг // 2-е изд.: Пер. с англ. М.: Издательский дом «Вильямс», 2006. 1408 с.: ил. 19

10. Robinson J.A. On Automatic Deduction // Rice University Studies, 50, 1964, p. 69-89.

11. Robinson J.A. A Machine-oriented Logic Based on the Resolution Principle // J. ACM, 12, 1965, p. 23-41.

12. Robinson J.A. The Generalized Resolution Principle. In: Machine Intelligence / D. Michie (ed.), 1968, 3, NY, American Elsevier, pp. 77-94.

13. Васильев С.Н. (и др.) Интеллектное управление динамическими системами / С.Н. Васильев, А.К. Жерлов, Е.А. Федосов, Б.Е. Федунов // М.: Физико-математическая литература, 2000. 352 с.

14. Суконнова А.А. Автоматизация решения некоторого класса вычислительных задач / А.А.Суконнова // Материалы X Международной научно-технической конференции. Таганрог: Изд-во ТТИ ЮФУ. 2009. 294 с.

15. Суконнова А.А. Алгоритмизация тестирования компьютерных программ, создаваемых студентом / А.А.Суконнова // Современные технологии и материалы – ключевое звено в возрождении отечественного авиастроения: Сборник докладов международной научно-практической конференции. Т. IV. Казань: Изд-во «Вертолет». 2010. 480 с.

16. Vassilyev S.N. Machine Synthesis of Mathematical Theorems. - J. of Logic Programming, 1990, v. 9, N 2 & 3, pp. 235-266.

17. McCarthy J. Parameterizing Models of Propositional Calculus Formulas [Электронный ресурс] / J. McCarthy (http://www-formal.stanford .edu/jmc/parameterize/parameterize.html).

18. Flach P.A. Towards the Inductive Logic Programming. - Proc. BENELEARN-91, Depart. of Social Science Informatics, Univ. of Amsterdam, 1991, pp. 88-96.

19. Kowalski R.A. Computational Logic in an Object-Oriented World / In: Reasoning, Action and Interaction in AI Theories and Systems – Festschrift in Honor of Luigia Carlucci Aiello (eds. O. Stock, M. Schaerf), Springer Verlag, LNAI, 2006.

20. Васильев С.Н. Метод синтеза условий выводимости хорновских и некоторых других формул / С. Н. Васильев // Сиб. мат. журн. 1997. Т. 38. № 5. С.1034-1046.