О самообучении интеллектуальной системы

А. С. Подколзин

Работа посвящена изложению результатов компьютерного моделирования логических процессов в логической системе «Искра». Исследовались вопросы, связанные с обучением компьютерных решателей задач и автоматическим синтезом приемов.

Ключевые слова: интеллектуальная система, самообучение, логическая система, компьютерный решатель задач, автоматический синтез приемов.

Введение

Легенда о самообучающемся, саморазвивающемся искусственном интеллекте зародилась в недрах научной фантастики очень давно, и с тех пор стала в произведениях этого жанра чем-то совсем обыденным. Мы редко задумываемся о том, что по отношению даже к естественному интеллекту понятия «самообучение» и «саморазвитие» следует использовать с большой осторожностью. Человек обучается, впитывая в себя знания и умения, накопленные человечеством на протяжении многовекового развития. Этот искусственно созданный мир знаний и является подлинным источником человеческого интеллекта. Нейросистема человеческого мозга становится естественным интеллектом, лишь адаптируясь к данному искусственному миру. Фактически, мы имеем единственную интеллектуальную систему, подлинно саморазвивающуюся и самообучающуюся — это интеллект всей человеческой цивилизации. Поэтому, говоря о «самообучающемся искусственном интеллекте», поначалу следует ограничиться установкой на создание компьютерной системы, способной, как и человек, «самообучаться» по книгам и другим носителям знаний.

Если же ориентироваться на создание искусственного интеллекта, саморазвивающегося в абсолютном смысле, то следует ясно отдавать себе отчет, что оъектом изучения здесь является не архитектура нейросистемы — естественной либо искусственной, а логические процессы, обеспечивающие развитие упомянутого выше «мира знаний». Даже создание идеальной искусственной нейросистемы не отменит необходимости развития науки о логических процессах, как не отменила потребности в развитии теоретической механики способность естественных нейросистем управлять динамическими процессами. К сожалению, на сегодняшний день науки о логических процессах мы не имеем. Математическая логика, предпринимая попытки найти какие-то эффективные общие принципы автоматического решения задач, столкнулась с проблемой экспоненциальной трудоемкости перебора, и вынуждена была отступить. В этой ситуации единственным подходом к изучению логических процессов оказывается компьютерное моделирование реальных процессов рассуждений. Компьютер становится своего рода микроскопом, позволяющим нарисовать детальную картину мира логических процессов во всем ее разнообразии. Уже первые попытки такого рода ясно показывают, насколько велико данное разнообразие. Оно выглядит не меньшим, чем разнообразие физических или химических явлений, и развитие науки о логических процессах, а вместе с ней и создание искусственного интеллекта, затянется на весьма длительный период.

Настоящая статья посвящена исследованию логических процессов, предпринятому с помощью компьютерной логической системы «Искра». Чтобы промоделировать в системе решение задачи человеком, оно разбивалось на атомарные шаги, и для каждого шага создавалась несложная программа, способная выполнять его в аналогичных ситуациях. Такие программы получили название приемов. Были проработаны многие предметные области, в первую очередь математические, и накоплена база приемов, имеющая более 40000 элементов. Фактически, она представляет собой огромную коллекцию «фотоснимков» с траекторий решения задач, взятых из задачников. В ряде областей база приемов оказалась способной функционировать как автоматический решатель задач, неплохо имитирующий действия человека. Но, что гораздо более важно, она стала исходным сырьем для анализа того, как базисные теоремы предметной области преобразуются в приемы, иными словами, для анализа процесса самообучения «по книгам». Были предприняты стандартизация и оптимизация при-

емов, создана их классификация. Почти для каждого приема указан источник — некоторая базисная теорема предметной области. Изучена последовательность действий при переходе от теоремы к приему, и предложены предварительные версии алгоритмов, воспроизводящих такие действия. Среди данных алгоритмов имеются процедуры адаптации приемов к обучающим задачам. Результатом явился некоторый прототип «генератора приемов», позволяющий в простых случаях сразу получать вполне осмысленные приемы, в более сложных — приходить к ним после автоматической доводки по задачнику. Совсем сложные случаи дают хороший материал для продолжения исследований. Данный генератор приемов можно уподобить пирамиде, в основании которой лежит база приемов, вершиной служит множество базисных теорем, а промежуточные слои соответствуют этапам синтеза приема.

В целом, проделанная работа подтвердила эффективность использования компьютерной логической системы как «микроскопа» для изучения логических процессов и наметила пути создания интеллектуальной системы, способной быстро обучаться по традиционным источникам научно-технической информации.

Архитектура решателя задач

Задачи формулируются в решателе на специальном логическом языке, пополняемом по мере обучения. Этот же язык используется для формулировки теорем, приемов и т. п. Он может рассматриваться как вариант языка исчисления предикатов, ориентированный на сближение с реальной математической практикой. Вместо традиционных для математической логики терминов «формула» и «терм» используются термины «утверждение» и «выражение». Все эти конструкции имеют скобочный вид $f(a_1 \dots a_n)$. Обычным образом вводятся связанные переменные, которые возникают лишь в связи с кванторами «существует», «длялюбого», «существует единственный», а также описателями «класс» и «отображение», задающими множества и функции. Каждое новое понятие регистрируется в системе и сопровождается справочной информацией, уточняющей его синтаксис и семантику. Для удобства обучения создан формульный редактор, конвертирующий скобочную запись в привычную математическую формулу и обратно.

Введена несложная общелогическая классификация задач, и зафиксированы структуры данных для представления таких задач. Задачи делятся на следующие четыре типа:

1) Задача на доказательство. Такая задача имеет список посылок $A_1, \ldots, A_n$ и условие A_0. Если удается установить, что утверждение A_0 является следствием утверждений $A_1, \ldots, A_n$, то задача имеет ответ «истина». В противном случае решатель выдает символ «отказ».
2) Задача на преобразование. Как и выше, имеются список посылок $A_1, \ldots, A_n$ и условие A_0, которое уже является не утверждением, а выражением. Кроме того, задан список целей, уточняющих, каким именно образом должно быть преобразовано выражение A_0. Одни цели накладывают строгие органичения на вид ответа, другие — указывают тенденции, учитываемые решателем (например, «упростить»). Преобразования проводятся в предположении истинности посылок $A_1, \ldots, A_n$. Ответом задачи служит результат преобразований выражения A_0. Он выдается по достижении требуемого вида ответа, а в допустимых случаях — по исчерпании средств решателя.
3) Задача на описание. Имеются список посылок $A_1, \ldots, A_n$, а также список условий $B_1, \ldots, B_m$. Задан список целей C. Требуется преобразовать условия в предположении истинности посылок, учитывая цели задачи; например, разрешить условия относительно неизвестных, заданных в списке C. В этом списке может содержаться указание, нужен ли полный ответ или достаточен частичный, нужно ли упрощать его относительно известных параметров, и т. п. Обычно задача на описание сопровождается вспомогательной задачей на исследование (см. ниже). Эта вспомогательная задача является накопителем общих следствий условий и посылок, вывод которых иногда позволяет получит явные выражения для неизвестных.
4) Задача на исследование. Имеется список посылок $A_1, \ldots, A_n$, и задан список целей C. Решение задачи заключается в выводе следствий из посылок с учетом целевой установки. Оно обрывается при получении результатов, интересных для внешней задачи (усмотрение противоречия, нахождение равнства для неизвестной и т. п.), либо по исчерпании лимита трудоемкости. Найденные важные следствия передаются во внешнюю задачу.

Кроме перечисленных логических компонент задачи, введены технические компоненты, используемые решателем. Во-первых, все условия и посылки задач сопровождаются «весами» — целыми числами от 0 до 20, регулирующими концентрацию внимания при просмотре задачи. Во-вторых, каждый терм задачи (условие либо посылка) сопровождается списком технических пометок — комментариев к этому терму. Такими же списками сопровождаются весь набор посылок и вся задача. Технические пометки позволяют сопровождать логические структуры данных сетевыми конструкциями различной природы. Они же регулируют взаимодействие между приемами.

Преобразования задачи, а также ввод и рассмотрение различных вспомогательных задач в процессе решения обеспечиваются приемами — процедурами, обобщающими атомарные шаги рассуждений человека. Таких приемов в процессе обучения накапливаются многие тысячи, и возникает проблема быстрого поиска нужного приема. В принципе, здесь возможны два подхода.

Первый из них состоит в сканировании древовидного каталога базы приемов, позволяющего отсекать ветви, заведомо не содержащие нужного приема. Если после отсечения осталось более одной ветви рассматриваемой вершины, то далее поиск предпринимается независимо в каждой из оставшихся ветвей. После того, как нужный прием найден, он реализуется, и далее цикл поиска повторяется заново. Недостатком этого подхода оказалось то, что в реальном многообразии приемов не удалось найти сколь-нибудь хороших принципов отсечения. Кроме того, по мере обучения и увеличения числа приемов время поиска будет увеличиваться. Это, безусловно, создаст (а при плохом отсечении — достаточно скоро) сильные ограничения на объем базы приемов, при котором возможна эффективная работа решателя.

Второй подход основан на цикле сканирования задачи. Большинство приемов, используемых при решении задач, допускает явное указание такого понятия, что для возможности применения приема необходимо появление этого понятия в задаче. Это позволяет организовать базу приемов по принципу энциклопедии: за каждым понятием логического языка закрепляется сравнительно небольшая группа его приемов. Как показывает опыт, исключение составляет лишь крайне незначительная группа приемов, для которых не выделяется какого-либо естественного «инициализирующего» понятия. Очередной шаг решения задачи при втором подходе состоит в последовательном, по-

нятие за понятием, просмотре всего текста задачи, с обращением для каждого текущего понятия к поиску внутри группы его приемов. Эта группа невелика, и во многих случаях ее можно организовать по древовидному принципу, получая таким образом дополнительное ускорение поиска. Преимуществом данного подхода является то, что при обучении системы новым разделам мы практически не уменьшаем скорости ее работы на ранее проработанных разделах. Так получается потому, что «новые» понятия не встречаются в «старых» задачах, и наличие даже очень большого числа «новых» приемов никак не сказывается на времени поиска «старых», связанных с другими понятиями. Фактически, здесь снимаются сколь-нибудь сильные ограничения на рост числа приемов и появляется принципиальная возможность создавать гигантские логические системы, имеющие фундаментальное разностороннее образование. Именно этот подход и был реализован в логической системе «Искра».

Таким образом, рабочий цикл решателя состоит в сканировании текста задачи — последовательном просмотре встречающихся в ней понятий и обращении для текущего понятия к его программе, объединяющей в себе все закрепленные за понятием приемы. Эта процедура представляет собой что-то типа внутреннего «логического зрения», обеспечивающего контроль за развитием событий. Как и человек, в процессе решения система предпринимает мысленное «рассмотрение» объектов задачи и связей между ними для выработки плана ближайших действий.

Чтобы выбрать из всех возможных действий наилучшее, необходимо каким-то образом сравнивать между собой результаты применения различных приемов в текущем контексте. Для этого естественно ввести числовые оценки приоритетности, выбирая каждый раз, например, прием с наименьшей такой оценкой. Вообще говоря, нерационально при сканировании задачи находить все возможные варианты применения приемов и лишь по окончании сканирования отбирать лучший — это может потребовать рассмотрения слишком большого числа вариантов, причем по своей априорной ценности анализируемые варианты тоже будут сильно различаться. К меньшим вычислительным затратам можно прийти, если разбить все приемы на группы или уровни, объединяя более-менее равноценные, и поиск нужного приема вести с последовательным увеличением номера уровня, обрывая его, как только найден применимый в текущей ситуации прием. Фактически, здесь оценкой приоритетности приема становит-

ся номер уровня. Так как обработка каждого уровня приемов требует своего цикла сканирования задачи, то число уровней целесообразно сделать не очень большим; в решателе это число равно 16, причем как правило срабатывание приема происходит на первых 5–6 уровнях. В действительности, один и тот же прием может в различных ситуациях относиться к различным уровням — номер уровня определяется, в зависимости от контекста, самой программой приема. Если этот уровень не соответствует тому, для которого выполняется текущее сканирование задачи, то дальнейшие действия по рассмотрению приема обрываются, и система переходит к другим приемам.

Важную роль при сканировании задачи играет управление переключением внимания. Вообще говоря, различные элементы описания задачи неравноценны при поиске очередного приема. Одни из них находятся в задаче давно, другие — только что занесены либо изменены. Вероятность обнаружить при рассмотрении первых ситуацию, требующую немедленных действий, обычно значительно ниже, чем для вторых. Чтобы учесть такую статистическую неоднородность элементов задачи и уменьшить среднее время поиска приема, элементы задачи (посылки и условия) снабжаются весами — целыми неотрицательными числами, указывающими номер того уровня сканирования, до которого ранее доходило рассмотрение элемента. Как только элемент изменяется, его вес уменьшается до 0; веса новых элементов также полагаются равными 0. При сканировании, связанном с приемами i-го уровня, игнорируются все посылки и условия, веса которых больше i — они образуют «теневую» часть задачи. По мере прохождения i-го уровня веса тех элементов задачи, для которых не произошло срабатывание приема, автоматически увеличиваются до $i + 1$. Как только срабатывает какой-либо прием и возникают элементы задачи с весом 0, сканирование задачи возобновляется начиная с нулевого уровня. При этом в «зону внимания» попадут только те элементы задачи, которые имеют вес 0, то есть только что были изменены либо введены. Эта простая автоматика уже обеспечивает в большинстве случаев разумную стратегию переключения внимания и значительно увеличивает быстродействие системы. В особенности это ощутимо для задач геометрического типа, имеющих сотни посылок и условий. Конечно, необходим ряд дополнительных средств, обеспечивающих переключение внимания в специальных случаях. Некоторые из этих средств включены в «общую автоматику» решателя; другие — реализованы непосредственно в приемах, которые могут избирательно

изменять веса различных элементов задачи и таким образом управлять зоной внимания.

В самых общих чертах, процесс обучения компьютерного решателя можно представить происходящим по следующей схеме. Рассматривается некоторая обучающая последовательность задач. Для очередной задачи анализируется траектория ее решения. Она разбивается на простейшие шаги, и формулируются эвристические объяснения целесообразности текущего шага в текущем контексте. На основе таких объяснений программируется прием — программа, которая в аналогичных ситуациях будет выполнять аналогичные действия.

Приемы накапливаются в компьютерной модели, так что в новой задаче некоторые из них могут сработать и предложить свою версию развития решения. Каждое такое срабатывание анализируется на предмет целесообразности, и в решающие правила приема вводятся необходимые коррективы. Эта процедура повторяется для многих сотен задач предметной области — до тех пор, пока сложившаяся система приемов не обнаружит достаточно устойчивого и эффективного поведения. В целом, процесс напоминает обучение нейросистемы методом «поощрений и наказаний»; отличие состоит лишь в более сложной процедуре коррекции, которая не сводится к изменению каких-либо числовых параметров, как у нейрокомпьютера, а требует привлечения логики для очередного перепроектирования приемов.

Обучение логической системы «Искра» было предпринято для таких предметных областей, как алгебра множеств и комбинаторика, элементарная алгебра, элементарная геометрия, математический анализ, аналитическая геометрия, дифференциальные уравнения, комплексный анализ, теория вероятностей. Проработаны многие разделы линейной алгебры, общей алгебры, дискретной математики, интегральных уравнений, элементарных физики и химии. Рассматривались нематематические области: шахматы, понимание естественного языка, анализ рисунков. Сразу заметим, что главным образом изучались стандартные задачи вычислительного характера, хотя и для многих теоретических задач тоже создавались приемы, обеспечивающие их решение. В некоторых предметных областях (например, элементарные алгебра и геометрия) удалось добиться достаточно стабильного решения стандартных задач, в других — лишь предложены объяснения отдельных траекторий решения.

При сравнении с обычными системами компьютерной математики становятся очевидны преимущества предлагаемого подхода в тех слу-

чаях, когда логика начинает играть существенную роль. По существу, логическую систему «Искра» можно рассматривать как экспериментальную систему компьютерной математики нового типа, допускающую постепенное перерастание в интеллектуальную математическую систему.

Общее число приемов сейчас превышает 42000, число проработанных задач — свыше 10000. При этом, даже без архивации, система занимет всего около 120 Мб. Это означает, что стандартные размеры внешнего твердого диска в 1Тб позволяют увеличить объем решателя в 8000 раз, без существенного замедления его в тех областях, где обучение уже завершено.

Алгоритмический язык ЛОС

Следующий важный вопрос, возникающий после выбора механизма сканирования задачи как диспетчера, организующего работу базы приемов, — это вопрос о языке для записи приемов. В приеме выделяются такие основные части, как описание ситуации, в которой логически допустимы реализуемые им действия; описание ситуации, в которой эти действия целесообразны, и описание процедуры, собственно реализующей действия приема.

Первые две части предполагают использование некоторого логического языка, на котором формулируются соответствующие условия. Эти условия относятся не к объектам предметной области, рассматриваемым в задаче (числам, точкам, векторам и т. п.), а к тем структурам данных, с помощью которых задача представлена в системе: к термам логического языка, вхождениям в эти термы, к спискам термов, к различного рода техническим пометкам и конструкциям, вводимым решателем для управления процессом. Чтобы формулировать такие условия, необходимо обеспечить достаточно богатый набор понятий логического языка, ориентированных на используемые в решателе структуры данных, и прежде всего — на задачи и их элементы.

Именно этот набор и составил основную часть нового языка ЛОС (Логический Описатель Ситуаций) для записи приемов; чтобы задавать те действия, которые должны быть выполнены приемом в уже «опознанной» им ситуации, оказалось достаточно присоединить к ло-

гической части языка сравнительно небольшое количество кодирующих типовые преобразования конструкций.

После того, как на логическом языке сформулирована ситуация, в которой применение приема возможно и целесообразно, необходимо обеспечить некоторый алгоритм, реализующий в задаче поиск этой ситуации. В описании ситуации должны фигурировать объекты, идентифицированные еще до начала поиска — сама задача; выделенное в ней при сканировании вхождение понятия; текущий уровень сканирования, и т. п. Описание представляет собой последовательность утверждений $P_1, \ldots, P_n$. Некоторые P_i определяют условия на ранее идентифицированные объекты. Так как все эти объекты относятся к текущим структурам данных решателя, истинность условий проверяется непосредственно и не требует какого-либо логического вывода. Для проверки этой истинности в интерпретаторе языка имеются специальные подпрограммы. Другие P_i вводят в рассмотрение новые объекты, связанные заданным образом с уже введенными до этого объектами. Если такие объекты определяются неоднозначно, то при поиске ситуации необходимо перечислять все возможные варианты. При программировании на обычном алгоритмическом языке здесь нужно было бы использовать операторы цикла, причем вложенность циклов была бы равна числу неоднозначно определенных переходов к «новым» объектам в цепочке $P_1, \ldots, P_n$. Чтобы избежать таких громоздких конструкций, в языке ЛОС выделен ряд специальных отношений между объектами, для которых перечисление реализуется автоматически. При обработке «перечисляющего» утверждения P_i сначала рассматривается первая версия выбора новых объектов, и предпринимается попытка для выбранных объектов продвинуться вправо по цепочке $P_1, \ldots, P_n$. Если в некоторый момент дальнейшее продвижение, ввиду ложности условия, становится невозможным, то происходит откат к последнему «перечисляющему» утверждению и определение очередной версии выбора объектов. Этот принцип «перечисления по умолчанию» позволяет в качестве исполняемого текста программы ЛОСа использовать само логическое описание искомой ситуации и таким образом существенно упростить понимание текста программы.

Заметим, что из совсем других соображений принцип реализации операторов с перечислением значений их выходных переменных возник в известном алгоритмическом языке ПРОЛОГ. В отличие от ЛОСа, ПРОЛОГ изначально ориентирован на работу с объектами

предметного, а не структурного уровня. Поэтому центральное место в ПРОЛОГе занимает логический вывод, основанный на процедуре унификации. Этот логический вывод становится совершенно излишним при работе с элементами структуры данных, допускающими непосредственную проверку условий, и какого-либо аналога его в ЛОСе не предусмотрено. С другой строны, в ПРОЛОГе отсутствует целый ряд важных возможностей, предусмотренных в ЛОСе для эффективной работы со сложными логическими описаниями образов «структурного» уровня и для использования режима сканирования задачи. ЛОС ориентирован на создание автоматики, управляющей логическими процессами, в то время как при разработке ПРОЛОГа эта автоматика осталась почти «за кадром».

В ЛОСе выделяются следующие основные типы объектов, обрабатываемые программой:

1) Символы переменных и логические символы. К последней категории относятся логические связки, кванторы, константы, символы отношений и операций и т. п.
2) Термы, построенные из переменных и логических символов. Заголовок терма — произвольный логический символ, число его операндов — любое.
3) Наборы $(a_1, \ldots, a_n)$ ранее определенных допустимых объектов.
4) Вхождения символов в термы и разрядов в наборы. Фактически, вхождение — это адрес позиции в терме либо наборе.

Программа решателя организована по энциклопедическому принципу и распадается на программы отдельных логических символов. Программа логического символа φ имеет древовидную структуру и состоит из совокупности пронумерованных натуральными числами текстов, называемых фрагментами этой программы. Каждый фрагмент программы логического символа φ представляет собой последовательность $P_1, \ldots, P_n$ операторов ЛОСа. К моменту реализации оператора P_i часть значений программных переменных уже определена, а часть — еще не определена. Обычно номера определенных переменных образуют начальный отрезок натурального ряда. Оператор либо обеспечивает проверку истинности некоторого условия, либо определяет значения своих (пока не определенных) выходных переменных. Иногда определение значений выходных переменных реализуется в режиме перечисления. При откате к перечисляющему оператору все

переменные, которые были определены после его первой реализации, снова оказываются не определенными. Наконец, оператор может изменять текущие структуры данных.

Связь между фрагментами программы организуются с помощью операторов «иначе N» и «ветвь N». Первый используется обычным образом: если условие P ложно, то после оператора «P иначе N» будет реализован переход к фрагменту с номером N. Оператор «ветвь N» является вырожденным перечисляющим оператором. При первом попадании на него никаких действий не предпринимается, и выполняется переход к следующему оператору. Если при продвижении по цепочке операторов вправо встретилось ложное условие, то откат к оператору «ветвь N» вызывает переход к фрагменту с номером N. Данный оператор удобен для склейки начальных частей программ нескольких независимых приемов.

Обращение к программе символа φ бывает следующих типов:

1) Обращение из сканирования задачи. Тогда программе передается информация о координатах вхождения текущего символа в задачу.
2) Обращение при реализации программы вспомогательного оператора либо операторного выражения, написанной на ЛОСе. Тогда текущий логический символ — заголовок программируемого программного терма.
3) Обращение при обработке справочного запроса. Такие обращения оказываются нужны, чтобы разбросать характеризующую логические символы информацию заданного типа по программам конкретных символов. Например, информацию, уточняющую тип значений операции, либо число ее операндов, и т. п.

Для уточнения типа обращения имеются соответствующие операторы.

В качестве простого примера рассмотрим программу приема, усматривающего в задаче утверждение вида «или($A_1 \dots P \dots \neg(P) \dots A_n$)» и заменяющего его на константу «истина». Эта программа имеет вид:

«операнд(х2 х6) символ(х6 не) операнд(х2 х7)
равныетермы(первыйоперанд(х6)х7)
заменавхождения(х2 х3 х4 х1 истина пустоеслово)».

Программа относится к символу «или», причем при обращении к ней значением переменной x2 является вхождение данного символа в текущий терм задачи. Оператор «операнд(x2 x6)» перечисляет всевозможные операнды дизъюнкции, присваивая их вхождения переменной x6. Оператор «символ(x6 не)» отфильтровывает лишь те операнды, заголовком которых служит символ «не». Таким образом, x6 будет идентифицироваться с вхождением операнда «$\neg(P)$». Оператор «операнд(x2 x7)» выполняет повторный просмотр операндов дизъюнкции. Наконец, оператор «равныетермы(первыйоперанд(x6)x7)» отфильтровывает те x7, которые суть вхождения утверждения P. Завершающий оператор реализует замену дизъюнкции на константу «истина». В этом примере оказались «спрятаны» два оператора цикла, соответствующих выделению первого и второго операндов. Если расписать их явным образом, то программа оказалась бы более длинной и менее понятной.

Еще одна интересная особенность ЛОСа — использование кванторов в качестве операторов алгоритмического языка. Например, оператор «длялюбого(x2 если входит(x2 x1) то заголовок(x2 или))» проверяет, что каждый элемент набора x1 имеет своим заголовком символ «или». Оператор «существует(x2 и(подчинено(x1 x2) символ(x2 синус)))» проверяет, что текущее вхождение x1 расположено внутри терма, имеющего своим заголовком символ «синус». В обоих случаях мы имеет «спрятанный» цикл. Если реализовать его традиционным образом, то понимание смысла оператора затруднится.

При программировании со «спрятанными» циклами длина программ существенно уменьшается, и становится сравнительно простым непосредственное их чтение. Режим перечисления сильно упрощает также программирование на ЛОСе перечисляющих процедур рекурсивного характера. Чтобы еще больше упростить работу с программами ЛОСа, предусмотрен ряд средств интерфейса, компенсирующих прочие недостатки. Впрочем, в действительности ЛОС оказывается лишь языком промежуточного уровня, редко используемым при обучении решателя. Обычно программы на ЛОСе пишет компилятор.

ЛОС — программы выполняются интерпретатором, реализованным на СИ. Это обеспечивает возможность автоматического перепрограммирования системой самой себя в процессе решения задач. Все интерфейсы системы тоже реализованы на ЛОСе.

Язык для записи приемов ГЕНОЛО́Г

Как правило, значительную часть программы приема на ЛОСе составляет описание вида тех утверждений или выражений, которые могут быть идентифицированы с фрагментами заданной теоремы предметной области. На практике редко бывает так, чтобы идентифицируемые логические конструкции в точности совпадали с теми, которые имеются в теореме. Например, при идентификации квадратного трехчлена $ax^2 + bx + c$ коэффициент a может оказаться равным единице, и тогда в задаче вместо произведения ax^2 будет записана степень; сама эта степень может не иметь в точности вида квадрата, а лишь иметь четный коэффициент показателя степени, и т. д. Чтобы, несмотря на эти различия, идентифицировать теорему, приходится фактически в программе приема описывать не ее вид, а вид некоторого класса теорем, получающихся из нее различными вариациями. В результате текст программы оказывается, во-первых, весьма громоздким, и, во-вторых, достаточно удаленным от исходной записи теоремы. Еще большую громоздкось этому тексту придают различные вставки, связанные с решающими правилами приема. Поэтому, несмотря на все предоставляемые ЛОСом и интерфейсом его редактора возможности для быстрого чтения и понимания логических описаний ситуаций, все же в сколь-нибудь невырожденных случаях описания приемов оказываются трудночитаемыми. Заметим, что хотя ПРОЛОГ, казалось бы, и ориентирован на запись программы в виде совокупности теорем предметной области, однако он тоже не позволяет преодолеть указанную трудность. Если теоремы идентифицируются без учета возможных вариаций, то программа оказывется заведомо слабой; если начинается их учет, то и на ПРОЛОГе, для задания общего вида класса теорем, придется от предметного уровня перейти к структурному (к которому ЛОС более приспособлен), и таким образом полностью утратить наглядность исходного текста теоремы.

Чтобы восстановить утерянную при погоне за эффективностью работы программы приема наглядность и компактность записи, был создан новый язык, уровень которого существенно выше, чем у ЛОСа или ПРОЛОГа. Прием на этом языке задается как теорема предметной области, снабженная некоторой «алгоритмизирующей» разметкой. На основе разметки компилятор осуществляет необходимое обобщение вида теоремы, формулирует описание на ЛОСе ситуаций,

в которых она должна быть применена, и добавляет к нему операторы ЛОСа, реализующие применение. Фактически этот язык имеет два независимых логических уровня: предметный уровень, на котором представлена теорема, и структурный уровень, на котором формулируются условия целесообразности применения теоремы. Оба эти уровня предполагают использование полноценного логического языка, однако логические записи структурного уровня, в отличие от записей уровня предметного, не участвуют в каком-либо логическом выводе — они используются только для проверки условий при принятии решения. Роль «алгоритмизирующей» разметки приема весьма разнообразна — это и указание способа применения теоремы, и определение необходимых правил ее обобщения, и указание условий целесообразности применения, и уточнение алгоритмических средств, используемых для обработки посылок теоремы, и указание на «технические» сообщения, передаваемые при срабатывании данного приема другим приемам, и указатели переключения внимания, и многое другое. Эта разметка представляет собой что-то вроде генотипа приема, элементы которого допускают независимое варьирование в достаточно широких пределах и по которому компилятор создает фактически реализуемую программу приема на ЛОСе. Такая аналогия позволила назвать новый язык ГЕНОЛО́Гом (ГЕНетический язык ЛОГического программирования).

Описание приема на ГЕНОЛО́Ге оказалось значительно более компактным и понятным, чем на ЛОСе. На экране изображается в стандартной математической записи теорема приема, под которой размещается в нескольких окнах сопровождающая теорему информация. Во многих случаях эта информация занимает всего несколько строчек, в то время как текст создаваемой компилятором программы на ЛОСе составляет несколько полных экранов. Трудоемкость чтения приема и его коррекции, по сравнению с ЛОСом, уменьшилась на порядок.

В отличие от обычных языков программирования, ГЕНОЛО́Г не представляет собой сколь-нибудь завершенного явления. По существу, он является коллекцией типовых алгоритмических конструкций, используемых при переходе от теоремы к программе. Эта коллекция, достаточно богатая на текущий момент, продолжает (хотя все реже и реже) пополняться при рассмотрении новых предметных областей. Соответственно, при переходе к новой предметной области обычно приходится затрачивать некоторые усилия на развитие ком-

пилятора ГЕНОЛО́Га. Как показывает опыт, эти затраты несопоставимо малы в сравнении с той последующей экономией, которую дает применение ГЕНОЛО́Га при обучении решателя в данной области.

Описание приема на ГЕНОЛОГе состоит из следующих компонент:

1) Теорема предметной области, на которой основан прием. Эта теорема вводится при помощи формульного либо текстового редактора, в стандартной математической либо скобочной записи. Для большей наглядности предусмотрено сопровождение теорем с геометрическими объектами чертежом; компилятор этот чертеж никак не использует.
   Заметим, что хотя ГЕНОЛОГ и приближен к логическому языку предметной области, все же он представляет собой язык программирования. Настоящие теоремы предметной области содержатся в базе теорем. В теореме приема же допустимы (как правило, небольшие) отклонения от логических стандартов предметного уровня. Эта теорема обычно преобразуется к виду, более удобному для практического использования — например, в посылках могут вводиться вспомогательные обозначения для объектов, упоминаемых в утверждении теоремы; часть посылок, которые в обычных «разумных» контекстах автоматически бывают выполнены, отбрасывается, и т. п. В особых случаях теорема приема вообще может представлять собой техническую заглушку — «псевдотеорему».
2) Заголовок приема. Он указывает на общую схему использования теоремы в приеме (например, тождественная замена слева направо; вывод следствия и т. п.)
3) Список фильтров приема. Элементы этого списка суть сформулированные с помощью несколько модифицированного логического языка условия на целесообразность применения приема в текущем контексте. Это — второй логический уровень описания приема; первым уровнем является сама теорема, также задаваемая на логическом языке. Однако, в случае теоремы логический язык используется для описания ситуаций на предметном уровне, а в случае фильтров — для описания ситуаций на структурном уровне. В этом он близок ЛОСу и может рассматриваться (с рядом существенных оговорок) как «ЛОС в теоремных переменных». В действительности многие операторы ЛОСа

имеют своих одноименных двойников в языке фильтров приемов, хотя эти двойники используются уже по-другому — у них часто отбрасываются однозначно восстанавливаемые из контекста описания приема операнды. Кроме того, в языке фильтров приема возникают конструкции, совершенно не имеющие аналогов в ЛОСе.

4) Список указателей, адресуемых компилятору (далее называем их просто указателями приема). Элементы этого списка уточняют способ формирования ЛОС-программы приема компилятором ГЕНОЛОГа. По существу, это основная часть «генотипа» приема. Здесь уточняются: средства, используемые для усмотрения в задаче «замаскированных» возможностей применения теоремы; способы обработки отдельных посылок теоремы; технические комментарии, вводимые в задачу либо удаляемые при применении приема; логика переключения внимания после применения приема, и др.

5) Список нормализаторов. При формировании результирующих (включаемых в структуру данных задачи) либо вспомогательных (рассматриваемых в процессе работы приема) термов могут применяться процедуры для их упрощения либо каких-либо иных специальных преобразований. Такие процедуры задаются последовательным применением нормализаторов — либо специальных пакетов приемов для тождественных и эквивалентных преобразований, либо вспомогательных задач. Использование небольшого пакета приемов вместо общей базы приемов, активизируемой при сканировании задачи, во многих случаях на порядок ускоряет вычисления. Эти «локальные» приемы отличаются от приемов сканирования задачи — они применяются не в контексте задачи, а в некотором другом специальном контексте, состоящем из преобразуемого терма, списка посылок, в предположении истинности которых ведутся преобразования, и списка технических комментариев. Задаются они, как и приемы сканирования задач, на ГЕНОЛОГе.
Специальный интерфейс позволяет достаточно быстро вводить обращения к нормализаторам путем указания в теореме (либо в сопровождающих теорему термах описания приема) подлежащих нормализации точек. В результате использования нормализаторов и вспомогательных процедур обработки посылок тео-

ремы, срабатывание приема оказывается иногда чрезвычайно сложным вычислительным процессом, основанным не на единственной теореме предметной области, а на десятках и даже сотнях (с учетом прменяемой рекурсии) различных теорем.

6) Шаблоны сопровождения. Здесь хранятся текст-формульные шаблоны, используемые при создании текстов, объясняющих срабатывания приемов. Способ их применения регламентируется специальными указателями приема.

Для примера описания приема по данной схеме, рассмотрим простейший прием решения уравнения $ax = b$.
Теорема приема имеет в этом случае вид:

$$\forall abx(\text{число}(a)\ \&\ \text{число}(b)\ \&\ \text{число}(x) \Rightarrow$$
$$\Rightarrow (ax = b \Leftrightarrow (\neg(a = 0)\ \&\ x = b/a) \vee (a = 0\ \&\ b = 0))).$$

Заголовком приема служит логический символ «второйтерм», указывающий, что теорема будет применяться для эквивалентной замены слева направо.

Фильтры приема указывают следующий контекст, в котором выполняется замена:

1) Тип решаемой задачи — на описание;
2) Преобразуемое равенство входит в условие задачи, причем это вхождение — корневое;
3) Текущий уровень сканирования, при котором происходит срабатывание приема, равен 1;
4) Выражение b не содержит неизвестных задачи, а x — содержит.

Указания компилятору уточняют следующие подробности:

1) Переменная x идентифицируется как совокупность всех множителей рассматриваемого произведения, содержащих неизвестные задачи (это автоматически предопределяет, что переменная a будет идентифицирована с выражением без неизвестных);
2) При идентификации выражения ax возможен учет стоящего перед ним знака «минус», который относится программой приема к коэффициенту a;

3) Посылки теоремы проверяются при помощи специального пакета приемов, обеспечивающего быстрое усмотрение условий вида «число(...)».

В приеме используются следующие обращения к нормализаторам:

1) Предпринимаются обращения к вспомогательному пакету приемов для разложения на множители выражений a, b. Если такие разложения не находятся сразу же, до преобразования рассматриваемого уравнения, то впоследствии может возникнуть разбор случаев, приводящий к дублирующим попыткам разложения на множители в различных подслучаях;
2) Выполняется обращение к пакету стандартизации дробных выражений для выражения b/a. Этот пакет состоит из нескольких десятков простых продукций, выполняющих, в частности, сокращение дробей, деление и умножение дробных выражений, учет констант 0 и 1, и т. п.;
3) Для уравнений $a = 0, b = 0$ выполняются обращения к пакетам приемов, стандартизирующим уравнения данного вида (сокращение левой части на множители, не обращающиеся в 0; усмотрение тождественной ложности; преобразование равенства нулю произведения в дизъюнкцию равенств нулю сомножителей, и т. п.);
4) К заменяющей дизъюнкции применяется пакет общелогических приемов (устранение логических констант «истина» и «ложь»; стандартизация конструкций с логическими связками и кванторами).

В результате выполнения указанных преобразований заменяющий терм в данном приеме приобретает достаточно устойчивый вид, не вызывающий немедленного срабатывания цепочки простейших нормализующих приемов. Заметим, что трудоемкость одного цикла сканирования задачи обычно возрастает пропорционально суммарной длине находящихся в активной области ее термов, а всего времени решения задачи — пропорционально квадрату средней такой длины. Поэтому вынесение во вспомогательные задачи преобразований подтермов сложного выражения приводит, если эти преобразования достаточно интенсивны, к значительному ускорению решения задачи. Соответственно, стандартом программирования на ГЕНОЛОГе является как можно большее использование нормализаторов.

Приемы, применяемые при сканировании задачи, могут обращаться к вспомогательным процедурам — так называемым пакетным операторам. Эти операторы представляют собой небольшие группы приемов, записываемых на ГЕНОЛОГе и компилируемых в программы ЛОСа. В приеме пакетного оператора допускаются обращения к другим пакетным операторам и рекурсивные обращения. Использование пакетных операторов позволило существенно ускорить работу решателя. Имеются 4 основных типа пакетных операторов, аналогичных 4 основным типам задач (на доказательство, описание, преобразование и исследование):

1) Аналогом задачи на доказательство является проверочный оператор, получающий в качестве входных данных проверяемое утверждение (точнее, набор выражений, подставляемых вместо переменных в шаблон, задающий вид такого утверждения); список посылок, в предположении истинности которых проводится проверка, а также список комментариев управляющего характера. Помимо проверки истинности, проверочный оператор также выдает список посылок, фактически использованных при проверке. Пример проверочного оператора — оператор «усм-меньше», получающий в качестве входных данных: выражения a, b, для которых проверяется условие $a < b$; список посылок, в предположении истинности которых происходит проверка; набор комментариев (в частности, в нем могут содержаться ограничения на применяемые при проверке средства).

2) Аналогом задачи на описание является пакетный оператор, названный синтезатором. Он предназначен для решения задач на описание с единственным условием, имеющим строго ограниченный вид, определяемый некоторым шаблоном. Часть переменных шаблона выделена как «известные», а часть — как «неизвестные». Синтезатор определяет значения неизвестных, получая в качестве входных данных: набор выражений, подставляемых вместо «известных» переменных в шаблон; список посылок, в предположении истинности которых предпринимается подбор значений неизвестных, и набор комментариев, имеющих управляющий характер (целевая установка и т. п.) Результатом работы синтезатора служат набор искомых значений неизвестных и список фактически использованных посылок. В зависимости от своего типа, синтезатор либо находит единствен-

ный пример допустимых значений неизвестных, либо перечисляет такие примеры. Пример синтезатора — оператор «верхняяоценка», реализующий утверждения вида $a \leqslant x$ для известной переменной a и неизвестной x, который перечисляет представляющие интерес (с учетом комментариев) константные верхние оценки x выражения a.

3) Аналогом задачи на преобразование является пакетный оператор, названный нормализатором. Он получает в качестве входных данных преобразуемый терм (выражение либо утверждение), список посылок, в предположении истинности которых выполняются преобразования, и набор комментариев, имеющих управляющий характер. Прием нормализатора выполняет некоторое преобразование его «условия». Это преобразование (в зависимости от предназначения нормализатора) не обязательно должно быть тождественным (например, при получении асимптотической оценки какого-либо выражения). В отличие от проверочного оператора и синтезатора, представляющих собой операторы ЛОСа, нормализатор является операторным выражением — его значением служит возникший после цепочки преобразований терм. Пример нормализатора — операторное выражение «нормдробь», осуществляющее общую стандартизацию дробных выражений (отбрасывание знаменателя 1, деление дроби, деление на дробь, вынесение минуса из числителя либо знаменателя, и т. п.)

4) Аналогом задачи на исследование является пакетный оператор, названный анализатором. Он обрабатывает копию списка посылок текущей задачи Z, оформленную в виде вспомогательной задачи на исследование Z'. Анализатор не обращается для сканирования Z' к основной базе приемов, а использует свои собственные приемы. Так как их существенно меньше, то удается за сравнительно небольшое время получить значительное число следствий. Иэ этих следствий отбираются лишь ценные для внешней задачи Z, которые по окончании работы анализатора переносятся в ее список посылок. Обращение к анализатору обеспечивается специальными приемами, указывающими лимит времени его работы. Передача отобранных утверждений происходит по исчерпании этого лимита. При получении особо важных следствий возможен немедленный обрыв работы.

Перечисленные пакетные операторы предназначены для работы с логическими структурами данных. Однако, ГЕНОЛОГ можно использовать и для создания программ, работающих с «обычными» техническими структурами данных — числами, массивами, таблицами и т. п. Эти программы, разумеется, тоже имеют своими источниками теоремы или группы теорем, так что естественным образом возникает задача компиляции их непосредственно с теоремного уровня. Для задания нелогических вычислений на ГЕНОЛОГе используются так называемые вычислительные пакеты.

ГЕНОЛОГ является языком пограничного слоя между теоремами и алгоритмами. Так как все программы — логические либо традиционные вычислительные — возникают в конечном счете из теорем, то ГЕНОЛОГ или подобные ему языки приобретают универсальный характер. Они должны будут присутствовать в любой самопрограммирующейся системе. Поэтому запись на ГЕНОЛОГе даже давно известных алгоритмов оказывается вполне оправданной. Она позволяет проследить логические источники программ и приближает к подлинной автоматизации программирования.

Анализатор решений

Интерфейс логической системы позволяет прослеживать ход решения задачи по шагам. Прием можно сопровождать текст — формульным шаблоном, с помощью которого на экран выдаются пояснения смысла выполняемых действий. Несмотря на это, полная трассировка действий решателя на семантическом уровне обычно перегружена множеством излишних шагов. Она отображает не столько решение задачи, сколько процесс его поиска. Чтобы отбрасывать ненужные действия и оптимизировать оставшиеся, предусмотрен режим решения задачи с сохранением протокола срабатываний приемов. По окончании решения предпринимается анализ протокола: создается граф использования промежуточных результатов, и выдается сокращенная последовательность действий, в которой отброшены не использованные логические выводы. Такой режим назван анализатором решений. Он создает предпосылки для последующей оптимизации цепочки рассуждений с точки зрения ее простоты и наглядности. Анализатор решений полезен также как отладочное средство, ибо дает возможность прослеживать траекторию решения в обоих направлениях и получать различные статистические данные об этой траектории.

Вероятно, анализатор решений можно будет использовать и для оптимизации отдельных приемов. Однако, попытки изменить прием для сокращения решения одной задачи обычно приводят к ухудшению хотя бы части решений других задач. Поэтому основным источником оптимизации приема оказывается анализ его поведения на всем подразделе задачника.

База объектов

Логические процессы интеллектуальной системы взаимодействуют с внешним миром через различные нелогические структуры данных: битмэпы изображений, численные массивы, схемы, таблицы, тексты естественного языка, географические карты, компьютерные программы, планы действий, технические проекты, задания на проектирование и т. п. Часть этих структур данных имеет статический характер и пополняет общие знания системы о внешнем мире, часть — динамически изменяется и позволяет системе реагировать на текущие процессы окружающей среды. Для представления в логической системе такой «нелогической» информационной среды создана структура, названная базой объектов. Технически она организована в виде оглавления, концевыми пунктами которого служат различные технические конструкции: числа, многочлены, численные массивы, наборы и таблицы, сетевые объекты, фрагменты синтезируемых либо анализируемых СИ-программ, шахматные позиции и др. В оглавлении выделен раздел, через который решатель может работать с файлами произвольной природы, помещенными в поддиректорию FLS. Пока база объектов представляет собой заготовку для будущего подключения логики решателя к прикладным задачам. На текущий момент созданы лишь простейшие интерфейсы для ввода объектов перечисленных типов. Предусмотрен также интерфейс для создания ссылки на объект из хранящейся в задачнике задачи. Он позволит работать с базой объектов в полуавтоматическом режиме, выполняя с помощью решателя различные операции над ее элементами. По мере обучения системы, эти операции будут укрупняться. Проработаны простейшие задачи, связанные с сетевыми данными, хранящимися в базе объектов — поиск кратчайшего пути в графе и определение максимального потока через сеть. Программы для работы с сетями реализованы на ГЕНОЛОГе.

Усиленный режим решения задач

Процесс решения задачи человеком редко имеет прямолинейный характер. После первой, зачастую неудачной, попытки решить задачу «в лоб», предпринимается возвращение к исходному условию и реализуются повторные попытки, в которых каждый шаг анализируется более глубоко и связан с привлечением дополнительных средств. Фактически, на каждом шаге здесь имеет место определенный перебор, направленный на усмотрение какого-то локального выигрыша. Трудоемкость перебора на повторных попытках может достигать весьма значительной величины, а привлекаемые средства могут оказаться весьма удаленными от исходных понятий задачи. Иногда для решения задачи приходится временно отвлекаться от нее и развивать примыкающие разделы теории.

Соответственно, и при обучении решателя нет смысла ограничиваться только лобовым подходом к решению задачи. Впрочем, при первичном обучении проще всего начинать все-таки именно с него. Когда эксперт разбивает ход решения задачи на шаги, ему уже известны все подводные камни предметной области, и вполне естественным является желание вводить такие приемы, срабатывание которых являлось бы хорошо мотивированным. К сожалению, часто хорошо мотивированным оказывается применение сразу целого блока логических переходов, в то время как мотивировка отдельных шагов внутри блока трудно объяснима. В результате появляются приемы, имеющие характер «заготовок» для специальных ситуаций. В задачниках это явление хорошо известно: предыдущая задача иногда может рекомендоваться как прием для решения следующей. Естественно, ее имеет смысл запомнить и сохранить в базе приемов решателя. Такое накопление «стандартных заготовок» может выглядеть, как некоторый обман, однако он подсказан самой логикой усиления лобового режима решателя. Результатом оказывается существенное ускорение получения ответов для стандартных задач. Оправданием может служить и то, что эксперты тоже обычно располагают большим запасом стандартных заготовок и сразу усматривают их в задачах.

С другой стороны, решатель все же должен справляться с задачами и без искусственных заготовок, которые для всех случаев жизни заранее запасти в базе приемов не удастся. Поэтому какие-то слабо мотивированные приемы ему нужны. Чтобы подключить их к решению задачи, имеются две возможности.

Первая — поднять уровень срабатывания слабо мотивированных приемов. В общем, иногда это удается сделать безболезненно, и за счет таких приемов решаются многие задачи. Однако, здесь имеются две трудности. Первая заключается в том, что на высоких уровнях срабатывания часто размещаются и сильно мотивированные действия. Они попадают туда из-за проблем, связанных с переключением внимания: проще бывает не понижать веса посылок для обеспечения срабатывания приема, так как это всегда притормаживает процесс решения, а продублировать прием, чтобы он мог сработать и на высоком текущем уровне сканирования. В этой ситуации поднятые даже на очень высокий уровень слабо мотивированные приемы будут сильно «зашумлять» работу системы. Вторая трудность заключается в том, что иногда локальный перебор, использующий слабо мотивированные переходы, необходимо применять в самом начале решения задачи. Он позволит найти какой-то ключевой момент, без которого стандартными средствами задача быстро окажется заведенной в тупик, и тогда подключение того же перебора на последующих этапах уже не поможет.

Вторая возможность подключения слабо мотивированных приемов заключается в том, чтобы после неудачной лобовой попытки решения возвратиться к исходной формулировке задачи и повторить попытку, разрешая теперь пользоваться дополнительными приемами. Такие приемы не должны предпринимать каких-либо скоропалительных изменений задачи, тем более слабо мотивированных. Однако, они могут обращаться к различного рода циклам локального перебора для поиска очередного «сильного» хода решения. Внутри этих циклов и будут допускаться слабо мотивированные попытки. Данный режим решения, в отличие «обычного» лобового режима решателя, будем называть усиленным. Хотя пока рассматривается только один уровень усиления, впоследствии, видимо, понадобится целая шкала таких уровней, с постепенным подключением все более мощных дополнительных средств, но и со все более медленным временным масштабом процессов.

Еще раз подчеркнем, что смысл усиленного режима состоит не в ослаблении фильтрации срабатываний приемов — такое ослабление сделает систему неспособной решать даже простые задачи. Эффект усиления достигается за счет более осмотрительного, с анализом на достаточно большую глубину, каждого отдельного шага преобразований задачи. Хотя эти шаги могут оказаться весьма трудоемкими и

в разы замедлят решение задачи, но замедление не будет иметь экспоненциального характера, а списки посылок и условий не окажутся захламлены огромным числом ненужных утверждений.

Развитие усилителя было начато на примере геометрических вычислительных задач. Чтобы подключить усиленный режим, вводится комментарий «усиление» к посылкам исходной задачи. Он может появиться, если изначально задача из задачника имела в качестве дополнительной посылки символ «усиление», либо если имел место откат к повторной попытке после неудачной лобовой попытки. Для геометрических задач такой откат предпринимается автоматически — если был получен отказ либо если был превышен заданный лимит трудоемкости.

Наиболее эффективным средством усиления решателя в планиметрии оказались пакетные анализаторы. Они получают в качестве входного данного копию текущей задачи на исследование и преобразуют ее с помощью собственного списка приемов, существенно меньшего, чем вся база приемов сканирования задач. Это обеспечивает значительное ускорение и позволяет провести перебор на сравнительно большую глубину. Источником усиления здесь является ограничение применяемых средств. В одном анализаторе акцент делается на применение теоремы косинусов, в другом — на анализ равных углов и фигур, в третьем — на соотношения пропорциональности, и т. д. Обращения к пакетным анализаторам реализуются приемами сканирования задач, срабатывающими на различных уровнях, в том числе на совсем маленьких. По мере повышения уровня увеличиваются ресурсы, отводимые на попытку обращения. При завершении работы анализатора в его списке посылок отбираются новые ценные факты, которые переносятся в список посылок текущей задачи. Таким образом, несмотря на большое количество выводимых анализаторами вспомогательных утверждений, размеры списка посылок основной задачи растут умеренным образом.

При первичном обучении решателя планиметрическим задачам было создано значительное число приемов, имеющих характер искусственных «заготовок». Впоследствии те задачи, ради которых они создавались, были переработаны для усиленного режима, уже без подобных заготовок. Разумеется, время решения при этом значительно увеличилось.

Все-таки, подавляющее большинство отказов решателя оказалось связано не с увязанием в переборе, а с отсутствием на текущий мо-

мент одного — двух простых приемов, легко извлекаемых из обычной теории. Поэтому главными средствами усиления являются, во-первых, заблаговременное создание достаточно полной коллекции простейших приемов, и, во-вторых, пополнение этой коллекции в процессе решения задачи. Чтобы получить такое усиление, необходим анализ цепочки превращений на пути от теоремы к приему.

Логический ассемблер

Ручное обучение решателя дает огромный материал для анализа процессов извлечения приемов из теорем. Однако, этот материал чрезвычайно сырой: каждый прием решателя был подсказан контекстом одной или нескольких конкретных задач, и никаких попыток стандартизации приемов не предпринималось. Как следствие, приемы иногда оказывались слишком частными, иногда — неоправданно общими, а в ряде случаев и вовсе неадекватными, так как задачу следовало решать совсем другими средствами. Помимо не очень высокого качества работы решателя, такое положение вещей создало принципиальные трудности для автоматического синтеза приемов.

Поэтому следующим этапом работы, необходимым для выхода на новый технологический уровень, оказалась переработка всей базы приемов, ориентированная на создание единой системы стандартных типов приемов. Как и многим другим естественным наукам, начинавшимся с общей классификации изучаемых объектов, науке о логических процессах тоже приходится начинать с такого этапа. Впрочем, речь здесь идет не просто о классификации, а о создании языка для задания приемов, имеющего более высокий уровень, чем ГЕНОЛОГ. В то время, как ГЕНОЛОГ подробно описывает, каким образом теорема применяется при решении задачи, новый язык определяет прием, сопровождая теорему лишь указанием той цели, ради которой она применяется. Это достигается заданием типа приема и нескольких уточняющих параметров (термов, переменных и т. п.) Каждый тип предполагает свой конкретный набор параметров. Ситуация до некоторой степени напоминает язык ассемблера, где вместо типа приема имеется тип операции. Поэтому новый язык получил название логического ассемблера. Само задание приема на данном языке названо его спецификацией.

Для развития логического ассемблера был создан несложный интерфейс, позволяющий пролистывать базу приемов и выполнять предварительную их классификацию — выявление типов приемов. Предпринят также предварительный цикл пролистывания этой классификации и ее уточнения. Если удавалось найти более естественную траекторию решения задачи, сомнительные приемы отбрасывались. Часть приемов была скорректирована для стандартизации и уменьшения числа типов. При этом преследовалась цель сохранения лишь достаточно сильно мотивированных действий. Система типов организована древовидным образом: для заданного типа иногда рассматриваются его подтипы, имеющие более сильную мотивировку срабатывания. Это необходимо, чтобы решатель мог адаптироваться к предметной области, выбирая для каждого приема свой необходимый уровень мотивированности.

На текущий момент в системе зарегистрированы 1335 типов приемов. Из них лишь 799 типов связаны более чем с одним приемом, 590 — более чем с двумя, и 224 типа — не менее чем с десятью. Вероятно, в процессе дальнейшей проработки многие «одноразовые» типы будут исключены, прочие — уточнены. Впрочем, имеется достаточное количество типов приемов, которые были введены ради единственного приема, но, безусловно, необходимы.

Приведем несколько десятков выбранных наугад примеров типов приемов. Каждая спецификация состоит из терма «тип(A)», указывающего тип приема A, и нескольких сопровождающих термов либо символов. Здесь A — логический символ, выбираемый для ссылок на тип приема достаточно случайным образом. В оглавлении типов приводится текст, поясняющий смысл соответствующих действий. Далее вместо названия типа A будем приводить лишь поясняющий текст из оглавления.

1) Общая стандартизация выражения.
   Теорема приема представляет собой тождество, применяемое для общей стандартизации выражений в любых ситуациях. Заменяемая и заменяющая части тождества не имеют связанных переменных. Дополнительный элемент спецификации «направл(N)» уточняет направление замены (слева направо либо справа налево). Пример приема данного типа:
   $$\forall_{ab}(a \subseteq b \to a \cup b = b).$$
2) Исключение описателя «класс».

Теорема приема — тождество, в заменяемой части которого расположено выражение вида $\mathrm{set}_x A(x)$, а заменяющая не имеет связанных переменных. Здесь $\mathrm{set}_x A(x)$ — множество всех элементов либо наборов x, удовлетворяющих условию $A(x)$. Направление замены уточняется элементом «направл(...)». Пример приема данного типа:

$$\forall_{fA}(\mathrm{set}_x(x \in \mathrm{Dom}(f)\ \&\ f(x) \in A) = \text{прообраз}(f, A)).$$

3) Непосредственное определение характеристики отображения.
Теорема приема — тождество, в заменяемой части которого расположено выражение вида $f(\lambda_x(t(x), A(x)))$, а заменяющая не имеет связанных переменных. Здесь $\lambda_x(t(x), A(x))$ — функция, определенная на множестве всех значений x, удовлетворяющих условию $A(x)$, причем значение ее определяется выражением $t(x)$. Направление замены уточняется элементом «направл(...)». Пример приема данного типа:

$$\forall_{abcdf}(\text{образ}(\lambda_x(\mathrm{arctg}\, x, f(x)), [a, b]) = [\mathrm{arctg}\, a, \mathrm{arctg}\, b]).$$

4) Сведение операции над семейством к операциям над семействами, имеющими более простой вид общего члена.
Теорема приема — тождество, позволяющее выносить константные подвыражения за знак операции над семейством. Единственный дополнительный элемент спецификации — «направл». Пример приема данного типа:

$$\forall_{afgh}(\sum_{x,f(x)} \frac{ag(x)}{h(x)} = a \sum_{x,f(x)} \frac{g(x)}{h(x)}).$$

5) Применение нормализатора общей стандартизации для вычисления.
Теорема приема имеет вид $\forall_{xy}(x = t(y) \to t(y) = x)$. Здесь x — переменная, y — набор переменных, $t(y)$ — выражение, для которого соответствующий нормализатор общей стандартизации обеспечивает «вычисление» его значения. Антецедент теоремы реализует обращение к данному нормализатору, и далее $t(y)$ заменяется на x. Спецификация имеет элемент «направл(...)», а также элемент, уточняющий вид выражения $t(y)$, для которого гарантирована возможность вычисления. Пример приема данного типа:

$$\forall_{abc}(c = a \times b \to a \times b = c).$$

Здесь $\times$ — прямое произведение множеств. Дополнительные условия на вид заменяемого выражения состоят в том, что a и b имеют вид конечных списков $\{p_1, \ldots, p_k\}$. В этой ситуации нормализатор «нормпрямоепроизведение» преобразует выражение к виду конечного списка пар.

6) Переход в задаче на преобразование к уже имевшимся подтермам.
Теорема приема — тождество, позволяющее таким образом проварьировать «сложное» подвыражение условие задачи на преобразование, что новая его версия уже встречается в другой части условия. Спецификация имеет элемент «направл(...)», элемент «исключение(t_1)», указывающий на исключаемое сложное подвыражение t_1, и элемент «терм(t_2)», указывающий новую его версию. Пример приема данного типа:

$$\forall_{abcd}(0 < a \to a^{d \log_b c} = c^{d \log_b a}).$$

Тождество применяется слева направо. Исключается выражение $\log_b c$, вводится новая его версия $\log_b a$. Проверяется, что она уже встречается в условии задачи. Для блокировки обратной замены вводится специальный комментарий.

7) Сокращенная переформулировка выражений при завершающем редактировании.
Теорема приема — тождество, позволяющее переформулировать выражение более компактным образом перед выдачей ответа. Предполагается, что применение его на более ранних этапах решения задачи либо противоречит используемой на этих этапах стандартизации, либо не представляет особой ценности. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{abcde}(a - \text{рациональное } \& \neg(\text{числитель}(a) - \text{четное}) \,\&$$
$$\neg(\text{знаменатель}(a) - \text{четное}) \to -(b-c)^a d/e = (c-b)^a d/e).$$

Этот прием позволяет «втягивать» внешний минус в сомножитель числителя, имеющий вид разности. Он применяется только при редактировании ответов задач на преобразование либо на описание.

8) Группировка относительно выражения с неизвестными.

Теорема приема — тождество, выполняющее группировку относительно неизвестных, то есть «вынесение за скобку» общего неизвестного фрагмента нескольких подвыражений. При этом в скобках остается выражение, не содержащее неизвестных. Спецификация имеет элемент «направл(...)», а также элемент «неизвестные($x_1 \dots x_k$)», перечисляющий переменные x_i, идентифицируемые с содержащими неизвестные выражениями. Пример приема данного типа:

$$\forall_{abcdefghij}(0 \leqslant a \ \& \ 0 \leqslant d \to a^{e+cj/fg} d^{i+bj/fh} = a^e d^i (a^{c/g} d^{b/h})^{j/f}).$$

Замена выполняется слева направо, спецификация имеет элемент «неизвестные($j \ f$)». Остальные переменные идентифицируются с выражениями, не содержащими неизвестных.

9) Преобразование, приводящее к возможности упрощения надтерма.
Теорема приема — тождество, которое позволяет так преобразовать выражение, что после этого можно выполнить сильное упрощение его надвыражения. Спецификация имеет элемент «направл(...)», а также элемент, уточняющий вид надвыражения. Простейший пример приема данного типа:

$$\forall_a(\frac{\sin a}{\cos a} = \operatorname{tg} a).$$

Замена выполняется слева направо, причем заменяемая дробь является операндом арктангенса либо арккотангенса. Несколько более сложный прием — попытка усмотрения в сумме с радикалами полного квадрата, если эта сумма сама находится под радикалом.

10) Стандартизация с помощью вычислений.
Теорема приема — тождество, позволяющее упростить выражение с помощью вычисления значений операций над встречающимися в нем константами. Спецификация имеет элемент «направл(...)», а также элемент, уточняющий, какие именно переменные идентифицируются с константами, и задающий типы этих констант. Пример приема данного типа:

$$\forall_{abcdefpq}(p = bf + de \ \& \ q = df \to ab/cd + ae/cf = pa/qc).$$

Замена выполняется слева направо. Переменные b, d, e, f идентифицируются с десятичными константами. Антецеденты теоремы выполняют вычисление новых константных значений p, q.

11) Свертка условного выражения.
Теорема приема — тождество, позволяющее преобразовать условное выражение в безусловное. Спецификация имеет единственный дополнительный элемент «направл(...)». Пример приема данного типа:

$$\forall_{ab}(\neg(a=0)\ \&\ b - \text{число} \to (b \text{ при } 0<a,\ \text{иначе}\ -b) = b\mathrm{sg}(a)).$$

12) Непосредственное исключение сложной операции.
Теорема приема — тождество, уменьшающее эвристическую оценку сложности встречающихся в выражении операций. Рассматривается максимум таких оценок для всех операций, имеющихся в заменяемом выражении, и он оказывается строго больше аналогичного максимума для заменяющего терма. Спецификация имеет единственный дополнительный элемент «направл(...)». Пример приема данного типа:

$$\forall_a(0 \leqslant a+1\ \&\ 0 \leqslant 1-a \to \cos(3\arcsin a) = (1-4a^2)\sqrt{1-a^2}).$$

Преобразование выполняется слева направо и позволяет исключить тригонометрические операции.

13) Декомпозиция сложной операции.
Теорема приема — тождество, выражающее сложную в эвристическом смысле операцию через несколько операций той же сложности, но зависящих лишь от части переменных исходной операции. Спецификация имеет единственный дополнительный элемент «направл(...)». Пример приема данного типа:

$$\forall_{abc}(0 \leqslant a\ \&\ 0 \leqslant b \to \log_c(ab) = \log_c a + \log_c b).$$

Здесь происходит декомпозиция логарифма. Заметим, что прием имеет фильтр, блокирующий преобразование, если выражения a, b известны, а c содержит неизвестные, либо если логарифм используется при определении значения некоторой функции, и c содержит варьируемую переменную, а a, b не содержат. Этот и другие фильтры порождаются по спецификации приема автоматически.

14) Группировка сложных операций.
Теорема приема — тождество, позволяющее сгруппировать несколько «сложных» операций в одну равноценную операцию.

Спецификация имеет единственный дополнительный элемент «направл(...)». Пример приема данного типа:

$$\forall_{abcd}(c\sqrt{\frac{a}{b}} + d\sqrt{\frac{b}{a}} = \frac{(ac+bd)\sqrt{a/b}}{a}).$$

Здесь два радикала группируются в один.

15) Специальная стандартизация условия задачи на преобразование.
Теорема приема — тождество, используемое для стандартизации выражений в специальном контексте, определяемом свойствами понятий предметной области. Кроме элемента «направл(...)», спецификация имеет элементы, уточняющие контекст. Пример приема данного типа:

$$\forall_{abcde}(p = a\sin b\sin c/d + e \to a\sin b\sin c/d + e = p).$$

Преобразуемая сумма является условием вспомогательной задачи на упрощение подынтегрального выражения. Переменная интегрирования входит в выражения b, c и не входит в d. Кроме того, она не встречается под логарифмом либо под обратной тригонометрической функцией. Антецедент обращается к пакетному нормализатору, определяющему результат p раскрывания скобок и преобразования тригонометрических произведений в суммы.
Принятие решения о контекстной стандартизации требует предварительного анализа не одной теоремы, а целой группы теорем предметной области. По итогам такого анализа в базе теорем фиксируются «протоколы стандартизации», являющиеся источниками соответствующих приемов.

16) Безусловная общая стандартизация одного утверждения.
Теорема приема — эквивалентность, обеспечивающая упрощение элементарного утверждения и применяемая в любых ситуациях, кроме тех, когда она заведомо нарушает принятую стандартизацию. Единственный дополнительный элемент спецификации — «направл(...)». Простейший пример приема данного типа:

$$\forall_{ab}(\{a\} = \{b\} \leftrightarrow a = b).$$

Другой, чуть более сложный пример:

$$\forall_{abc}(a \subseteq b \cup c \leftrightarrow a \setminus b \subseteq c).$$

Замена выполняется справа налево. Применение приема сопровождается радом стандартных ограничений. Например, если c — неизвестная, а выражения a и b известны, то преобразование отменяется. Эти ограничения автоматически создаются по спецификации.

17) Конъюнктивная декомпозиция элементарного утверждения.
Теорема приема — эквивалентность, преобразующая элементарное утверждение в конъюнкцию элементарных утверждений. Переменные каждого конъюнктивного члена заменяющей части образуют собственное подмножество переменных заменяемой части. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{abc}(a \in b \setminus c \leftrightarrow \neg(a \in c)\ \&\ a \in b).$$

18) Общая стандартизация группы посылок.
Теорема приема — эквивалентность, позволяющая заменить несколько элементарных посылок задачи на одну такую посылку, без какого-либо ущерба для последующего решения. Единственный дополнительный элемент спецификации — «направл(...)». Аналогичный тип введен для стандартизации группы условий. Пример приема данного типа:

$$\forall_{ab}(a = b \leftrightarrow a \subseteq b\ \&\ b \subseteq a).$$

Замена выполняется справа налево.

19) Дизъюнктивно — конъюнктивная свертка в условии задачи на компактную переформулировку утверждения («свертку»).
Теорема приема — эквивалентность, позволяющая преобразовать дизъюнктивно — конъюнктивную конструкцию, составленную из элементарных утверждений, в одно элементарное утверждение. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{abc}(a \in b \cup c \leftrightarrow a \in b\ \vee\ a \in c).$$

Замена выполняется справа налево. Прием применяется в задачах на описание, целевая установка которых явно указывает на необходимость компактной переформулировки условий.

20) Кванторная свертка.
Теорема приема — эквивалентность, позволяющая переформулировать бескванторным образом квантор общности либо существования. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{ab}(\text{непересек}(a,b) \leftrightarrow \forall_c(c \in a \rightarrow \neg(c \in b))).$$

Замена выполняется справа налево. Прием сопровождается группой стандартных фильтров, блокирующих преобразование, если оно противоречит текущей целевой установке. Все эти фильтры генерируются автоматически. Исключение квантора обеспечивается даже в тех случаях, когда требуются неслоежные предварительные преобразования — переход от квантора существования к отрицанию квантора общности либо группировка части антецедентов под внутренний квантор общности.

21) Кванторная расшифровка в условии задачи на доказательство.
Теорема приема — эквивалентность, позволяющая перейти к кванторной расшифровке элементарного утверждения. Прием применяется к подутверждению условия задачи на доказательство — корневому либо находящемуся под корневым отрицанием. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{ab}(a - \text{set} \ \& \ b - \text{set} \rightarrow a = b \leftrightarrow \forall_c(c \in a \leftrightarrow c \in b)).$$

Замена выполняется справа налево. Фильтры приема, уточняющие контекст его применения, генерируются по спецификации автоматически.

22) Попытка использования явного параметрического описания при получении частичного ответа.
Теорема приема — эквивалентность, дающая явное параметрическое описание для значений переменной, при которых истинно заданное элементарное утверждение. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{bc}(b \in c \leftrightarrow \exists_a(c = a \cup \{b\} \ \& \ a - \text{set})).$$

Левая часть эквивалентности идентифицируется с условием задачи на описание, причем c — неизвестная этой задачи, не вхо-

дящая в выражение b. В задаче требуется найти лишь пример значений неизвестных.

23) Разрешение условия задачи на описание либо посылки задачи на исследование относительно заданных неизвестных.
Теорема приема — эквивалентность, заменяющая часть которой дает явное разрешение заменяемой части относительно заданных переменных. Дополнительные элементы спецификации — «направл(...)» и «неизвестные($x_1 \dots x_k$)», где $x_1, \dots, x_k$ — переменные, относительно которых происходит разрешение. Пример приема данного типа:

$$\forall_{abc}(b < ac \leftrightarrow c < 0 \;\&\; a < b/c \vee 0 < c \;\&\; b/c < a \vee c = 0 \;\&\; b < 0).$$

Замена выполняется слева направо, неизвестной считается переменная a.

24) Свертка группы неизвестных условий в одно условие, явно разрешенное относительно неизвестных.
Теорема приема — эквивалентность, позволяющая объединить несколько условий задачи на описание, явно разрешенных относительно неизвестной, в одно условие, явно разрешенное относительно этой неизвестной. Дополнительные элементы спецификации — «направл(...)» и «неизвестные(x)», где x — неизвестная. Пример приема данного типа:

$$\forall_{abc}(a \cup b \subseteq c \leftrightarrow a \subseteq c \;\&\; b \subseteq c).$$

Замена выполняется справа налево. Переменная c идентифицируется с неизвестной, переменные a, b — с выражениями, не содержащими неизвестных.

25) Преобразование условия задачи на описание, исключающее сложное выражение с неизвестными.
Теорема приема — эквивалентность, позволяющая исключить наиболее сложное в эвристическом смысле подвыражение заменяемой части. Дополнительные элементы спецификации — «направл(...)» и «терм(t)», где t — исключаемое подвыражение. Пример приема данного типа — возведение уравнения в квадрат для исключения неизвестного радикала:

$$\forall_{abcdp}(0 \leqslant d \to ad^{p/2} + b = c \leftrightarrow a^2 d^p - b^2 + 2bc = c^2 \;\&\; 0 \leqslant (c - b)a).$$

Переменная p идентифицируется с натуральной константой. Выражение d содержит неизвестные, c — не содержит. Исключается подвыражение $d^{p/2}$.

26) Переход к дизъюнктивной посылке, определяющей значение переменной, входящей в условие.
Теорема приема — эквивалентность, позволяющая преобразовать элементарную посылку к виду дизъюнкции, один из членов которой явно выражает некоторую переменную, входящую в условие задачи. Таким образом подготавливается полезный разбор случаев. Дополнительные элементы спецификации — «направл(...)» и «переменная(x)», где x — переменная, для которой находится явное выражение. Пример приема данного типа:

$$\forall_{abc}(a \in \{b; c\} \leftrightarrow a = b \ \vee \ a \in \{; c\}).$$

Замена выполняется слева направо. Здесь $\{b; c\}$ — конечный список, в котором выделен некоторый элемент b, причем c — остальные элементы списка. В спецификации указана переменная a. Проверяется, что она не входит в выражение b, но встречается в условии задачи. Фильтры приема генерируются по спецификации автоматически.

27) Развертка квантора существования в дизъюнкцию.
Теорема приема представляет собой эквивалентность, в обеих частях которой находятся кванторы существования. Заменяющий квантор явно указывает конечное множество A допустимых значений своей варьируемой переменной. Прием разворачивает его в дизъюнкцию, члены которой соответствуют элементам множества A. Единственный дополнительный элемент спецификации — «направл(...)». Пример приема данного типа:

$$\forall_{fmkpx}(p = k - m \ \& \ k - \text{целое} \ \& \ m - \text{целое} \rightarrow \exists_n(n - \text{целое} \ \& \ x = f(n) \ \& \ m \leqslant n \ \& \ n \leqslant k) \leftrightarrow \exists_l(l \in \{0, \ldots, p\} \ \& \ x = f(m + l))).$$

Замена выполняется слева направо. Первый антецедент находит результат p упрощения разности выражений k, m. Проверяется, что этот результат — натуральная константа, меньшая 9. Правый квантор существования разворачивается в дизъюнкцию для значений $l = 0, \ldots, p$.

28) Разрешение неконстантного выражения относительно констант.

Теорема приема — эквивалентность, позволяющая разрешить элементарное утверждение относительно некоторой переменной, в предположении, что эта переменная идентифицирована с неконстантным выражением, а прочие переменные — с константными выражениями. Дополнительные элементы спецификации — «направл(...)» и «терм(x)», где x — переменная, относительно которой выполняется разрешение. Пример приема данного типа:

$$\forall_{ab}(a < -b \leftrightarrow b < -a).$$

Замена выполняется слева направо. Выражение b неконстантное, a — константное.

29) Непосредственное усмотрение истинности либо ложности.
Теорема приема не имеет вида $\forall_{x_1 \ldots x_n} A(x_1 \ldots x_n)$, где $A(x_1 \ldots x_n)$ — элементарное утверждение. Если в задаче встречается утверждение вида $A(t_1 \ldots t_n)$, быть может, с отброшенным корневым отрицанием, то прием заменяет его на константу «истина» либо, соответственно, «ложь». Спецификация не имеет дополнительных элементов. Пример приема данного типа:

$$\forall_a(a \subseteq a).$$

30) Усмотрение истинности либо ложности с помощью проверочного оператора.
Теорема приема имеет вид кванторной импликации $\forall_{x_1 \ldots x_n}(A_1 \;\&\; \ldots \;\&\; A_n \to A_0)$, где консеквент A_0 — элементарное утверждение, а антецеденты $A_1, \ldots A_n$ допускают проверку с помощью пакетных проверочных операторов. Прием использует эти операторы для усмотрения истинности утверждения A_0, которое заменяется на константу «истина». Спецификация не имеет дополнительных элементов. Пример приема данного типа:

$$\forall_{ab}(b \leqslant a \to \neg(a < b)).$$

Прием усматривает ложность строго неравенства, проверяя истинность встречного нестрогого.

31) Вывод одноместного предиката.
Теорема приема — кванторная импликация, консеквент которой имеет вид $P(x)$, где x — переменная, P — символ одноместного

отношения. Часть антецедентов теоремы приема идентифицируются с посылками задачи, остальные — проверяются с помощью вспомогательных средств. После этого выводится новая посылка $P(x)$. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABC}(A \in \text{отрезок}(BC) \to A - \text{точка}).$$

32) Вывод равенства двух числовых атомов.
Теорема приема — кванторная импликация, имеющая консеквент вида $t_1 = t_2$, где t_1, t_2 — атомарные числовые выражения, не являющиеся переменными. Здесь и далее называем числовое выражение $f(A_1 \dots A_m)$ атомарным, или числовым атомом, если хотя бы одно из подвыражений A_i принимает нечисловые значения. Численные переменные называем вырожденными числовыми атомами. Примеры невырожденных числовых атомов — «расстояние(A B)», «масса(A)», «угол(ABC)», «скорость(A t)», и т. п. Часть антецедентов теоремы приема идентифицируются с посылками, остальные — проверяются вспомогательными средствами. После этого к посылкам задачи присоединяется равенство $t_1 = t_2$. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABCD}(\text{прямая}(AC) \perp \text{прямая}(BD)\ \&\ B \in \text{прямая}(AC)\ \&$$
$$\angle(ADB) = \angle(BDC) \&\ \text{разныеточки}(B, D) \to l(AB) = l(BC)).$$

Прием усматривает, что BD — высота треугольника ADC и одновременно его биссектриса. Из этого выводится следствие, что BD — медиана.

Заметим, что в приемах данного типа не накладывается никаких дополнительных ограничений на рассматриваемые числовые атомы. Иногда неограниченный вывод следствий нежелателен, и тогда используются подтипы этого типа, в которых действия более мотивированы. Например, возможен следующий подтип:

33) Вывод равенства двух старых числовых атомов.
Теорема приема — такого же вида, как в предыдущем случае, но перед выводом соотношения $t_1 = t_2$ проверяется, что числовые атомы t_1, t_2 уже рассматриваются в задаче. Дополнительных элементов спецификации нет. Пример приема данного типа:

$$\forall_{ABCDE}(\text{актив}(\angle(DAE))\ \&\ \text{актив}(\angle(BAC))\ \&$$
$$A \in \text{отрезок}(BE)\ \&A \in \text{отрезок}(CD) \rightarrow$$
$$\angle(BAC) = \angle(DAE)).$$

Выводится равенство вертикальных углов BAC и DAE, которые уже встречаются в посылках задачи. Посылка «актив(X)» означает, что расстояние либо угол X уже встречаются в задаче. Такие посылки заблаговременно создаются другими приемами решателя.
Как уже замечалось выше, выбор необходимой степени мотивированности (иными словами, подтипа в ветви дерева типов) происходит при доводке приемов на задачнике.

34) Общий случай соотношения для числовых атомов.
Теорема приема позволяет вывести некоторое уравнение для невырожденных числовых атомов. Вывод реализуется в посылках задачи на исследование либо на доказательство, причем никаких дополнительных ограничений на срабатывание приема не накладывается. Спецификация состоит только из указателя типа приема. Пример приема данного типа:

$$\forall_{ABCDE}(\text{прямая}(AB) \perp \text{прямая}(AC)\ \&$$
$$\text{окружность}(DE)\text{вписана в фигура}(ABC) \rightarrow$$
$$2l(DE) = l(AB) + l(AC) - l(BC)).$$

Всякий раз, как только в задаче усматривается окружность, вписанная в прямоугольный треугольник, выводится равенство, выражающее ее радиус через длины сторон треугольника.

Рассматриваемый тип приемов является корнем большой ветви подтипов. Эти подтипы возникли после анализа приемов решения планиметрических задач, используемых решателем. Выбор необходимого подтипа при обучении системы происходит после «примерки» приема на задачнике. Для автоматизации такой примерки создана процедура доводчика приемов. Приведем несколько примеров подтипов данного типа.

35) Соотношение для старых числовых атомов.
Теорема приема позволяет вывести некоторое соотношение для невырожденных числовых атомов. Вывод реализуется только

в том случае, если все эти атомы уже встречаются в задаче. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABC}(\text{актив}(l(AB)) \ \&\ B \in \text{отрезок}(AC) \rightarrow l(AC) = l(AB) + l(BC)).$$

Соотношение, связывающее длину отрезка с длинами двух подотрезков выводится, если все эти длины уже встречаются в задаче.

36) Определение старого числового атома.
Теорема приема позволяет вывести некоторое соотношение для невырожденных числовых атомов. Вывод реализуется только в том случае, если все эти атомы уже встречаются в задаче, причем один из них не известен, а остальные — известны. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABC}(\text{прямая}(AB) \perp \text{прямая}(AC) \ \&\ \text{актив}(\angle(ABC)) \rightarrow l(AC) = \sin(\angle(ABC))l(BC)).$$

Соотношение связывает длину катета с длиной гипотенузы и синусом острого угла. Оно выводится, если одна из величин $l(AC)$, $l(BC)$, $\angle(ABC)$ не известна, а две другие — известны.

37) Определение числового атома «неизв».
Теорема приема позволяет вывести некоторое соотношение для невырожденных числовых атомов. Вывод реализуется только в том случае, если все эти атомы уже встречаются в задаче, причем один из них не известен и встречается в уравнениях с численными неизвестными, а остальные — известны. Заметим, что невырожденные числовые атомы, встречающиеся в уравнениях с численными неизвестными, называются атомами типа «неизв». Определение их значения позволяет, в конечном счете, получить для численной неизвестной «обычное» уравнение, имеющее только численные параметры. Данный тип приемов относится в геометрии к наиболее мотивированным. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABCDEF}(\text{прямая}(AB) \parallel \text{прямая}(DE)\ \&$$
$$\text{прямая}(AC) \parallel \text{прямая}(DF)\ \&\ \text{прямая}(BC) \parallel \text{прямая}(EF)\ \&$$
$$\text{актив}(l(AB))\ \&\ \text{актив}(l(BC))\ \&\ \text{актив}(l(DE))\ \&\text{актив}(l(EF))\ \&$$
$$\text{разныепрямые}(\text{прямая}(AB), \text{прямая}(BC)) \rightarrow$$
$$l(AB)l(EF) = l(BC)l(DE)).$$

Здесь усматривается подобие треугольников ABC и DEF, имеющих параллельные соответственные стороны, и выводится соотношение пропорциональности длин сторон. Одна из этих длин имеет тип «неизв», а три оставшиеся — известны.

В некоторых случаях ограничение на невырожденные числовые атомы, встречающиеся в выводимом равенстве, имеет асимметричный характер. Иногда это вызвано асимметричным характером самого соотношения, иногда — необходимостью ускорить отсечение ненужных ситуаций при идентификации. Такое ускорение объясняется тем, что проверять условие, накладываемое на числовой атом, можно сразу же после его идентификации, не дожидаясь идентификации остальных атомов. Приведем пример типа приема с асимметричным ограничением:

38) Определение заданного старого числового атома.
Теорема приема позволяет вывести некоторое соотношение для невырожденных числовых атомов. Вывод реализуется только в том случае, когда заданный числовой атом этого соотношения не известен, но уже встречается в задаче, а все остальные атомы — известны. Дополнительный элемент спецификации — «терм(p)», указывающий определяемый числовой атом p. Пример приема данного типа:

$$\forall_{ABC}(\text{актив}(\angle(BAC)) \rightarrow$$
$$2l(AB)l(AC)\cos(\angle(BAC)) = l(AB)^2 + l(AC)^2 - l(BC)^2).$$

Спецификация имеет элемент «терм(угол(BAC))». Прием выводит соотношение теоремы косинусов для определения неизвестного угла BAC, который уже рассматривается в задаче.

39) Вывод двуместного отношения в задаче на исследование либо на доказательство.

Теорема приема позволяет выводить утверждение вида $P(a, b)$, где P — символ отношения, отличного от равенства и числового неравенства; a и b — переменные, идентфицируемые с уже рассматриваемыми в задаче объектами. Дополнительный элемент спецификации — либо отсутствует, либо имеет вид «антецедент(i)» и указывает номер i антецедента, идентифицируемого с посылкой задачи. Пример приема данного типа:

$$\forall_{abc}(a \in b \ \& \ b \subseteq c \to a \in c).$$

40) Разбор случаев в задаче на исследование.
Теорема приема — кванторная импликация с дизъюнкцией в консеквенте. Прием реализует вывод этой дизъюнкции в посылках задачи на исследование, сопровождая ее комментарием, инициирующим разбор случаев. Дополнительные элементы спецификации уточняют контекст срабатывания. Пример приема данного типа:

$$\forall_{abx}(x \subseteq \{a; b\} \to a \in x \ \vee \ \neg(a \in x)).$$

Здесь x — неизвестная внешней задачи на описание, a — известно.

41) Усмотрение противоречия в посылках задачи на исследование, возникшей при разборе случаев.
Теорема приема — кванторная импликация, имеющая своим консеквентом символ «ложь». Прием усматривает в посылках задачи на исследование ситуацию, описываемую антецедентами теоремы, и выводит новую посылку «ложь». Он применяется только в задачах, возникших после разбора случаев, то есть если имеется некоторая вероятность нереализуемости рассматриваемого подслучая. Такие задачи распознаются по цели «контроль». Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ABa}(l(AB) = a \ \& \ a < 0 \to \text{ложь}).$$

При усмотрении в посылках равенства $l(AB) = a$, где выражение a не содержит неизвестных, предпринимается быстрая попытка проверки отрицательности a.

42) Вывод утверждения существования.

Теорема приема имеет вид кванторной импликации, консеквентом которой служит квантор существования. Часть антецедентов идентифицируются с посылками, остальные — проверяются вспомогательными средствами. Прием выводит квантор существования, ориентируясь на то, что сразу вслед за этим для его связанных переменных в задаче будут введены вспомогательные обозначения, а сам квантор преобразован в конъюнкцию подкванторных утверждений. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{fgAB}(\text{Отображение}(f,A,B)\ \&\ \text{Отображение}(g,A,B)\ \&\ \neg(f=g)\to\exists_a(a\in A\ \&\ \neg(f(a)=g(a)))).$$

Если в посылках задачи рассматриваются различные отображения f и g с одинаковыми областью определения и областью значений, то вводится в рассмотрение точка a, на которой эти отображения принимают различные значения.

43) Вывод определения.
Теорема приема имеет вид кванторной импликации, консеквент которой является расшифровкой по определению некоторого антецедента A. Обычно этот консеквент является кванторной импликацией. Часть антецедентов (включая A) идентифицируются с посылками, часть — проверяются вспомогательными средствами. Прием выводит расшифровку посылки, идентифицированной с A, сохраняя также ее исходную версию. Он бывает полезен на ранних этапах использования нового понятия, когда не созданы средства, позволяющие работать с ним без расшифровки по определению. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_A(A\subseteq\mathbb{R}\ \&\ \text{замкнутое}(A)\to\forall_x(\text{предельныточка}(x,A)\to x\in A)).$$

Посылка «замкнутое(A)» сопровождается своей квантрной расшифровкой, которая может понадобиться для вывода следствий. Заметим, что эквивалентная замена рассматриваемой посылки привела бы к отключению приемов, реагирующих на утверждение о замкнутости множества A.

44) Вывод условия, ограничивающего значения неизвестного подвыражения конечным множеством.

Теорема приема имеет вид кванторной импликации, консеквент которой ограничивает значения некоторой переменной x конечным множеством. Прием усматривает истинность антецедентов, идентифицируя x с выражением, содержащим неизвестные задачи на описание. Затем ограничение на значения x присоединяется к списку условий задачи, чтобы далее предпринять разбор случаев по конечному множеству альтернатив. Единственный дополнительный элемент спецификации — «неизвестная(x)». Пример приема данного типа:

$$\forall_{abcdmn}(n - \text{натуральное } \& \ 0 < a \ \& \ d = [(c - m)/a] \ \& \\ m \leqslant b \ \& \ an + b = c \to n \leqslant d).$$

В условиях задачи на описание усматриваются уравнение $an + b = c$ и утверждение «n — натуральное», где выражение n содержит неизвестные. Переменные a, c идентифицируются с целочисленными константами. Проверяется условие $0 < a$. При помощи пакетного синтезатора находится константная нижняя оценка m значений выражения b. Наконец, вычисляется целочисленное значение d, равное $[(c - m)/a]$, и в список условий заносится неравенство $n \leqslant d$. Таким образом, значения выражения n ограничиваются конечным множеством $\{1, \ldots, d\}$.

45) Исключение несущественных неизвестных в задаче на описание при невырожденном ограничении.
Теорема приема имеет вид эквивалености, в заменяемой части которой находится квантор существования $\exists_{x_1 \ldots x_n}(A_1 \& \ldots \& A_m)$, а в заменяющей — некоторое элементарное утверждение B. Переменные $x_1, \ldots, x_n$ идентифицируются с несущественными неизвестными задачи на описание. Значения таких неизвестных не обязательно указывать в ответе, а достаточно лишь убедиться в их существовании. Проверяется, что все содержащие данные неизвестные условия задачи идентифицируются в точности с утверждениями $A_1, \ldots, A_m$. Затем эти условия удаляются, а вместо них добавляется условие B. Дополнительные элементы спецификации отсутствуют. Пример приема данного типа:

$$\forall_{ae}(\exists_d(d - \text{set } \& \ a \subseteq d \ \& \ d \subseteq e) \leftrightarrow a \subseteq e).$$

Если на несущественную неизвестную d накладываются лишь условия «d — set», $a \subseteq d$ и $d \subseteq e$, то эти условия отбрасываются, а вместо них вводится условие $a \subseteq e$.

46) Подбор примера значений неизвестных, не входящих в невырожденные условия.
Теорема приема имеет вид кванторной импликации, один из антецедентов которой — равенство $x = a$, где a не содержит x, а консеквент — элементарное утверждение B. Прием усматривает, что неизвестная x задачи на описание встречается только в условии B, а также, быть может, в простейших сопровождающих утверждениях — одноместных предикатах $P(x)$ либо отрицаниях равенств $\neg(x = t)$. Если в задаче нужно найти лишь пример значений неизвестных, то прием предпринимает попытку свести ее к вспомогательной задаче, в которой значение x положено равным a. Единственный дополнительный элемент спецификации — «неизвестная(x)». Пример приема данного типа:

$$\forall_{ax}(x = \varnothing \to \neg(a \in x)).$$

Чтобы реализовать условие $\neg(a \in x)$ с неизвестной x, прием предпринимает попытку положить x равным $\varnothing$.

47) Прием проверочного оператора.
Теорема приема — кванторная импликация, позволяющая усматривать истинность утверждений заданного вида, соответствующего рассматриваемому проверочному оператору. Единственный дополнительный элемент спецификации — «оператор(A)», где A — название проверочного оператора. Пример приема данного типа:

$$\forall_{ab}(0 < b \;\&\; 0 < a \to 0 < ab).$$

Элемент спецификации — «оператор(усмменьше)». Проверочный оператор «усмменьше» усматривает истинность строгих неравенств. Прием сводит проверку положительности произведения к проверке положительности каждого из сомножителей.

48) Прием нормализатора общей стандартизации.
Теорема приема — тождество, выполняющее простейшую стандартизацию выражений с заданным заголовком. Прием применяется в пакетном нормализаторе общей стандартизации. Дополнительные элементы спецификации — «оператор(A)» и «направл(. . .)». Здесь A — название нормализатора. Прием примеа данного типа:

$$\forall_a(a + 0 = a).$$

Элементы спецификации — «оператор(нормплюс)» и «направл(второйтерм)».

49) Прием синтезатора.
Теорема приема — кванторная импликация, позводяющая подобрать значения переменных, при которых реализуемое пакетным синтезатором утверждение становится истинным. Единственный дополнительный элемент спецификации — «оператор(A)», где A — название синтезатора. Пример приема данного типа:

$$\forall_a(\cos a \leqslant 1).$$

Дополнительный элемент спецификации — «оператор(верхняяоценка)». Данный синтезатор реализует утверждения вида $a \leqslant x$, где a — входное данное, x — выходная переменная. Прием предлагает выбрать единицу в качестве верхней оценки косинуса.

Для каждого типа приема создана программа, инициирующая компиляцию спецификации в описание приема на ГЕНОЛОГе. После инициализации используется цепочка блоков, выполняющих стандартные действия компилятора: уточнение способов обработки антецедентов теоремы приема, уточнение способов идентификации, уточнение способов обработки новых подтермов, создаваемых приемом, уточнение списка дополнительных фильтров приема. Заметим, что создание основных фильтров, предопределяемых типом приема, обеспечивается стартовой программой компилятора. Вообще говоря, результат работы компилятора требует некоторой доводки — примерки на разделе задачника и необходимых коррекций. В первую очередь, здесь уточняется уровень срабатывания приема.

Создание коллекции типов приемов находится лишь на начальной стадии. Часть этих типов можно смело создавать по теореме сразу же, другие — лишь по мере надобности, в процессе решения нестандартных задач. В последнем случае логический ассемблер играет роль «генератора идей». Он может предложить множество потенциально интересных приемов, из числа которых после решения задачи будут отбираться лишь действительно полезные.

Создание спецификации приема по теореме

Как правило, заданный на ГЕНОЛОГе прием имеет своим источником некоторую теорему предметной области. Ее не следует путать с теоремой приема, составляющей часть описания приема. Источником приема служит истинное утверждение, заданное на логическом языке. Теорема приема задается на языке, который является «первым шагом» от логического к алгоритмическому, и вид ее может сильно отличаться от источника. От нее требуется лишь, чтобы она была понятна компилятору ГЕНОЛОГа. Перечислим некоторые возможные различия между источником приема и его теоремой.

Прежде всего, в теореме приема отбрасываются те антецеденты, истинность которых предполагается в контексте по умолчанию — ввиду стандартных условий на области допустимых значений. Так, источник приема:

$$\forall_{ab}(a - \text{число } \&\ b - \text{число} \to -ab = (-a)b)$$

преобразуется в теорему приема:

$$\forall_{ab}(-ab = (-a)b).$$

Далее, в теореме приема могут появиться антецеденты вида «актив(A)», которые означают лишь, что в задаче уже рассматривается выражение A, и имеется явно указывающая на это фиктивная посылка «актив(A)». Пример может служить следующая теорема приема, выводящего соотношение для суммы углов треугольника:

$$\forall_{ABC}(\text{актив}(\angle(ABC))\ \&\ \text{актив}(\angle(BCA)) \to \\ \angle(ABC) + \angle(BCA) + \angle(BAC) = \pi).$$

В некоторых случаях технически проще оказывается компилировать проверку истинности антецедента источника, перенеся его в фильтры приема. Соответственно, из теоремы приема этот антецедент исключается. Если теорема приема обращается к некоторой вспомогательной процедуре — задаче либо пакетному нормализатору, то в ее антецедентах может появиться равенство, организующее данное обращение. В этой ситуации источником приема может оказаться вообще не теорема предметной области, а некоторый технический протокол, указывающий на необходимость решения вспомогательной задачи.

После того, как для приема ГЕНОЛОГа уточнена спецификация, следующим шагом анализа процесса создания этого приема становится указание в базе теорем его источника. Первоначально база теорем возникает не за счет перенесения информации из учебников, а как своеобразная «проекция» базы приемов, извлеченной из задачников.

Таким образом, из базы приемов извлекается обучающий материал для развития процедур, синтезирующих спецификации приемов по заданной теореме. Создание спецификаций начинается с характеризации теоремы — сопровождения ее набором характеристик, указывающих особенности, существенные для алгоритмизации. На текущий момент выявлено более полутора сотен полезных характеристик. Приведем несколько их примеров:

1) «нормализация(n)». Теорема представляет собой тождество, обпспечивающее общую стандартизацию. Логический символ n указывает направление замены. Усмотрение тождества общей стандартизации имеет эвристический характер. При анализе примеров были найдено множество ситуаций, когда тождество естественно относить к данной категории. Приведем несколько таких ситуаций:
   а) В заменяемом терме выделяется подтерм, из которого заменяющий терм получен отбрасыванием части операндов.
   б) Заменяемый терм неконстантный, а заменяющий — константный.
   в) Заменяющий терм бесповторный и имеет единственную переменную. Заменяемый терм тоже содержит эту переменную, но имеет и другие переменные. Эвристическая оценка сложности символов заменяющего терма не больше чем у заменяемого.
   г) Заменяемый и заменяющий термы имеют вид $f(A_1, \ldots, A_n)$ и $g(A_1 \ldots, A_n)$. Операция f не коммутативная, а g — коммутативная.
2) «свертка(n)». Теорема представляет собой тождество, позволяющее переходить к более короткой записи. n — направление замены.
3) «описатель(n)». Теорема представляет собой тождество, которое можно использовать для перехода к более простым описателям либо для исключения описателей. n — направление замены.

4) «дистрибразвертка(n)». Теорема представляет собой обобщенное тождество дистрибутивности (допускается изменение знака внутренней операции).
5) «и(n)» либо «или(n)». Теорема представляет собой эквивалентность, преобразующую элементарное утверждение в конъюнкцию (соответственно, дизъюнкцию) бесквантворных утверждений.
6) «кванторнаясвертка(n)». Теорема представляет собой эквивалентность, позволяющую переходить от квантора к бесквантворному утверждению.
7) «глуб(x n)». Теорема представляет собой эквивалентность, заменяемое утверждение которой элементарно, причем глубина вхождений в него переменной x, с отбрасыванием внешнего отрицания, больше единицы, а аналогичная глубина для элементарного заменяющего утверждения — равна 1. Такие эквивалентности интересны для порождения приемов явного разрешения относительно неизвестной x.
8) «пример». Теорема представляет собой кванторную импликацию без существенных посылок, консеквент которой — элементарное утверждение $f(A_1 \dots A_k)$ либо отрицание такого утверждения. Все A_i, кроме одного, суть попарно различные переменные. На таких теоремах основаны приемы, подбирающие в простейших случаях пример значения неизвестной.
9) «разделение(n)». Теорема представляет собой эквивалентность, позволяющую переносить в разные части двуместного отношения две переменные, ранее расположенные в одной части. n — направление замены.

Источники приемов можно разделить на два класса. В первом случае источник получается из «обычных» теорем предметной области с помощью специализированного логического вывода, причем заранее известен тип приема, ради которого этот вывод реализуется. Тогда характеризация предпринимается процедурами логического вывода. Такой вывод далее называем «программирующим». Примеры программирующих выводов будут рассмотрены в следующем разделе. Во втором случае источник сам является «обычной» теоремой либо близок к ней. Тогда характеризация теоремы предпринимается специальной процедурой, рассматривающей весь спектр возможных направлений синтеза приемов.

После того, как теорема снабжена набором характеристик, начинается собственно создание спецификаций. Оно реализовано как процесс сканирования характеристик и обращения к приемам спецификатора, распределенным между различными названиями характеристик. Эти приемы представляют собой тривиальные переходники от характеристики теоремы к типу приема, хотя иногда учитываются и дополнительные свойства теоремы. Часть спецификаций создаются по теореме сразу — они соответствуют наиболее сильно мотивированным и часто используемым типам приемов. Другие — создаются лишь при наличии специальных опций, уточняющих целевую установку при создании приемов. Например, эти опции могут быть подсказаны задачей, для решения которой запускается синтез дополнительных приемов.

Программирующий логический вывод

Источник приема часто представляет собой теорему, полученную из «обычной» теоремы цепочкой модификаций и обобщений, направленных на расширение области применимости приема. Рассмотрим простой пример. Среди базисных теорем тригонометрии имеется тождество, связывающее между собой значения синуса и косинуса:

$$\forall_a((\sin a)^2 + (\cos a)^2 = 1).$$

Это тождество подсказывает прием общей стандартизации, применяемый слева направо. Чтобы такой прием срабатывал в ситуациях, когда перед квадратами находятся одинаковые коэффициенты, тождество следует несколько обобщить:

$$\forall_{ab}(b(\sin a)^2 + b(\cos a)^2 = b).$$

Следующий шаг обобщения — учет случаев, когда коэффициент b имеет сомножителями некоторые степени синуса и косинуса a:

$$\forall_{abcdef}(c - d = 2 \ \& \ e - b = 2 \rightarrow$$
$$f(\cos a)^c(\sin a)^b + f(\cos a)^d(\sin a)^e = f(\sin a)^b(\cos a)^d).$$

Заметим, что прием обеспечивает усмотрение вырожденных нулевых значений показателей степени b, e. Наконец, если коэффициенты при слагаемых различны, но «подобны», то есть имеют вид pK и qK, где

p, q — десятичные константы одинакового знака, то можно так обобщить тождество, чтобы оно применялось к слагаемому с меньшим по модулю численным коэффициентом p и части другого слагаемого, имеющей такой же численный коэффициент. В результате получаем окончательную версию теоремы, являющуюся источником приема решателя:

$$\forall_{abcdefghijk}(a-b=2\ \&\ c-d=2\ \&\ (0<e\ \&\ 0<f\ \&\ g=\min(e,f)\ \vee$$
$$e<0\ \&\ f<0\ \&\ g=\max(e,f))\ \&\ h=e-g\ \&\ i=f-g\rightarrow$$
$$ej(\cos k)^c(\sin k)^b+fj(\cos k)^d(\sin k)^a=$$
$$hj(\cos k)^c(\sin k)^b+ij(\cos k)^d(\sin k)^a+gj(\cos k)^d(\sin k)^b).$$

Этот пример показывает типичную ситуацию: хотя источник приема и является логически корректным истинным утверждением, он может сильно отличаться от базисной теоремы предметной области за счет ввода большого числа обобщающих «технических» параметров. Иногда таких параметров нет, а источник приема представляет собой просто результат варьирования базисной теоремы с помощью некоторых других ранее освоенных теорем.

Чтобы проследить, каким образом источники приемов возникают из «обычных» теорем, последние тоже заносятся в базу теорем. При этом все источники приемов, полученные из одной и той же обычной теоремы, регистрируются в общем концевом подменю базы теорем. Первый пункт подменю соответствует базисной теореме (иногда она тоже может быть использована как источник приема), остальные — полученным из нее источникам приемов. Такая организация базы теорем превращает ее в задачник для обучения системы программирующему выводу — получению источников приемов из базисных теорем.

Процедура программирующего вывода анализирует список исходных теорем и пополняет его новыми элементами. Исходные теоремы суть базисные тоеремы, извлекаемые из первого пункта некоторого подменю базы теорем. Чтобы получить следствия текущей теоремы списка, последовательно просматриваются ее характеристики, и предпринимаются обращения к программе справочника «прогрвывод» на логическом символе, являющемся заголовком характеристики. Эта программа объединяет некоторую группу приемов вывода теорем.

Для упорядочения применения приемов, как и в случае решателя задач, используются веса теорем. Изначально веса теорем нулевые. После каждой попытки получения следствий теоремы ее вес увеличивается на единицу. На каждом шаге выбирается теорема с минимальным весом v. При ее рассмотрении срабатывают только те приемы, у которых допустимый уровень срабатывания равен v. Цикл вывода следствий обрывается либо по исчерпании возможностей, либо, в особых случаях, по исчерпании лимитов времени или длины списка.

На текущий момент программирующий вывод был проработан лишь для алгебры логики, алгебры множеств и части разделов элементарной алгебры. При этом было создано 212 приемов. Приведем некоторые из них.

1) Обобщение сверточного тождества путем ввода бесповторного дополнительного параметра операнда.
Рассмотрим следующее тождество:

$$\forall_{abc}(a - \text{число } \&\ b - \text{число } \&\ c - \text{число } \&\ 0 < a\ \&\ 0 < b \to a^c b^c = (ab)^c).$$

Одна из его характеристик имеет вид «свертка(второйтерм)», то есть в направлении слева направо тождество может быть использовано для сжатой перезаписи выражения. Если использовать вспомогательное тождество

$$\forall_{acd}(a - \text{число } \&\ c - \text{число } \&\ d - \text{число } \&\ 0 < a \to (a^d)^c = a^{cd}),$$

то нетрудно вывести следствие, обобщающее первое тождество относительно характеристики «свертка(второйтерм)»:

$$\forall_{abcd}(a - \text{число } \&\ b - \text{число } \&\ c - \text{число } \&\ d - \text{число } \&\ 0 < a\ \&\ 0 < b \to a^{cd} b^c = (a^d b)^c).$$

При получении его мы подставили вместо переменной a первого тождества выражение a^d и воспользовались вторым тождеством.
Такой обобщающий переход нетрудно сформулировать в виде следующего приема программирующего вывода. Если в заменяемой части A тождества свертки встречается выражение $f(x,t)$, где x — переменная, не имеющая других вхождений в A, то

при помощи специального справочника поиска теорем находятся всевозможные тождества вида $f(g(u,v),w) = f(u,h(v,w))$, где u,v,w — переменные, h — коммутативно, g имеет единицу по переменной v. Приемы данного справочника создаются заблаговременно, при просмотре базы теорем. В нашем примере роль этого тождества играет повторное возведение в степень. Проверяется, что выражение t не имеет вида $h(\ldots y\ldots)$, где y — переменная, не имеющая других вхождений в A. Затем выбирается новая переменная v, и в обобщаемое тождество вместо x подставляется $g(x,v)$. После применения вспомогательного тождества получаем тождество, у которого в заменяемой части вместо $f(g(x,v),t)$ расположено $f(x,h(v,t))$. Легко видеть, что оно обобщает исходное тождество свертки за счет нового коэффициента v.

2) Обобщение сверточного тождества путем ввода бесповторного внешнего параметра.
Чтобы продолжить обобщение тождества, полученного в предыдущем пункте, достаточно подставить вместо переменной d выражение d/e и использовать следующее вспомогательное тождество:

$$\forall_{cde}(\neg(e=0)\ \&\ c - \text{число}\ \&\ d - \text{число}\ \&\ e - \text{число} \rightarrow c(d/e) = (cd)/e).$$

В результате получим:

$$\forall_{abcde}(\neg(e=0)\ \&\ a - \text{число}\ \&\ b - \text{число}\ \&\ c - \text{число}\ \&\ d - \text{число}\ \&\ e - \text{число}\ \&\ 0 < a\ \&\ 0 < b \rightarrow a^{(cd)/e}b^c = (a^{d/e}b)^c).$$

Этот переход обеспечивается следующим приемом программирующего вывода. Если в заменяемой части A тождества свертки встречается выражение $f(x,t)$, где x — переменная, не имеющая других вхождений в A, то при помощи справочника поиска теорем находятся всевозможные тождества вида $f(g(u,v),w) = h(f(u,w),v)$, где u,v,w — переменные, причем g и h имеют единицу по переменной v. В нашем примере роль такого тождества играет умножение на дробь. Затем выбирается новая переменная v, и в обобщаемое тождество вместо x подставляется $g(x,v)$. После применения вспомогательного тождества и тривиальных упрощений получается тождество, у которого в заменяемой части вместо $f(x,t)$ расположено $h(f(x,t),v)$. Перед

выдачей результата проверяется неизбыточность добавленного параметра v.
Несколько применений описанных двух приемов обобщают рассматриваемое тождество до следующего вида:

$$\forall_{abcdefg}(\neg(e=0)\ \&\ \neg(g=0)\ \&\ a-\text{число}\ \&\ b-\text{число}\ \&$$
$$c-\text{число}\ \&\ d-\text{число}\ \&\ e-\text{число}\ \&\ f-\text{число}\ \&\ g-\text{число}\ \&$$
$$0<a\ \&\ 0<b\to a^{cd/e}b^{cf/g}=(a^{d/e}b^{f/g})^{c}).$$

3) Обобщение сверточного тождества путем ввода повторного дополнительного параметра операнда.
Продолжим цепочку обобщений. Для этого подставим в предыдущее тождество вместо переменной c выражение c/h и используем следующее вспомогательное тождество:

$$\forall_{ade}(\neg(a=0)\ \&\ \neg(e=0)\ \&\ a-\text{число}\ \&\ d-\text{число}\ \&$$
$$e-\text{число}\to(d/a)/e)=d/(ae).$$

Получим следующий результат, который является окончательным и служит источником приема решателя:

$$\forall_{abcdefgh}(\neg(e=0)\ \&\ \neg(g=0)\ \&\ \neg(h=0)\ \&\ a-\text{число}\ \&$$
$$b-\text{число}\ \&\ c-\text{число}\ \&\ d-\text{число}\ \&\ e-\text{число}\ \&$$
$$f-\text{число}\ \&\ g-\text{число}\ \&\ h-\text{число}$$
$$\&\ 0<a\ \&\ 0<b\to a^{cd/(eh)}b^{cf/(gh)}=(a^{d/e}b^{f/g})^{c/h}).$$

Строго говоря, здесь приходится использовать два вспомогательных тождества — кроме тождества деления дроби, нужно также тождество умножения на дробь.
Соответствующий прием программирующего вывода может быть сформулирован, например, следующим образом. В заменяемой части тождества свертки усматривается подвыражение $f(g(x,t),s)$, где x — переменная, имеющая в этой части несколько вхождений, причем каждое — в контексте $f(g(x,p),q)$. Операция g — ассоциативная и коммутативная. При помощи справочников поиска теорем находятся тождества вида $g(h(x,y),z)=u(g(x,z),y)$ и $f(u(x,y),z)=f(x,v(y,z))$. Здесь операция h имеет единицу по переменной y; операция v — ассоциативна и коммутативна. Данные тождества суть аналоги тождеств $(x/y)\cdot z=$

$(x \cdot z)/y$ и $(x/y)/z = x/(yz)$ соответственно. Затем выбирается новая переменная w, в обобщаемое тождество вместо переменной x подставляется $h(x, w)$, и применяются указанные выше вспомогательные тождества. Проверяется неизбыточность нового параметра w.

Заметим, что промежуточные теоремы цепочки обобщений отбрасываются, и в качестве результата выдаются лишь окончательные версии. Кроме перечисленных выше, имеется множество других обобщающих приемов. Приведем теперь несколько приемов программирующего вывода, обеспечивающих варьирование теоремы.

4) Попытка опрокинуть тождество общей стандартизации путем сильного упрощения заменяемого терма.
Рассмотрим тождество для сокращения дробей:

$$\forall_{cdf}(\neg(c = 0)\ \&\ \neg(f = 0)\ \&\ c - \text{число}\ \&\ d - \text{число}\ \&\ f - \text{число} \to (df)/(cf) = d/c).$$

Это тождество имеет характеристику «нормализация(второй-терм)», то есть является тождеством общей стандартизации.
Если воспользоваться другим тождеством общей стандартизации

$$\forall_{ab}(a - \text{число}\ \&\ b - \text{число}\ \&\ \neg(b = 0) \to (a/b)b = a),$$

представляющим собой определение деления, то при унификации числителя df заменяемой части первого тождества с заменяемой частью $(a/b)b$ второго можно вывести следствие:

$$\forall_{abc}(a/(bc) = (a/b)/c).$$

Это следствие тоже является тождеством общей стандартизации, однако направление его применения — обратное.
Прием, реализующий данный вывод, усматривает в заменяемой части тождества общей стандартизации вхождение операции $f(\ldots)$, и обращается к справочнику поиска теорем для обнаружения другого тождества общей стандартизации, обеспечивающего сильное упрощение выражений с заголовком f. Создано несколько справочников такого типа. Например, для поиска тождеств, у которых заменяемая часть имеет имеет несколько

переменных, а заменяющая — не более одной. Затем подтерм $f(\ldots)$ заменямой части первого тождества унифицируется с заменяемой частью второго, и после унификации к нему применяется второе тождество. Проверяется, что результатом служит тождество общей стандартизации, имеющее встречное направление.

5) Попытка проварьировать заменяемую часть тождества общей стандартизации.
Рассмотрим тождество для вынесения наружу минуса из сомножителя:

$$\forall_{ab}(a - \text{число } \& \ b - \text{число} \to -ab = (-a)b).$$

Оно имеет характеристику «нормализация(первыйтерм)». Если воспользоваться вспомогательным тождеством «перегруппировочного» типа:

$$\forall_{ab}(a - \text{число } \& \ b - \text{число} \to -(a - b) = b - a),$$

то нетрудно вывести следующее следствие:

$$\forall_{abc}(a - \text{число } \& \ b - \text{число } \& \ c - \text{число} \to -a(b - c) = a(c - b)).$$

Последнее тождество уже не рассматривается как тождество общей стандартизации, а имеет характеристику «свертка(второйтерм)». Оно используется для сокращенной перезаписи выражения при редактировании ответа задачи. Прием, реализующий данный вывод, аналогичен предыдущему. Отличие заключается лишь в использовании других справочников поиска теорем. Они отбирают тождества, у которых обе части имеют одни и те же переменные, причем каждая переменная — бесповторная. Не обязательно, чтобы такое тождество обеспечивало общую стандартизацию.

6) Извлечение из декомпозирующей эквивалентности импликации для проверочного оператора.
Рассмотрим определение принадлежности объединению множеств:

$$\forall_{abc}(b - \text{set } \& \ c - \text{set} \to a \in b \cup c \leftrightarrow a \in b \ \vee \ a \in c).$$

Эта эквивалентность имеет характеристику «или(второйтерм)», то есть представляет собой декомпозицию элементарного утверждения в дизъюнкцию, построенную с помощью логических связок из элементарных утверждений. Она позволяет получить следующие два следствия:

$$\forall_{abc}(b - \text{set} \ \& \ c - \text{set} \ \& \ a \in b \to a \in b \cup c),$$
$$\forall_{abc}(b - \text{set} \ \& \ c - \text{set} \ \& \ \neg(a \in b) \ \& \ \neg(a \in c) \to \neg(a \in (b \cup c))).$$

Первое из них может быть использовано в проверочном операторе, усматривающем принадлежность, второе — в операторе, усматривающем непринадлежность.

Данный пример приводит к приему программирующего вывода, обрабатывающему эквивалентность вида $A \leftrightarrow (B_1 \vee \ldots \vee B_n)$, где A — элементарное утверждение. Действия приема распадаются на две части. Первая часть приводит дизъюнкцию $(B_1 \vee \ldots \vee B_n)$ к виду дизъюнкции конъюнкций элементарноых утверждений, и для каждой конъюнкции C выводит следствие $C \to A$. Предварительно проверяется, что существует проверочный оператор, обрабатывающий утверждения вида A. Вторая часть — приводит к виду дизъюнкции конъюнкций элементарных утверждений отрицание утверждения $(B_1 \vee \ldots \vee B_n)$, и для каждой конъюнкции C выводит следствие $C \to \neg(A)$. Как и выше, предварительно устанавливается существование проверочного оператора, обрабатывающего утверждения вида $\neg(A)$.

7) Вывод из определения класса условия принадлежности классу. Рассмотрим определение образа множества:

$$\forall_{af}(a - \text{set} \ \& \ f - \text{функция} \ \& \ a \subseteq \text{Dom}(f) \to$$
$$\text{образ}(f, a) = \text{set}_x(\exists_y(y \in a \ \& \ x = f(y)))).$$

Это тождество имеет характеристику «описатель(первыйтерм)», так как оно может быть использовано для исключения описателя «класс». В обычных ситуациях тождество применяется в направлении «справа налево». Однако, для условия принадлежности образу бывают нужны приемы, выполняющие обратное преобразование. Они основаны на следующем тривиальном следствии исходного тождества:

$$\forall_{abf}(f - \text{функция} \ \& \ b - \text{set} \ \& \ b \subseteq \text{Dom}(f) \to$$
$$a \in \text{образ}(f, b) \leftrightarrow \exists_x(a = f(x) \ \& \ x \in b)).$$

Прием программирующего вывода здесь просто преобразует тождество
$B = \mathrm{set}_x A(x)$ в эквивалентность $x \in B \leftrightarrow A(x)$.

8) Разбиение теоремы с условным выражением на две теоремы для подслучаев.
Рассмотрим определение модуля:

$$\forall_a(a - \text{число} \to |a| = (a \text{ при } 0 \leqslant a, \text{ иначе } -a)).$$

Для двух различных альтернатив условного выражения выводим следующие два следствия:

$$\forall_a(a - \text{число } \& \ 0 \leqslant a \to |a| = a),$$
$$\forall_a(a - \text{число } \& \ a < 0 \to |a| = -a).$$

Из этого примера легко извлекается прием программирующего вывода, применяемый к произвольной теореме с условным выражением в консеквенте.

9) Использование тождества общей стандартизации для получения тождества, устраняющего сложное подвыражение с неизвестными при навешивании внешней операции.
Для исключения неизвестных радикалов путем возведения обеих частей уравнения в некоторую степень используется тождество:

$$\forall_{abcd}(\neg(c = 0) \ \& \ a - \text{число } \& \ b - \text{число } \& \ c - \text{число } \&$$
$$d - \text{число } \& \ 0 \leqslant a \ \&0 \leqslant b \to (ba^{d/c})^c = b^c a^d).$$

Оно сопровождается в базе теорем характеристикой «смнеизв(...)», указывающей, что было получено именно для этой цели. В ней уточняется контекст применения тождества: преобразуется подтерм условия задачи на описание, выражение a содержит неизвестные, d — натуральная константа, выражение c не содержит неизвестных, причем либо b тоже не содержит неизвестных, либо c — натуральная константа. Рассматриваемое тождество получается обобщением относительно данной характеристики следующего более простого тождества, которое, собственно, и будет представлять для нас интерес в этом пункте:

$$\forall_{acd}(\neg(c = 0) \ \& \ a - \text{число } \& \ c - \text{число } \& \ d - \text{число } \&$$
$$0 \leqslant a \to (a^{d/c})^c = a^d).$$

Оно легко получается из тождества последовательного возведения в степень

$$\forall_{abc}(a - \text{число } \& \ b - \text{число } \& \ c - \text{число } \& \ 0 \leqslant a \to (a^b)^c = a^{bc})$$

с помощью вспомогательного тождества

$$\forall_{ab}(a - \text{число } \& \ b - \text{число } \& \ \neg(b = 0) \to (a/b)b = a).$$

Прием программирующего вывода, выполняющий этот шаг, анализирует подтерм заменяющего терма тождества общей стандартизации (в нашем случае — выражение bc) и обращается к справочнику поиска теорем для нахождения сокращающего тождества, применимого к данному подтерму. Под сокращающим понимается тождество вида $f(x, g(x, y)) = a$, где a — константа либо переменная. Далее находится результат последовательного применения двух тождеств, и проверяется, что он может быть использован для исключения сложного выражения с неизвестными при навешивании на обе части уравнения некоторой внешней операции. Таким образом и появляется изначально характеристика «смнеизв(...)». Заметим, что никаких других характеристик выведенному тождеству не присваивается, и далее оно будет использоваться только по своему прямому назначению.

База теорем предоставляет обучающий материал не только для программирующего вывода, извлекающего источники теорем из базисных теорем раздела, но и для «исследовательского вывода», объясняющего получение одних базисных теорем из других. Граница между программирующим и исследовательским выводами весьма условная. Рассмотрим несколько приемов, которые могут быть отнесены к любой из этих категорий.

10) Вывод упрощающего тождества в стандартной форме путем унификации заменяемых частей двух ранее найденных тождеств.
Во многих разделах математики упрощение выражений связано с использованием многочленоподобных «стандартных форм». Например, таковы дизъюнктивные и конъюнктивные нормальные формы в алгебре логики и их аналоги в алгебре множеств. Преимущества, которые создает переход к стандартной форме,

заключаются в уменьшении расстояний между ее подвыражениями: из любой точки к любой другой можно перейти всего за три шага — два раза через аналог «умножения» и один раз через аналог «сложения». В результате существенно повышается эффективность использования упрощающих тождеств и уменьшается их количество. Чтобы порождать новые такие тождества, задается список базисных тождеств, по которым сразу генерируются приемы вспомогательного пакетного нормализатора N. Этот нормализатор используется только в цикле вывода новых тождеств. Он будет упрощать обе части каждого выводимого тождества, чтобы исключить избыточные следствия. Основной шаг вывода — унификация заменяемых частей двух упрощающих тождеств, позволяющая применить их последовательно — сначала первое тождество применяется в направлении усложнения, затем второе — в направлении упрощения. Данное правило вывода хорошо известно в математической логике. С его помощью можно быстро создавать аппарат упрощения выражений в различных алгебраических системах. Преобразование тождеств в приемы, ввиду тривиальной целевой установки, здесь не вызывает затруднений.

В качестве примера рассмотрим тождества $ac \vee bc\overline{a} = ac \vee bc$ и $p \vee pq = q$ из алгебры логики, используемые для упрощения дизъюнктивных нормальных форм. Если проводить унификацию заменяемых частей, вводя вспомогательные переменные для дополнительных дизъюнктивных членов, то в качестве одного из вариантов получим выражение $ac \vee bc\overline{a} \vee b\overline{a}$, При этом заменяющие части обоих тождеств равны, соответственно, $ac \vee bc \vee b\overline{a}$ и $ac \vee b\overline{a}$. Отсюда выводится тождество обобщенного склеивания: $ac \vee bc \vee b\overline{a} = ac \vee b\overline{a}$.

11) Попытка склеить два члена факторизуемого выражения.
Иногда возникает задача преобразовать выражение к такому виду, у которого заголовком служит заданный логический символ. Например, задача разложения на множители. Для вывода тождеств, обеспечивающих приведение к заданным заголовкам, используется процедура, аналогичная описанной в предыдущем пункте. Сначала порождаются простейшие тождества данного типа. В случае разложения на множители для этого берется произведение двух или более сумм различных переменных, и предпринимается раскрывание скобок. На период вывода

по найденным тождествам создается нормализатор приведения к заданным заголовкам, используемый для отбрасывания тождеств, сводящихся к ранее полученным. Основной шаг вывода — применение к нескольким членам заменяемой части текущего тождества вспомогательного тождества, обеспечивающего взаимное уничтожение этих членов. В результате промежуточные члены, необходимые для непосредственного разложения, оказываются «спрятаны», и возникает ценное новое тождество.
В качестве примера рассмотрим тождество $(a + b)(c + d) = ac + bc + ad + bd$. При использовании вспомогательного тождества $p - p = 0$ можно унифицировать ad с p, bc - с $-p$ и получить следствие $(a + b)(a - b) = a^2 - b^2$. Таким же образом порождаются остальные тождества сокращенного умножения. Кроме них, удается получить множество других полезных для ускоренного разложения на множители формул. Обнаруживаются и такие экзотические соотношения, как $a^5 - b^5 - ba^4 = (a^2 + b^2 - ab)(a^3 - b^3 - ab^2)$.

12) Попытка проварьировать эквивалентность, разрешающую относительно неизвестной, с помощью тождества, создающего кратные вхождения переменной.
Прием рассматривает эквивалентность, разрешающую относительно неизвестной x некоторое утверждение A с единственным вхождением этой неизвестной. Внутри A выбирается содержащее x подвыражение t, имеющее самый сложный в эвристическом смысле заголовок. Справочник поиска теорем перечисляет тождества, позволяющие преобразовать t в выражение t' с несколькими вхождениями переменной x. В исходной эквивалентности предпринимается замена t на t', и после несложной стандартизации выдается результат. Цель данного вывода — получить эквивалентность, разрешающую, хотя бы в частном случае, утверждение с несколькими вхождениями неизвесетной. Обычно она подлежит цепочке обобщений.
В качестве примера возьмем эквивалентность для разрешения простейшего степенного уравнения, имеющего рациональный показатель степени с четным числителем:

$$\forall_{abc}(a - \text{число } \& \ b - \text{число } \& \ c - \text{число } \& \ b - \text{rational } \& \\ \text{числитель}(b) - \text{even } \& \neg(b = 0) \rightarrow \\ a^b = c \leftrightarrow 0 \leqslant c \ \& \ (a = c^{1/b} \ \vee \ a = -c^{1/b})).$$

Вспомогательным тождеством будет тождество для квадрата суммы

$$\forall_{ab}(a - \text{число } \& \ b - \text{число} \to (a+b)^2 = a^2 + 2ab + b^2),$$

позволяющее перейти от выражения $(a+b)^2$ с бесповторными переменными к повторным переменным. После группировки в правой части всех известных слагаемых, получим:

$$\forall_{cde}(c - \text{число } \& \ d - \text{число } \& \ e - \text{число} \to d^2 + 2de = c - e^2 \leftrightarrow$$
$$((d = -e + \sqrt{c} \ \vee \ d = -(e + \sqrt{c})) \ \& \ 0 \leqslant c)).$$

Роль неизвестной здесь играет переменная d. Чтобы получить из этой эквивалентности известную формулу для решения квадратного уравнения, далее можно использовать прием программирующего вывода, усматривающий подвыражение уравнения, не содержащее неизвестных, и обозначающий его новым параметром. На первом шаге, вводя вспомогательный параметр a для выражения в правой части уравнения и разрешая относительно c уравнение $c - e^2 = a$, получим:

$$\forall_{ade}(a - \text{число } \& \ d - \text{число } \& \ e - \text{число} \to d^2 + 2de = a \leftrightarrow$$
$$((d = -e + \sqrt{a + e^2} \ \vee \ d = -(e + \sqrt{a + e^2})) \ \& \ 0 \leqslant a + e^2)).$$

Еще раз применяем этот же прием, обозначая посредством b подвыражение $2e$ и разрешая относительно e уравнение $2e = b$. Получаем:

$$\forall_{abd}(a - \text{число } \& \ b - \text{число } \& \ d - \text{число} \to$$
$$d^2 + bd = a \leftrightarrow ((d = -(b + \sqrt{b^2 + 4a})/2 \vee$$
$$d = (-b + \sqrt{b^2 + 4a}/2)) \ \& \ 0 \leqslant b^2 + 4a)).$$

Опускаем описание приемов программирующего вывода, доводящих обобщение данной формулы до случая произвольного коэффициента перед квадратом неизвестной. Заметим, что на всех этапах обобщения теорема сопровождалась характеристикой, указывающей неизвестную d.

В заключение приведем прием вывода теорем, позволяющий системе «открывать» формулу Кардано.

13) Попытка использовать оператор усмотрения повторяющихся подвыражений для получения импликации, подбирающей корень уравнения.
Прием выводит теорему, позволяющую подобрать значение неизвестной, имеющей кратные вхождения в реализуемое условие. Он анализирует исходную теорему вида $\forall_x(A(x) \to B(x))$ и пытается таким образом преобразовать утверждение $B(x)$, чтобы какое-либо неконстантное подвыражение t встречалось в нем более одного раза. Далее выбирается новая переменная y, находится результат $C(x,y)$ замены всех вхождений t в преобразованное $B(x)$ на переменную y, и выводится следствие $\forall_x(A(x)\ \&\ y = t \to C(x,y))$. Роль неизвестной здесь играет переменная y, причем t — подбираемое значение. Это отражено в характеристике, сопровождающей теорему. Далее реализуется цепочка обобщений.
В качестве примера рассмотрим тождество для куба суммы:

$$\forall_{ab}(a - \text{число}\ \&\ b - \text{число} \to (a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3).$$

Консеквент $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3)$ нетрудно привести к виду

$$(a+b)^3 = a^3 + 3ab(a+b) + b^3,$$

используя для этого вспомогательный пакетный нормализатор, выделяющий «скрытые» повторные вхождения одного и того же выражения. Подвыражение $a+b$ здесь встречается дважды, и для вспомогательной переменнй y получаем следствие:

$$\forall_{aby}(a - \text{число}\ \&\ b - \text{число}\ \&\ y = a+b \to y^3 - 3aby = a^3 + b^3).$$

Здесь все неизвестные члены сгруппированы слева, а известные — справа. Далее реализуется цепочка обобщений, основанных на том же принципе, что и для квадратного уравнения: выбирается подвыражение без неизвестных, которое обозначается вспомогательным параметром. Прежде всего, для $-3ab$ вводится параметр c. Подставляемое вместо a выражение определяется из уравнения $-3ab = c$, и получаем первый шаг цепочки обобщений:

$$\forall_{bcy}(b - \text{число}\ \&\ \neg(b=0)\ \&\ c - \text{число}\ \&$$
$$y = -c/(3b) + b \to y^3 + cy = -c^3/(27b^3) + b^3).$$

На следующем шаге — вводим для выражения $-c^3/(27b^3)+b^3$ вспомогательный параметр a. Уравнение $-c^3/(27b^3)+b^3=a$ легко решается, так как преобразуется к квадратному относительно b^3. В результате получаем формулу корня кубического уравнения, не имеющего члена с квадратом неизвестной:

$$\forall_{abcy}(c - \text{число } \&\ \neg(c=0)\ \&$$
$$a - \text{число } \&\ 0 \leqslant 4c^3+27a^2\ \&\ y=-c/(3b)+b\ \&$$
$$b=\sqrt[3]{3\sqrt{3}a-\sqrt{4c^3+27a^2}}/(\sqrt[3]{2}\sqrt{3}) \to y^3+cy=a).$$

Для остальных шагов цепочки обобщений, связанных с добавлением коэффициента при y^3 и члена с y^2, создание приемов вывода не составляет труда.

При обучении решателя «вручную» обычно создаются лишь те приемы, которые необходимы для текущей задачи. Множество аналогичных приемов, которые понадобились бы в близких задачах, при этом пропускаются. Именно их отсутствием (а вовсе не переборными трудностями) обычно и объясняются отказы решателя. Процедура программирующего вывода позволяет восполнять такие недостачи и приносит большую пользу даже как дополнительный элемент среды «ручного» программирования. С помощью весьма скромного аппарата вывода теорем, созданного на текущий момент, в решатель были занесены многие десятки полезных приемов.

Схема алгоритмизации предметной области

При развитии решателя была создана целая пирамида языков — ЛОС, ГЕНОЛОГ, логический ассемблер. Видимо, для задания одного отдельного приема уровень логического ассемблера — наивысший возможный. Однако, возникает потребность в языке еще более высокого уровня — для задания неких общих эвристических принципов алгоритмизации предметной области как единого целого, своего рода ее «схемы алгоритмизации». Здесь можно было бы фиксировать решения о создании в предметной области различных пакетных операторов и их использовании, решения о специализированной контекстной стандартизации термов и т. п. В настоящее время такие решения

фиксируются с помощью термов технического характера, помещаемых в базу теорем — так называемых «протоколов базы теорем». Приведем несколько примеров протоколов.

1) Указание контекстной стандартизации: числители и знаменатели дробных выражений в условии задачи на преобразовании разлагаются на множители.
   Имеется несколько таких протоколов, уточняющих различные контексты. Каждый из них задает пакетный нормализатор, с помощью которого реализуется попытка разложения на множители. Эти протоколы являются источниками приемов, выполняющих стандартизацию. Анализ связанных с дробями тождеств легко объясняет ее происхождение: подготовка возможности сокращения дробных выражений, а также упрощение сложения таких выражений при усмотрении общих множителей в знаменателях либо в числителях. Обобщая данную ситуацию, можно создать прием для процедуры, планирующей схему алгоритмизации.
2) Указание контекстной стандартизации: раскрывание скобок под тригонометрическими операциями.
   Для каждой тригонометрической операции создан свой протокол. Он служит источником приема, обращающегося к специальному нормализатору раскрывания скобок, если усматривается возможность преобразовать операнд тригонометрической операции к виду суммы. Данный протокол учитывается в других приемах, чтобы заблокировать обратное преобразование. Происхождение рассматриваемой стандартизации очевидно: существуют тождества для тригонометрических операций от суммы либо разности аргументов, и не существует тождеств для аргументов — произведений либо степеней.
3) Указание контекстной стандартизации: преобразование логарифмов к общему основанию.
   Протокол определяет создание приемов, преобразующих в различных ситуациях логарифмы к общему основанию. Источником этих приемов служит тождество для перехода к новому основанию, а спецификатор, создающий по нему приемы, учитывает существование протокола. Мотивировка ввода протокола планировщиком схемы алгоритмизации тривиальна: большинство имеющихся базисных тождеств относятся к логарифмам с одинаковыми основаниями.

4) Решение о создании в алгебре множеств стандартной формы «объединение пересечений».
   Протокол фиксирует факт создания нормализатора стандартной формы, преобразующего выражения алгебры множеств к виду объединения пересечений либо разностей. Этот протокол служит источником приемов, предпринимающих попытку упростить выражение алгебры множеств с помощью приведения к данной стандартной формы, использования ее упрощающих тождеств и последующей свертки. Приемы различаются точкой привязки — той операцией алгебры множеств, при усмотрении которой выполняется попытка.
5) Решение о создании нормализатора вычисления: регистрация всего аппарата формального интегрирования в рамках пакетного нормализатора.
   Протокол служит источником приема, обращающегося к данному нормализатору для вычисления неопределенного интеграла.

Автоматическая доводка приемов по задачнику

Чтобы обеспечить разумное взаимодействие нового приема с ранее созданными приемами, необходим анализ его срабатываний на задачах и определенная коррекция — «доводка». В первую очередь, должен быть уточнен уровень срабатывания приема.

Необходимая для доводки информация появляется, как только находятся задачи, в которых срабатывание приема является полезным. Программа доводчика подбирает по этим задачам уровень срабатывания, минимизируя суммарное время их решения. Сохраняется информация о контекстах срабатывания приема в данных задачах; такие контексты называются «позитивными». Затем реализуется цикл решения всех задач из тех разделов задачника, в которых возможно срабатывание приема. Находятся задачи, у которых после создания приема решение оказалось утраченным либо ощутимо замедлилось. Уточняется информация о контекстах срабатывания приема в последних задачах; эти контексты называются «негативными». После выявления списков позитивных и негативных контекстов срабатывания приема, доводчик предпринимает попытку перейти к подтипам данного типа приема, чтобы отсечь его срабатывания во всех нега-

тивных случаях, сохранив срабатывания во всех позитивных. Иногда приходится разбивать один прием на несколько, имеющих различные типы и уровни срабатывания.

Помимо разделения позитивных и негативных контекстов, доводчик также анализирует замедление «холостого хода», то есть замедление решения тех задач, где прием вообще не срабатывал. Если идентификация приема трудоемка, то замедление холостого хода может оказаться значительным, и тогда доводчик увеличивает уровень срабатывания приема.

Программа доводчика развивалась для доводки приемов решения планиметрических задач, причем оказалась способной воссоздавать приемы, созданные вручную, иногда даже несколько улучшая их качество.

Генератор приемов

Перечисленные выше блоки программирующего вывода, характеризации, создания спецификации приема, компиляции спецификаций, планирования алгоритмизации предметной области и доводки приема позволяют проследить весь процесс перехода от базисной теоремы к приему, и в этом смысле образуют аппарат автоматического синтеза приемов. Возможны следующие способы применения данного аппарата (генератора приемов) в процессе самообучения логической системы:

1) Синтез приемов при первом ознакомлении с теоремой.
   Применяется программирующий вывод, позволяющий сопоставить новую теорему с ранее известными и получить полезные ее следствия. В некоторых случаях они уже будут сопровождены характеристиками, в других — понадобится процедура характеризации. Далее выполняются синтез спецификаций приемов и компиляция их в ГЕНОЛОГовские описания приемов.
   По такой схеме будут синтезироваться лишь наиболее естественные и хорошо мотивированные приемы. Для их отбора потребуются эвристические ограничители. Обычно в учебниках новые теоремы сопровождаются простыми упражнениями, которые можно использовать для калибровки уровней срабатывания введенных указанным образом приемов.
2) Синтез приемов при решении нестандартной задачи.

Если попытка решить задачу «в лоб» либо со стандартными средствами усиления оказалась неудачной, то предпринимается повторная попытка, в которой разрешается синтезировать дополнительные приемы. Вводятся специальные приемы решателя, усматривающие в текущем контексте целесообразность создания дополнительных приемов заданного типа. При помощи соответствующих справочников поиска теорем (либо при непосредственном просмотре оглавления теорем) отбираются нужные теоремы, для которых либо сразу создаются приемы искомого типа, либо предварительно предпринимается программирующий вывод, ориентированный на потребности решаемой задачи. После синтеза приемов попытка решения повторяется. Если после нескольких обращений к генератору приемов задачу удается решить, то отбираются фактически использованные новые приемы, которые оптимизируются доводчиком и сохраняются в базе приемов. Остальные приемы удаляются.

3) Синтез приемов, подсказанный усилителем.
Если после неудачной попытки решения задачи «в лоб» было получено ее решение с помощью стандартного усилителя (то есть без синтеза дополнительных приемов), то анализируется протокол решения задачи. В нем рассматриваются точки, где последовательное применение нескольких слабо мотивированных шагов (например, при локальном переборе) дало сильно мотивированное действие. В базе теорем находятся теоремы, определяющие указанные слабо мотивированные шаги, и реализуется программирующий вывод, воспроизводящий получение сильно мотивированного шага. Найденная теорема представляет собой искусственную «заготовку», которая может оказаться полезной в будущем. По ней создается новый прием, который оптимизируется доводчиком.

Более подробному описанию логической системы «Искра» посвящена монография «Компьютерное моделирование логических процессов». В первом томе этой монографии [1] описываются архитектура системы, а также языки ЛОС и ГЕНОЛОГ. В следующих трех томах будут приведены приемы решателя, созданные при обучении его в таких разделах, как дискретная математика, теория множеств, элементарная алгебра, математический анализ, дифференциальные уравнения, элементарная геометрия, аналитическая геометрия, ли-

нейная алгебра, комплексный анализ, теория вероятностей, элементарная физика, элементарная химия, вычисления, понимание естественного языка, шахматы и анализ рисунков. В настоящее время подготовлены первые два из этих трех томов. Наконец, в пятом томе монографии предполагается описать блоки генератора приемов.

Логическая система возникала и развивалась исключительно за счет анализа примеров. В основе ее лежит база приемов решателя, извлекаемая в процессе обучения из разнообразных задачников. Эта база приемов послужила исходным материалом для выявления архитектуры генератора приемов и его обучения. Если какие-то технические решения оказывались малоэффективными при работе с примерами, то они отбрасывались; работоспособные версии — сохранялись. В этом смысле, компьютерное моделирование сыграло роль «микроскопа» для изучения мира логических процессов. Продолжение обучения логической системы на всех ее уровнях позволит уточнить найденные архитектурные решения и вплотную подойти к созданию искусственного интеллекта.

Автор выражает искреннюю благодарность В. Б. Кудрявцеву, поддержка которого сделала возможным проведение данного исследования.

Список литературы

[1] Подколзин А. С. Компьютерное моделирование логических процессов. Архитектура и языки решателя задач. — М.: Физматлит, 2008.