О формализации приемов решения математических задач

А.С. Подколзин

Можно выделить, по-видимому, три основных разновидности знаний, используемых в математике для решения задач: аксиомы и теоремы, обеспечивающие потребности логического вывода в процессе решения; алгоритмы, представляющие собой готовые планы действий для решения задач специальных типов, а также эвристические приемы, выполняющие при отсутствии необходимого алгоритма планирование действий непосредственно в процессе решения задачи. Формализация первых двух типов знаний успешно осуществлена в рамках математической логики и теории алгоритмов, и для возникших здесь математических моделей были получены глубокие результаты в таких важных направлениях, как исследование свойств аксиоматических систем, вопросы алгоритмической разрешимости массовых проблем и вопросы сложности вычислений. К сожалению, иначе обстоит дело с формализацией применяемых при решении задач процедур планирования действий, оказывающихся необходимыми даже при небольшом отклонении задачи от "стандартных"ситуаций, обеспеченных алгоритмическими заготовками. Хотя такие процедуры в конкретных предметных областях давно и хорошо известны и без труда передаются при обучении человека, получение их явного формального описания оказалось весьма сложной проблемой, на пути решения которой сегодня сделаны лишь первые шаги. В отличие от алгоритма решения задачи, гарантированным образом приводящего к получению ответа для любых исходных данных из заданного класса, прием решения задачи выполняет лишь некоторые промежуточные преобразования текущих структур данных, и итоговая целесообразность его действий всецело зависит от действий других приемов, участвующих в общем процессе решения. Это означает, что проведение какой-либо оптимизации приема, или хотя бы получение качественных или количественных характеристик его эффективности, является невозможным в отрыве от рассмотрения всего многообразия приемов, с которыми данный прием взаимодействует при решении задач. Даже само описание приема, принимающего решение о целесообразности тех или иных действий с

учетом возможных их последствий, уже предполагает наличие весьма детальной информации о поведении других приемов, и небольшие изменения в решающих правилах одного приема либо добавление нового приема могут повлечь за собой необходимость значительной коррекции многих других приемов. Математическое исследование динамики взаимодействия сотен и тысяч приемов, используемых в реальных предметных областях, наталкивается на серьезные трудности, типичные для динамики больших и сверхбольших систем. Более перспективным, а возможно, и единственно доступным, представляется подход к формализации приемов путем создания компьютерных моделей логических процессов, реализуемых человеком при решении задач, и развитии формальных языков высокого уровня, позволяющих не только адекватно сформулировать выполняемые приемом действия, но и создающих предпосылки для объяснения механизмов возникновения приемов и их автоматического синтеза. Ввиду большой сложности моделируемых процессов, особо важным становится при этом развитие мощного технологического комплекса, обслуживающего процесс обучения системы.

В настоящей работе описывается компьютерная система решения математических задач, созданная в рамках указанного подхода и позволившая в весьма наглядной и компактной форме получить формальные описания приемов решения задач для ряда областей элементарной математики и математического анализа.

Используемая в системе модель процесса решения задачи содержит следующие основные компоненты:

1. Логический язык, используемый для формулировки задач. Этот язык представляет собой открытую версию языка логики предикатов, развиваемую в направлении приближения к естественному математическому языку. Правильно построенные выражения языка имеют обычную скобочную структуру $f(x_1, \ldots, x_n)$, причем для каждого символа f определяются свои собственные правила формирования допустимых выражений и вводится их семантика. Так как утверждения и выражения данного языка используются в контексте решаемой задачи, появляется возможность семантику их так же определять в контексте этой задачи, и таким образом использовать ряд стандартных математических обозначений (например, $o(x)$ и $O(x)$), недопустимых в классической логике предикатов. В системе предусмотрена возможность вводить математические формулы в обычной записи, транслируя их затем в указанный "внутренний"логический язык, а также просматривать в обычной записи формулы, возникающие в процессе решения задачи. Эта возможность является исключительно важной для обеспечения интенсивного диалогового режима обучения системы.

2. Представление задач в системе. Задачи делятся на 4 типа: задачи

на доказательство, на преобразование, на описание и на исследование. Задача Z представляется набором $(A_0, \ldots, A_n)$, где A_0 - логический символ, кодирующий тип задачи. Вне зависимости от типа задачи элементы A_1 - A_3 задают следующие объекты:

а) A_1 есть набор утверждений $(f_1, \ldots, f_m)$, называемых посылками задачи. Эти утверждения предполагаются при решении задачи истинными; фактически они задают исходные условия на "известные"объекты задачи.

б) A_2 есть набор $(v_1, \ldots, v_m)$ целых неотрицательных чисел, называемых весами посылок задачи и используемых, как это указано ниже, для переключения внимания в процессе решения задачи. Изначально все веса автоматически устанавливаются на 0.

в) A_3 есть набор $(K_1, \ldots, K_m, K_0)$. Элементы $K_1, \ldots, K_m$ этого набора суть наборы комментариев, вводимых системой к посылкам $f_1, \ldots, f_m$ в процессе решения; элемент K_0 есть набор комментариев, вводимых системой ко всему списку посылок в целом. Комментарии используются для сохранения информации, необходимой для принятия решений, а также связывают логические структуры данных задачи с сопровождающими их сетевыми структурами данных, что ускоряет поиск объектов, находящихся в заданных отношениях с объектами, упоминаемыми в текущем утверждении задачи.

В случае задачи на доказательство $n = 6$, причем A_4 есть утверждение, истинность которого требуется доказать в предположении истинности посылок (оно называется условием задачи); A_5 - вес условия (используемый, как и веса посылок, для управления переключением внимания); A_6 - список комментариев к задаче. Если условие задачи на доказательство является следствием ее посылок, то ответом данной задачи является логический символ "истина в противном случае ответ не определен.

В случае задачи на преобразование $n = 7$, причем A_4 есть выражение, которое требуется тождественным образом преобразовать в предположении истинности посылок (как и в случае задачи на доказательство, оно называется условием задачи); A_5 есть вес условия; A_6 - список комментариев к задаче. Новым элементом является A_7 - список целей задачи. Этот список необходим для уточнения того вида, к которому требуется преобразовать условие задачи, а также для определения действующих в процессе решения установок на оптимизацию. Фактически, список целей задачи является набором дополнительных входных параметров процедуры ее решения, позволяющим определенным образом управлять поведением всего многообразия используемых при решении приемов.

В случае задачи на описание $n = 8$; A_4 есть список $(g_1, \ldots, g_k)$ утверждений, представляющих собой условия на значения неизвестных

задачи. При решении задачи требуется получить полное либо частичное (в зависимости от списка целей) описание значений неизвестных, удовлетворяющих данным условиям. A_5 есть список весов утверждений списка A_4 (эти утверждения называем условиями задачи на описание); A_6 - список $(Q_1, \ldots, Q_k, Q_0)$ наборов, первые k из которых суть списки комментариев к условиям, а последний - список общих комментариев ко всей задаче в целом. Элемент A_7 является списком целей задачи. Как и в случае задачи на преобразование, он определяет требования, предъявляемые к виду результирующего описания, и задает оптимизационные тенденции, учитываемые при решении; в частности, список целей выделяет переменные задачи, являющиеся неизвестными. При решении задач на описание часто приходится извлекать совместные следствия из ее посылок и условий. Такие следствия либо могут явным образом задавать значения неизвестных (как это часто оказывается, например, при решении систем уравнений), либо содержать существенную для принятия решений информацию. К сожалению, регистрация этих следствий непосредственно в списке условий задачи противоречит общей тенденции к постепенному "упрощению"этого списка и преобразованию его в окончательный ответ задачи. Многочисленные следствия, возникающие при анализе задачи, в случае присоединения их к списку условий, потребуют, даже при получении равенств, явным образом задающих значения неизвестных, проведения весьма трудоемкой проверки их истинности. По этой причине приходится вводить для регистрации указанных следствий специальный накопитель - элемент A_8, который технически оформлен в виде независимой задачи на исследование (см. ниже). Список посылок последней изначально формируется как объединение списков посылок и условий задачи на описание, а список целей определяет такую направленность логического вывода в посылках, которая обслуживала бы потребности "основной"задачи на описание.

Наконец, в случае задачи на исследование $n = 4$. В этом случае A_4 - список целей задачи. Решение задачи на исследование представляет собой процесс логического вывода в ее списке посылок и эквивалентных преобразований этих посылок, регламентируемый ограничениями, содержащимися в списке целей - своего рода "логическое замыкание"исходного описания некоторой ситуации. Основная роль, которую играют задачи такого типа в решателе - обеспечение указанного выше анализа условий в задачах на описание.

3. Общая схема хранения приемов и активизации их в процессе решения задачи. В системе используется процедура сканирования задачи, которую можно представлять как своего рода модель внутреннего логического зрения. Для большинства приемов активизация их при рассмотрении задачи начинается с

обнаружения в логических структурах данных некоторого логического символа, заранее выбранного для этого приема. В качестве такого "ключевого"логического символа берется обычно некоторый логический символ, появление которого необходимо для возможности применения рассматриваемого приема, причем при нескольких возможных выборах предпочтение отдается наиболее редко встречающемуся символу. Указанное закрепление за приемами логических символов предопределяет организацию всей базы приемов решателя по принципу энциклопедии: она распадается на группы приемов, "принадлежащих"соответствующим символам, причем каждая группа реализуется в виде отдельной алгоритмической процедуры A_f - программы логического символа f. В особых случаях прием не удается связать с каким-либо конкретным логическим символом, появление которого является необходимым для срабатывания. Такие приемы распределены по четырем логическим символам - указанным выше названиям типов задач, при решении которых возможно их применение.

При сканировании задачи используются два вспомогательных параметра - текущий уровень сканирования u и максимальный уровень U. Эти уровни введены для учета приоритетов при срабатывании приемов; они помогают организовать отбор приемов таким образом, чтобы в текущей ситуации система выбирала прием, наиболее целесообразный с точки зрения обучающего ее эксперта. Значениями параметров u, U служат целые неотрицательные числа из сравнительно небольшого диапазона. Изначально значение u устанавливается на 0; величина U определяется перед обращением к решению задачи и указывает на тот уровень привлекаемых для решения средств, по исчерпании которых выдается отказ. Цикл сканирования задачи осуществляется по следующей схеме:

а) Предпринимается обращение к группе приемов, закрепленных за названием типа задачи. Эта группа представляет собой алгоритмическую процедуру, получающую в качестве входных данных решаемую задачу и ее текущий уровень. В случае срабатывания какого-либо приема из данной группы полагается $u := 0$, и снова переход к пункту а); иначе - переход к пункту б).

б) Последовательно просматриваются все вхождения логических символов в условия и посылки задачи, вес которых равен u либо $u + 1$. Для текущего такого символа f реализуется обращение к группе A_f приемов этого символа. Процедуре A_f сообщаются при этом в качестве входных данных: решаемая задача, координаты рассматриваемого вхождения в нее символа f, а также u. Таким образом, приемы могут анализировать окрестность вхождения символа f, усматривать возможность применения соответствующих логических преобразований, и с учетом значения текущего уровня u принимать

решение о целесообразности их выполнения, откладывая в сомнительных случаях срабатывание до больших значений u. Если ни один из приемов процедуры A_f при рассмотрении очередного терма задачи не сработал, то вес этого терма увеличивается на 1; если же прием сработал, веса всех измененных либо введенных им термов устанавливаются на 0, $u := 0$, и переход к пункту а). Установка на 0 весов всех "новых"термов задачи обеспечивает первоочередное их рассмотрение на следующих циклах сканирования. При этом термы, имеющие большие веса, временно исключаются из поля зрения решателя. Однако, по мере увеличения текущего уровня они вновь вовлекаются в цикл сканирования, а так как за период их пребывания в "теневой"зоне задачи могли произойти существенные изменения, в тот момент, когда вес указанного терма T становится равным $u + 1$, реализуется серия повторных его сканирований для значений текущего уровня от 0 до u. Далее текущий уровень восстанавливает свое прежнее значение u, и осуществляется переход к очередному терму задачи. Описанный механизм автоматического переключения внимания может дополняться предметно-ориентированной логикой переключения внимания, реализуемой специальными приемами, непосредственно изменяющими веса термов.

в) Если просмотр всех термов задачи для заданного значения u завершился безрезультатно, то $u := u + 1$. Если после этого оказалось, что $u > U$, то выдается отказ, иначе - переход к пункту а).

4. Языки, используемые для записи приемов решения задач. Прием решения задачи представляет собой алгоритмическую процедуру, выполняющую следующие действия:

а) Усмотрение возможности применения некоторого логического преобразования текущей ситуации;

б) Оценка целесообразности применения найденного преобразования и принятие решения о его выполнении;

в) Фактическая реализация преобразования.

Хотя в принципе для записи такого рода процедуры можно использовать любой из широко распространенных языков программирования, необходимость обеспечения динамичного интерактивного процесса обучения системы заставляет создавать специальные языки, ориентированные на возможно более компактное и наглядное представление приема. В подавляющем большинстве случаев прием базируется на некоторой теореме предметной области, и естественно было бы включать текст этой теоремы в формальное описание приема. В принципе, здесь можно было бы воспользоваться, например, хорошо известным языком логического программирования ПРОЛОГ. Однако, при более детальном анализе потребностей программирования приемов в развитых предметных областях на

этом пути возникают существенные технические препятствия.

Во-первых, выражения используемой теоремы могут не встречаться в решаемой задаче "в чистом"виде, а требовать определенных процедур распознавания, усматривающих их неявные вхождения. Так, при усмотрении квадратного трехчлена $ax^2 + bx + c$ коэффициент a может обратиться в единицу, и тогда заголовком подтерма ax^2 окажется не символ умножения, а степень; слагаемое bx может вообще отсутствовать, и тогда процедура должна идентифицировать коэффициент b как 0; x может представлять собой сложное выражение, и тогда возникает потребность в дополнительной процедуре, усматривающей полные квадраты, и т.д. Преодоление этих трудностей приводит к тому, что на ПРОЛОГе фактически описывается не сама теорема предметной области, а лишь сложное "метауровневое"описание вида теорем, близких к ней. Разумеется, наглядность и компактность по сравнению с исходной формулировкой теоремы при этом практически утрачиваются.

Во-вторых, для ускорения отсечения ненужных приемов оценка целесообразности их применения должна происходить одновременно с идентификацией определяемой теоремой ситуации, то есть в программе приема "теоремная"часть должна быть существенным образом перемешана с "управляющей"частью. Это также удаляет описание приема от текста его "базисной"теоремы и усложняет понимание выполняемых им действий при последующих коррекциях решающих правил, необходимых в процессе обучения. Сами решающие правила приема представляют собой сложные логические описания, и их формулировка требует использования развитого предикатного языка. Необходимость ссылаться из этих описаний на те или иные подтермы, идентифицированные с фрагментами теоремы, еще более усиливает тенденцию к представлению в приеме теоремы не на "предметном"а на "структурном"уровне.

В-третьих, при описании приема на логическом языке "структурного"уровня обнаруживаются многочисленные недостатки ПРОЛОГа, создававшегося первоначально с иной ориентацией - на уровень предметной области. Прежде всего, здесь следует отметить отсутствие в ПРОЛОГе такого существенно важного при сканировании задач типа данных, как вхождения в термы. Моделирование процессов просмотра вхождений на ПРОЛОГе связано с многократными переписываниями подтермов, и это дополнительно усложняет программирование. При описании ситуаций, складывающихся в задаче на "структурном"уровне, весьма полезными оказываются кванторные конструкции, непосредственная реализация которых на ПРОЛОГе не предусмотрена. Наконец, сама описанная выше общая схема процесса решения задач путем их сканирования требует развития определенной программной среды, для наиболее эффективной реализации которой

предпочтительнее использовать непродукционные языки - например, C++.

Для преодоления указанных трудностей потребовалось создать два различных языка, первый из которых позволяет задавать ситуации, приводящие к срабатыванию приема, а также выполняемые приемом преобразования в виде логических описаний "структурного"уровня, а второй - задает прием посредством формулировки некоторой теоремы из предметной области, сопровождаемой краткими пояснениями относительно способа ее использования - своего рода "генотипом"приема, преобразуемого компилятором в программу приема на первом языке. Процесс компиляции обеспечивает при этом возможность усмотрения неявных вхождений в задачу фрагментов используемой теоремы, перемешивание логики принятия решения с логикой идентификации теоремы, обращение к вспомогательным процедурам усмотрения истинности посылок теоремы либо выполнения промежуточных преобразований рассматриваемых в ней выражений, а также многие другие действия, приводящие к синтезу приема, способного в реальных ситуациях обеспечивать удовлетворительную имитацию поведения эксперта. Фактически компиляция сама оказывается здесь сложнейшим логическим процессом и осуществляется базой приемов, активизируемых в процессе сканирования специальных задач на трансляцию. Создание второго языка практически решило проблему наглядного и компактного представления приемов в решателе, сделало их устройство "видимым"для эксперта, обучающего систему. Это также позволило приблизиться к автоматической генерации приемов, которая сводится теперь к снабжению теорем предметной области определенной целевой разметкой.

Приведем несколько более подробное описание обоих языков. Первый из них, называемый ЛОС (Логический Описатель Ситуаций), позволяет формулировать логическое описание вида окрестности текущей точки сканирования задачи, при котором происходит срабатывание приема. Программа на языке ЛОС представляет собой ориентированное от корня дерево, в вершинах которого расположены последовательности операторов, называемые фрагментами программы. Ребра, выходящие из каждой вершины v, занумерованы числами $1, 2, \ldots, n$; переход по ребру с номером i осуществляется оператором вида "ветвь i"либо "иначе i относящимся к фрагменту вершины v. Этот фрагмент $P_1, \ldots, P_m$ образован операторами следующих типов:

а) Проверочный оператор. Это запись некоторого условия на значения входящих в оператор программных переменных. Часть этих переменных уже "определена"предшествующими операторами, и данный оператор предпринимает попытку доопределить значения еще не определенных переменных таким образом, чтобы указанное условие

было истинным.

б) Перечисляющий оператор. Как и в предыдущем случае, оператор представляет собой запись некоторого условия на значения его переменных. Однако, в отличие от проверочного оператора, перечисляющий оператор выдает не единственную версию набора значений своих выходных (то есть не определенных до обращения к нему) переменных, а перечисляет серию возможных таких наборов. Это перечисление осуществляется оператором P_i следующим образом. Сначала выдается первая версия набора значений переменных, и происходит обращение к оператору P_{i+1}. Если далее при рассмотрении цепочки операторов в некоторый момент обнаруживается ложность текущего оператора P_j, то предпринимается откат к последнему перечисляющему оператору, который выдает очередную версию набора значений переменных, и т.д. При откате к оператору P_i все значения переменных, определенные "после него автоматически сбрасываются. Наличие режима перечисления позволяет свести к минимуму специальные конструкции для организации циклов. Совокупность условий на подлежащую обнаружению при рассмотрении задачи ситуацию здесь одновременно оказывается записью процедуры такого обнаружения.

в) Оператор перехода. Это один из указанных выше операторов "ветвь i"иначе i". Первый из них используется для перехода к i-му подфрагменту текущего фрагмента при откате (первоначальное прохождение через этот оператор "слева направо"не вызывает никаких изменений программных переменных). Он удобен для склейки начальных отрезков различных приемов: сначала предпринимается попытка применить первый прием, а в случае неудачи ее, при откате, программа переходит к попытке реализации другого приема. Второй оператор, размещаемый непосредственно после некоторого проверочного либо перечисляющего оператора P_j, в случае ложности этого оператора передает управление на j-й подфрагмент. Заметим, что ложность перечисляющего оператора устанавливается в конце реализуемого им цикла перечисления.

Значениями программных переменных в языке ЛОС могут служить объекты следующих типов:

а) Символы переменных и логические символы, используемые в логическом языке решателя.

б) Термы логического языка.

в) Наборы произвольных объектов, типы которых перечисляются в данных пунктах.

г) Вхождения символов в термы и разрядов в наборы.

Язык содержит порядка 200 базисных операторов, обеспечивающих работу с логическими и сетевыми структурами данных, встречающимися

в задачах. Использование этих операторов, вместе с сотнями вспомогательных операторов, реализованных уже на ЛОСе, значительно ускоряет формулировку описаний сложных структурных образов, необходимых для принятия решений. Наличие режима перечисления позволяет рассматривать конструкции с кванторами общности и существования как своего рода операторы цикла; применение таких конструкций также сильно упрощает запись и чтение сложных логических условий. Хотя наличие режима перечисления сближает язык ЛОС с ПРОЛОГом, в нем отсутствует основной механизм функционирования ПРОЛОГа - механизм логического вывода в предметной области, использующий процедуру унификации. Такой механизм оказывается ненужным для ЛОСа вследствие того, что формулируемые на нем логические условия относятся не к объектам предметного уровня, а к структурным объектам, и проверка истинности этих условий не требует какого-либо логического вывода, а осуществляется при помощи "непосредственного усмотрения"процедурами интерпретатора. В отличие от ПРОЛОГа, обрабатывающего поступающий на вход программы запрос, язык ЛОС предназначен для обеспечения описанного выше процесса сканирования задач, что обуславливает возникновение дополнительных различий между этими языками (например, в ЛОСе появляется такой необходимый для эффективного сканирования тип данных, как вхождения). В целом, язык ЛОС, по-видимому, обеспечивает более высокий технологический уровень для обучения решателей математических задач, чем ПРОЛОГ. На этом языке была создана компьютерная система решения математических задач по элементарной алгебре и математическому анализу, продемонстрировавшая хорошую целенаправленность действий и достаточно высокий процент (порядка 90%) решаемых задач средней сложности по стандартным сборникам. Система позволяла не только получать ответ задачи, но и прослеживать пошаговым образом весь ход решения, с отображением выполняемых преобразований в обычной математической записи. В этом отношении, а также по уровню сложности решаемых задач и качеству ответа, решатель существенным образом превосходил известные системы компьютерной алгебры (отсутствие верных корней либо наличие ошибочных - типичный случай для предлагаемых такими системами ответов). Реализованный на языке ЛОС решатель доказал принципиальную возможность получения эффективной имитации действий эксперта в рамках описанного выше механизма "внутреннего логического зрения". Применявшаяся при его обучении технология - последовательные коррекции логики принятия решений в процессе анализа траекторий решения задач из обучающей выборки - может рассматриваться как своего рода аналог простейших механизмов

коррекции, используемых в нейросистемах, ориентированный на логические процессы верхних уровней. Анализ использовавшихся здесь принципов коррекции и реализация их в виде механизмов саморегулировки представляются важными направлениями дальнейших исследований.

Продолжению работ в данном направлении, однако, должно было предшествовать развитие самого языка, используемого для записи приемов. Как уже отмечалось выше, переход от логического языка предметного уровня к сложным структурным описаниям, интегрирующим в себе и логику предметного уровня, и логику принятия решений, привел к полной потере наглядности в представлении приемов. Фактически, несмотря на использование языка весьма высокого уровня, обилие заложенного в программы приемов материала сделало их столь же трудно читаемыми, как если бы они были записаны на языке ассемблера. В этой связи была предпринята попытка разработать язык еще более высокого уровня, в котором задание приема было бы максимально приближено к заданию соответствующей теоремы из предметной области. Для создания такого языка потребовалось повторно проработать серии задач по элементарной алгебре и математическому анализу, использовавшиеся при обучении первой версии решателя, чтобы выделить и систематизировать те типы дополнительной к теореме предметной области информации, наличие которых позволило бы транслятору синтезировать полноценный прием. Накопленный к настоящему времени запас таких "указателей алгоритмизации"теоремы значительно упростил и ускорил процесс обучения решателя; объем записи приема сократился, в среднем, в 3-4 раза; существенно упростилось прочтение заложенных в прием решающих правил, что необходимо для проведения коррекций. В процессе развития языка была создана вторая версия решателя, обучение которой предпринималось как для уже освоенных в первой версии разделов элементарной математики и анализа, так и для ряда новых разделов. Организованный по принципу "оглавления учебника интерфейс базы приемов этой версии позволяет быстро находить в процессе обучения нужные приемы, а представление самих приемов при помощи формулировки теоремы в стандартных математических обозначениях приближает наглядность их записи к наглядности обычных математических текстов.

Как уже отмечалось выше, второй язык позволяет формулировать нечто вроде "генотипа"приема, и был назван по этой причине ГЕНОЛÓГом (Генетический язык логического программирования). Транслятор ГЕНОЛОГа преобразует данный генотип приема в его программу на ЛОСе, которая и используется далее при решении задач. Описание приема на ГЕНОЛОГе складывается из следующих пунктов:

1) Формулировка теоремы, на которой основан прием.

2) Заголовок приема, определяющий способ применения теоремы при решении задачи.

3) Решающие правила приема - совокупность представленных на логическом языке условий на целесообразность его применения.

4) Указания транслятору, уточняющие способ генерации программы приема.

5) Ссылки на пакеты продукций, применяемые для обработки встречающихся в теореме выражений (как для редактирования термов, заносимых приемом в задачу, так и для обеспечения процесса идентификации описываемой теоремой ситуации).

В качестве примера, иллюстрирующего вид таких описаний, рассмотрим простейший прием для решения уравнения $ax = b$. Формальная запись теоремы, используемой в этом случае, имеет вид:

$$\forall abx(a \in \mathbf{R} \,\&\, b \in \mathbf{R} \,\&\, x \in \mathbf{R} \Rightarrow$$
$$\Rightarrow (ax = b \Leftrightarrow (a \neq 0 \,\&\, x = \frac{b}{a}) \vee (a = 0 \,\&\, b = 0))).$$

Заголовок приема здесь - некоторое служебное слово, указывающее, что теорема будет применяться для эквивалентной замены в направлении слева направо. Решающие правила приема указывают следующий контекст, в котором выполняется замена:

а) тип решаемой задачи - на описание;

б) преобразуемое равенство входит в условие задачи, причем это вхождение - корневое;

в) текущий уровень сканирования, при котором происходит срабатывание, равен 1;

г) выражение b не содержит неизвестных задачи, а x - содержит.

Как и формулировка теоремы, условия на контекст срабатывания записываются на "полномасштабном"логическом языке, позволяющем задавать произвольные описания ситуаций, складывающихся при решении задачи. Таким образом, в описании приема на ГЕНОЛОГе присутствуют сразу два логических уровня - предметный (теорема) и структурный (решающие правила).

Указания транслятору в рассматриваемом примере уточняют следующие подробности:

а) Переменная x идентифицируется как совокупность всех множителей рассматриваемого произведения, содержащих неизвестные задачи (это автоматически предопределяет, что переменная a будет идентифицирована с выражением без неизвестных).

б) При идентификации выражения ax возможен учет стоящего перед ним знака минус, который относится к коэффициенту a.

в) Посылки теоремы проверяются при помощи специального пакета продукций, обеспечивающего быстрое усмотрение условий вида $a \in \mathbf{R}$.

Новые термы формируются приемами путем подстановки

идентифицированных с переменными выражений в некоторые подтермы теоремы. Если при этом не выполнять каких-либо упрощающих преобразований, то как правило возникает цепочка срабатываний простейших приемов, выполняющих такие упрощения впоследствии. Каждому срабатыванию при этом предшествует весьма трудоемкий цикл сканирования, и работа решателя существенно замедляется. Для преодоления этого явления в описании приема указываются ссылки на пакеты продукций либо вспомогательные задачи, применяемые для упрощения новых термов еще до занесения их в решаемую задачу. В рассматриваемом примере используются следующие преобразования такого типа:

а) Предпринимаются обращения к вспомогательным задачам разложения на множители выражений a,b. Если такие разложения не находятся сразу же, до преобразования рассматриваемого уравнения, то впоследствии возникает разбор случаев, приводящий к дублирующим попыткам разложения на множители в различных подслучаях.

б) Выполняется обращение к пакету стандартизации дробных выражений для $\frac{b}{a}$. 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.

в) Для уравнений с $a = 0$, $b = 0$ выполняются обращения к пакетам продукций, 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.).

г) К заменяющей дизъюнкции применяется пакет общелогических продукций (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 цикла сканирования задачи обычно возрастает пропорционально квадрату суммарной длины находящихся в активной области ее термов. Поэтому вынесение во вспомогательные задачи преобразований подтермов сложного выражения приводит, если эти преобразования достаточно интенсивны, к значительному ускорению решения задачи.

Приведем краткое описание наиболее часто встречающихся при описании приемов конструкций языка.

1. Заголовки приемов. Самые типичные случаи - замена слева направо либо справа налево при помощи теоремы - тождества либо эквивалентности; вывод следствий в посылках либо условиях задачи. Для одновременной замены группы посылок либо условий задачи

используются специальные приемы групповой замены. Теорема может использоваться не для фактического преобразования некоторого терма задачи, а лишь для создания некоторого запаса его "альтернативных представлений используемых по мере надобности другими приемами. Для указания на такую "неявную"замену введены два дополнительных заголовка, двойственных заголовкам "явной"замены. Наконец, теорема может использоваться в составе вспомогательного пакета продукций. Пакеты продукций создаются для следующих целей:

а) Преобразование к специальному виду поступающего на вход пакета выражения (например, стандартизация дробных либо степенных выражений).

б) Обеспечение быстрого усмотрения истинности заданного отношения на конкретном наборе входных термов (например, проверка отличия числа от 0).

в) Обеспечение быстрого определения некоторых характеристик заданного набора входных термов (например, нахождение знака числа, с указанием того, является ли он строгим).

Входным данным любого из таких пакетов является также список посылок - утверждений, в предположении истинности которых применяются продукции. Продукции задаются на ГЕНОЛОГе так же, как приемы, активизируемые при сканировании задач, и для них предусмотрены соответствующие заголовки. Обращение к пакетам типа а) происходит при указанной выше стандартизации термов, формируемых приемом. Обращения к пакетам типов б) и в) происходят при усмотрении истинности посылок теоремы приема. Аналогичные обращения к пакетам продукций допускаются и из самих продукций, что делает эти пакеты мощным средством для реализации вспомогательных вычислений. Вообще говоря, вынесение из цикла сканирования задачи цепочки преобразований, использующей сравнительно небольшой запас теорем, и реализация их в рамках специального пакета продукций приводит к существенному ускорению процесса решения. Разумеется, элиминировать полностью цикл сканирования задачи таким образом не удается, возникает лишь тенденция к укрупнению приемов, то есть к реализации за один цикл сканирования возможно большего объема вычислений.

2. Решающие правила приема. Эта часть описания приема наиболее близка к описаниям ситуаций структурного уровня, возникающим при программировании на ЛОСе. Такая близость вполне естественна, если учесть, что ЛОС представляет собой язык предикатного типа, ориентированный на описание "метауровневых"ситуаций. Здесь возникает лишь возможность добиться некоторого сокращения текстов, обеспечиваемая транслятором ГЕНОЛОГа: исключение из описаний переменных, восстанавливаемых по умолчанию, и ссылка на

рассматриваемые в условиях термы либо вхождения через теоремные переменные, идентифицируемые с этими термами и вхождениями. Кроме того, при формулировке решающих правил могут использоваться те же указатели на альтернативную идентификацию выражений (вырожденные значения коэффициентов, учет симметрии и т.п.), что и при описании способа идентификации определяемой теоремой ситуации. Это вносит дополнительные сокращения в запись таких правил. В результате, текст записанных на ГЕНОЛОГе решающих правил приема составляет, как правило, не более одной десятой части от общего объема программы приема на ЛОСе. В сложных случаях предусмотрено создание специальных пакетов продукций, в которые выносится формулировка отдельных условий срабатывания приема.

3. Указания транслятору. Прежде всего, уточняются возможности альтернативной (формально отличающейся от текста теоремы) идентификации теоремных переменных. Приведем некоторые наиболее часто встречающиеся типы такой идентификации:

а) Учет "единичного"значения переменной, при котором внешняя двуместная операция вырождается. Например, при идентификации квадратного трехчлена $ax^2 + bx + c$ следует учитывать возможность обращения коэффициента a в 1, и тогда заголовком терма, идентифицируемого с первым слагаемым, должно быть не умножение, а символ степени.

б) Учет внешней одноместной операции, которая при идентификации относится к одному из операндов текущей многоместной операции. Например, при идентификации суммы дробей $a/b + c/d$ возможно допускать наличие перед этими дробями знаков минус (фактически в тексте теоремы отсутствующих), относя эти знаки, например, к термам, идентифицированным с числителями a и c.

в) Учет возможности одновременной перестановки операндов у некоторой группы двуместных операций либо отношений. Например, в теореме $0 < a \,\&\, 0 < b \Rightarrow 0 < ab$ допускается одновременная перестановка операндов у первых двух неравенств.

г) Учет возможности отсутствия операнда многоместной операции при обращении его "коэффициента"в 0. Например, при усмотрении многочлена третьей степени $ax^3 + bx^2 + cx + d$ возможно фактическое отсутствие члена bx^2, и формируемая транслятором процедура идентификации должна в таких случаях полагать b нулем.

д) Группировка операндов многоместной операции по заданному признаку. Например, при усмотрении уравнения $ax = b$ переменная x должна идентифицироваться с группой всех неизвестных сомножителей соответствующего произведения, а a - с остальными сомножителями.

е) Идентификация выражения вида $f(x)$ может происходить, при наличии соответствующего указания транслятору, с произвольным

выражением R текущего терма (предполагается, что переменная x до этого уже идентифицирована с некоторой переменной y текущей задачи), а далее, при формировании выражений вида $f(A)$, будет находиться результат подстановки в R вместо переменной y выражения для теоремного терма A.

ж) Для усмотрения некоторых специальных подвыражений могут применяться вспомогательные процедуры, выполняющие достаточно сложные преобразования. Так, при идентификации x^2 используется специальная процедура усмотрения полного квадрата в анализируемом произведении. Эта процедура, в зависимости от контекста, может использовать даже "искусственное"представление вида $(\sqrt{x})^2$ (например, если в упрощаемом выражении уже встречается $\sqrt{x}$).

Всего накоплено несколько десятков указателей альтернативной идентификации, соответствующих наиболее часто встречающимся простейшим свойствам операций и отношений, позволяющим обобщать вид теоремы. Следующий важный класс указаний транслятору составляют указания на способ учета посылок теоремы. Здесь можно выделить следующие типичные возможности:

а) Посылка теоремы непосредственно идентифицируется с некоторым утверждением из контекста текущего вхождения в задачу.

б) Для усмотрения истинности посылки используется некоторый пакет продукций. Либо такой пакет просто проверяет истинность заданного утверждения (например, усматривает отличие значения выражения от 0), либо пытается найти значения некоторых еще не идентифицированных переменных посылки, при которых она истинна (например, при определении знака выражения и уточнении того, является ли он строгим).

в) Посылка теоремы, имеющая вид равенства, может использоваться для идентификации одной из частей этого равенства с другой, все переменные которой до этого уже были определены. Последняя может быть обработана до такой идентификации некоторым нормализующим пакетом продукций либо даже вспомогательной задачей на преобразование. Так, при разложении квадратного трехчлена на множители можно вынести идентификацию дискриминанта с полным квадратом в посылку вида $R^2 = b^2 - 4ac$, связав с правой частью этого равенства указание на обращение к вспомогательной задаче, предпринимающей попытку разложения на множители.

г) Истинность посылки теоремы может устанавливаться при помощи обращения к задаче на доказательство.

Среди указаний транслятору размещаются также описания различных дополнительных действий, выполняемых при срабатывании приема. Такими действиями могут быть: ввод либо удаление различных

комментариев задачи; занесение в список посылок либо условий задачи различных сопровождающих преобразование утверждений (например, указание ограничений по о.д.з. для новых выражений); изменение весов посылок либо условий задачи для избирательного переключения внимания, и т.п. Наконец, может уточняться версия выполняемого приемом основного преобразования: применять ли замену лишь к текущему вхождению заменяемого терма, либо ко всем его вхождениям в задачу; осуществлять ли фактическую замену выражения, либо попытаться решить с применением данной замены лишь вспомогательную копию текущей задачи, с откатом при неудаче к первоначальной ситуации, и др.

Описание приема на языке ГЕНОЛОГ приближается по своей наглядности к описаниям на "естественном"математическом языке и в этом смысле может рассматриваться как формализация более высокого уровня, чем программа этого приема на языке типа ЛОСа или ПРОЛОГа. Язык ГЕНОЛОГ оказывается своего рода связующим звеном между логическим языком предметной области и языком программирования. Он существенно упрощает формирование алгоритмических конструкций на основе теорем предметной области и позволяет создавать циклы логического вывода в теории, сопровождающиеся автоматической генерацией приемов для возникающих новых теорем. Решение вспомогательных задач на "редуцирование"теоремы, получаемой после очередного применения правила вывода, позволяет сохранять лишь "базисные"(в смысле применяемого редуцирования) теоремы, и в сочетании с несложными ограничителями (например, на длину теоремы) сводит к практически приемлемому количество новых теорем. Циклы логического вывода с редуцированием и автоматической генерацией приемов для новых теорем были реализованы при помощи ГЕНОЛОГа в таких простейших разделах, как алгебра логики и алгебра множеств. Многообразия приемов, возникших при этом, практически совпали с многообразиями приемов, введенных для указаных разделов при обучении системы "вручную". Проведение дальнейших исследований, связанных с автоматической генерацией приемов и организацией процессов логического вывода в теории, могло бы привести к новым технологическим возможностям разработки эффективных решателей задач.

В заключение автор хотел бы выразить свою искреннюю благодарность В.Б. Кудрявцеву за поддержку.