Развитие языка Пролог.

Лекция 9.

Специальности : 230105, 010501

Три проблемы реализации вычислительной модели логического программирования.

- Уточнение порядка выборов, оставшихся неопределенными в абстрактном интерпретаторе.
- Увеличение выразительной силы исходной вычислительной модели за счет добавления металогических и внелогических средств.
- Использование возможностей архитектуры применяемого компьютера.

"Чистый" Пролог.

“Чистый” Пролог представляет собой приближенную реализацию вычислительной модели логического программирования на последовательной машине. Логические программы, исполняемые с помощью вычислительной модели Пролога, называются программами на “чистом” Прологе.

Выполнение программ на Прологе заключается в работе абстрактного интерпретатора, при которой вместо произвольной цели выбирается самая левая цель, а недетерминированный выбор предложения заменяется последовательным поиском унифицируемого правила и механизмом возврата. Иными словами, используется стековый метод расписания.

Вычислительная модель логических программ.

Основу вычислительной модели логических программ составляет алгоритм унификации.

Считается, что терм t есть общий пример двух термов t_1 и t_2, если существуют такие подстановки θ_1 и θ_2, что t равно $t_1\theta_1$ и $t_2\theta_2$. Терм s называется более общим, чем терм t, если t – пример s, но s не является примером t. Унификатором двух термов называется подстановка, которая делает термы одинаковыми. Наибольшим общим унификатором (н.о.у.) двух термов называется унификатор, соответствующий наиболее общему примеру.

Процесс вычисления логической программы начинается с некоторого исходного вопроса G и завершается либо успешно (существует как минимум один доказанный пример G), либо отказом.

Вычисление логической программы развивается с помощью редукции целей. Вычислением цели $Q=Q_0$ программой P называется последовательность троек $<Q_i,G_i,C_i>$, где Q_i – конъюнктивная цель, G_i – цель, входящая в Q_i,C_i – предложение $A\leftarrow B_1,\ldots,B_k$ в P с таким переименованием, что новые символы переменных не встречаются в Q_j, $0\le j\le l$. Для всех $i>0$ цель Q_{i+1} является или результатом замены G_i на тело C_i в Q_i и применения подстановки Q_i, н.о.у. для термов G_i и A_i (A_i – заголовок C_i), или константой true, если G_i – единственная цель в Q_i и тело C_i пусто, или константой "отказ" (false), если G_i и заголовок C_i не унифицируемы. Протоколом вычисления $<Q_i,G_i,C_i>$ логической программы называется последовательность пар $<G_i,\theta_l'>$, где θ_l' – подмножество н.о.у. θ_l, полученного на i-й редукции и ограниченного переменными из исходного вопроса G.

Под методом расписания интерпретатора понимается метод, в соответствии с которым добавляются и удаляются цели в резольвентах. В абстрактном интерпретаторе метод расписания не уточнен.

Вычислительная модель Пролога.

Вычисление цели G относительно программы P, написанной на Прологе, состоит в порождении всех решений цели G относительно программы P.

Протокол вычисления в Прологе есть расширение протокола логической программы при использовании абстрактного интерпретатора. При этом вводятся обозначения для описания отказов и возвратов. Для вычислительной модели Пролога вводятся различия между внешним и глубоким возвратами.

Внешний возврат возникает при безуспешной унификации цели и предложения, в этом случае пробуется новое предложение. Глубокий возврат возникает при безуспешной унификации цели и последнего предложения процедуры, в этом случае происходит возврат к другой цели в дереве вычисления.

При успешном выполнении вычисления средства управления программ на языке Пролог подобны средствам управления в обычных процедурных языках. Обращение к некоторой цели соответствует вызову процедуры, порядок целей в теле правила соответствует последовательности операторов.

Рекурсивный вызов цели в Прологе в последовательности действий и в реализации подобен тому же вызову в рекурсивных языках. Различие заключается в использовании механизма возврата.

Сравнение Пролога с традиционными языками программирования.

1) **Терм как единообразная структура данных в Прологе соответствует общим структурам записей в обычных языках программирования.**
2) **Логические переменные соотносятся с объектами, а не с ячейками памяти.**
3) **Логическое программирование не поддерживает механизм деструктивного присваивания, который позволяет изменять значение инициализированной переменной.**
4) **В логическом программировании обработка данных полностью заключена в алгоритме унификации. В унификации реализованы :**

– **однократное присваивание,**

– **передача параметров,**

– **размещение записей,**

– **доступ к полям записей для одновременных чтения/записи.**

5) **Чистый Пролог не содержит механизма обработки ошибок и исключительных ситуаций,**

Особенности различных реализаций языка Пролог.

1977 г. - Edinburgh Prolog – транслятор с языка Пролог Девида Уоррена. Выделял специальные случаи унификации и преобразовывал их в эффективные последовательности обычных операций работы с памятью.

1980 г. - К. Кларк и Ф. Маккейб разрабатывают интерпретатор языка микропролог (IC-пролог) для микрокомпьютеров. Недостаток : недостаточно эффективная реализация дополнительных средств управления, как результат – увеличение времени работы программ.

1981 г. – начало развития параллельных логических языков. Часть идей заимствована из IC-пролога.

1983 - Clark & Gregory (Imperial College, Великобритания) создают язык ParLog. Особенности : защита целей и недетерминизм с передачей выбора.

1997 г. – под рук. Gopalan Nadathur (University of Minnesota, США) был разработан Lambda-Prolog – позволяет использует предикаты высших порядков (аналоги функционалов в Лиспе). Для их интерпретации применяется стратегия “сначала вглубь”.

1980-е - CU-Prolog (Constraint Unification Prolog, (CUP) разработан в Institute for New Generation Computer Technology (ICOT), Япония, как экспериментальный язык программирования на основе логики ограничений. В отличие от большинства систем логического программирования в ограничениях CU-Prolog позволяет использовать определяемые пользователем предикаты в качестве ограничений, что особенно удобно при решении задач обработки ЕЯ с применением унификационных грамматик.

Применение металогических предикатов.

Металогические предикаты служат для анализа структуры доказательства. Поскольку данный вид предикатов выходят за рамки логики первого порядка, их применение весьма ограничено в реализациях Пролога со строгой типизацией данных. Основное назначение таких предикатов – анализ структуры доказательства. К числу металогических предикатов относят :

- **Типовые металогические предикаты. Их назначение состоит в определении принадлежности рассматриваемого объекта заданному типу. Если металогическая цель стоит в начале тела Пролог-правила для определения того, какой именно образец правила должен быть выполнен, то такую металогическую цель называют металогическим тестом. Металогические тесты могут быть использованы для выбора наилучшего порядка целей в Пролог-правилах.**
- **Предикаты проверки унифицируемости произвольных термов.**
- **Предикаты для использования переменных в качестве объектов. Примеры : freeze и melt в Прологе-10. Смысл : все переменные исходного терма, которым не были сопоставлены значения, заменяются уникальными константами.**

Полезным примером применения типовых металогических предикатов может послужить программирование алгоритма унификации. Посредством типовых металогических предикатов, определяющих принадлежность рассматриваемых термов либо к константам, либо к функторам, либо к переменным, либо к составным объектам явным образом может быть определена унификация с проверкой на вхождение переменной в терм, с которым производится унификация переменной.

Программирование в ограничениях.

Логическое программирование в ограничениях (Constraint Logic Programming, CLP) представляет собой сочетание подхода к решению задач в ограничениях с логическим программированием. Сам процесс удовлетворения ограничениям представляет собой поиск таких комбинаций значений переменных, которые соответствуют ограничениям.

Сама проблема удовлетворения ограничениям формулируется следующим образом.

Дано :

1) **Множество переменных;**
2) **Области определения, из которых могут выбираться значения переменных;**
3) **Ограничения, которым должны удовлетворять переменные.**

Требуется найти такие значения, присваиваемые переменным, которые удовлетворяют всем заданным ограничениям.

К типичным примерам таких задач относятся задачи планирования, снабжения и управления ресурсами на производстве, на транспорте и в складском хозяйстве. Для решения этих задач необходимо распределять ресурсы по процессам : транспортные средства общего пользования по заданным маршрутам, экипажи по самолетам, распараллеливание вычислительных задач и т.д. Такие задачи могут быть описаны в терминах известной задачи составления расписаний.

Решение задачи удовлетворения ограничений.

Чаше всего условия задач удовлетворения ограничений изображают в виде сетей ограничений – графов, узлы которых соответствуют переменным, а дуги – ограничениям. Для каждого бинарного ограничения p(X,Y) между переменными X и Y в этом ориентированном графе имеются две дуги, (X,Y) и (Y,X). Для поиска решения задачи удовлетворения ограничений могут использоваться различные алгоритмы обеспечения совместимости. Они проверяют совместимость областей определения переменных с ограничениями.

Пусть X и Y имеют области определения Dx и Dy, соответственно. Предположим, что X и Y : ∃ p(X,Y) – бинарное ограничение. Дуга (X,Y) называется совместимой с определяемым ограничением (или просто совместимой), если для ∀ X∈Dx ∃ Y∈Dy : p(X,Y)=true. В целях обеспечения совместимости дуги (X,Y) при введении ограничения p(X,Y) в ряде случаев приходится сокращать либо область определения Dx, либо область определения Dy. При этом возможен эффект побочного действия, заключающийся в появлении других несовместимых дуг. Данный эффект может распространяться в течение некоторого времени по всей сети до тех пор, пока все дуги не станут совместимыми или некоторая область определения не станет пустой. В случае, если все дуги являются совместимыми, могут возникать еще две ситуации.

- **Каждая область определения включает единственное значение; это означает, что данная задача удовлетворения ограничений имеет единственное решение.**
- **Все области определения непусты, и по меньшей мере одна область определения содержит несколько значений. В этом случае нет гарантии, что все возможные сочетания значений из областей определения являются решениями задачи удовлетворения ограничений. Здесь требуется выполнить комбинаторный поиск по сокращенным областям определения.**

Расширение Пролога для использования в качестве языка логического программирования в ограничениях.

Стандартный Пролог может рассматриваться как язык удовлетворения ограничений. Сами ограничения при этом представляют собой ограничения равенства между термами. Поскольку подобного рода ограничения есть ограничения между параметрами предикатов и задаются в терминах других предикатов, то вызовы этих предикатов в конечном итоге сводятся к согласованию. Для расширения интерпретатора Пролога до системы CLP необходимо введение в его состав обработки арифметических ограничений равенства и неравенства, которая позволила бы непосредственно решать задачи типа составления расписаний.

Системы CLP различаются по типам областей определения и типам ограничений, который они способны обрабатывать, и упоминаются в форме CLP(X), где X обозначает область определения. Например, в методах CLP(R) областями определения переменных являются действительные числа, а в качестве ограничений применяются операции проверки на равенство и неравенство, а также операции сравнения действительных чисел.

Доступные области определения и типы ограничений в фактических реализациях в значительной степени зависят от существующих методов решения конкретных типов ограничений. Например, в системах CLP(R) обычно доступны линейные равенства и неравенства, поскольку существуют эффективные методы обработки ограничений этих типов.

Современные результаты исследований в этой области публикуются в специализированном журнале “Constraints” (издательство “Kluwer Academic”), а также в журналах Journal of Logic Programming и Artificial Intelligence Journal.

Actor Prolog.

Основная проблема традиционных логических языков – отсутствие разрушающего присваивания. Данная проблема является актуальной при рассмотрении взаимодействия программы с внешней средой. Здесь вознакает техническая трудность реализации "бэктрэкинга" для необратимых процессов (простейший пример – вывод Пролог-программой какой-либо информации на принтер), для которых возврат не возможен по определению.

Для решения данной проблемы была предложена идея логических акторов как повторно доказываемых подцелей. При этом Пролог-программа представляется состоящей из акторов, для каждого из которых в случае возникновения логического противоречия будет отменен результат доказательства всех акторов, вступивших в противоречие с рассматриваемым. Например, когда требуется присвоить значение 5 переменной A, которая уже имеет значение 7, необходимо отменить результаты доказательства всех акторов, вступающих в логическое противоречие с утверждением A=5, после чего провести повторное доказательство этих акторов при A=5. При успешном завершении повторного доказательства считается, что было совершено логически корректное разрушающее присваивание для переменной A. В случае невозможности осуществления последнего совершается откат, но до некоторой определённой точки в истории исполнения программы. При этом в ходе отката программы откат некоторых внешних акторов может задерживаться. Основным требованием здесь является отсутствие логических противоречий между "внешним" актором и и акторами, находящимися внутри компьютера.

С точки зрения идеологии Акторного Пролога, во внешнем мире тоже существуют логические акторы, и программа взаимодействует именно с ними. Причем повторное доказательство этих акторов может происходить в непредсказуемые моменты времени, независимо от исполняемой программы. В примере с выводом на печать испорченный лист бумаги рассматривается как логический актор, откат которого никогда не произойдёт, но доказательство которого так и не будет отменено.

Основные преимущества введения акторов.

- **Реализуемость средствами логики предикатов первого порядка.**
- **Возможность отмены результатов доказательства отдельно взятых акторов позволяет осуществлять "опережающие" вычисления в параллельном программировании, появляется возможность обходиться без синхронизации параллельных процессов.**
- **Возможность сохранения состояния программы в файле на внешнем носителе с последующим возобновлением выполнения программы точно с того места, когда она была сохранена.**

Перспективы логического программирования в Искусственном Интеллекте.

С точки зрения перспектив Логического Программирования из тенденций развития технологий интеллектуальной обработки информации наибольший интерес представляют :

- **Развитие идей методов перебора, позволяющих максимально упростить описание моделируемых объектов. Данный подход позволит решить ряд задач, в частности, из области криптографии.**
- **Развитие простых, но ресурсоемких стратегий адаптивного поведения интеллектуальных объектов в сочетании с ростом производительности вычислительной техники.**
- **Интеграция различных систем логического вывода в единых оболочках с целью более полного отражения реальности.**
- **Переход от концепции детального представления информации об объектах и приемов манипулирования этой информацией к более абстрактным формальным описаниям и применению универсальных механизмов вывода.**
- **Объединение и взаимная интеграция парадигм Логического, Объектно-ориентированного и Функционального программирования.**

Рекомендуемая литература.

1. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М.: Мир, 1990. С. 62-71, 80-86, 118-127, 210-224
2. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. - М.: Издательский дом "Вильямс", 2004. С. 149-161, 301-325
3. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. - СПб.: БХВ-Петербург, 2003. С. 159-160
4. Dale A. Miller and Gopalan Nadathur Some Uses of Higher-Order Logic in Computational Linguistics // Proceedings of the 24th annual meeting on Association for Computational Linguistics. New York, April 1986, PP. 247-256
5. λProlog Home Page. // http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/
6. Cu-prolog : experimental constraint logic programming language. // http://www.miba.auc.dk/~magnus/pkgsrc-kolga/lang/cu-prolog/
7. Constraint Programming. History of Research and Application. // A Site for Constraint Programming. http://www.constraint.org/constraint_org_e.htm
8. Морозов А.А. Введение в Акторный Пролог. // Getting Started in Actor Prolog. http://www.cplire.ru/Lab144/start/r_index.html