Лекция 5.
Тема: Методы поиска решений на основе исчисления предикатов.

Альфа-бета-процедура

Теоретически, это эквивалентная минимаксу процедура, с помощью которой всегда получается такой же результат, но заметно быстрее, так как целые части дерева исключаются без проведения анализа. В основе этой процедуры лежит идея Дж. Маккарти об использовании двух переменных, обозначенных α и β (1961 год).

Основная идея метода состоит в сравнении наилучших оценок, полученных для полностью изученных ветвей, с наилучшими предполагаемыми оценками для оставшихся. Можно показать, что при определенных условиях некоторые вычисления являются лишними. Рассмотрим идею *отсечения* на примере рис. 1. Предположим, позиция A полностью проанализирована и найдено значение ее оценки α. Допустим, что один ход из позиции Y приводит к позиции Z, оценка которой по *методу минимакса* равна z. Предположим, что $z \leq \alpha$. После анализа узла Z, когда справедливо соотношение $y \leq z \leq \alpha \leq s$, ветви дерева, выходящие из узла Y, могут быть отброшены (альфа-отсечение).

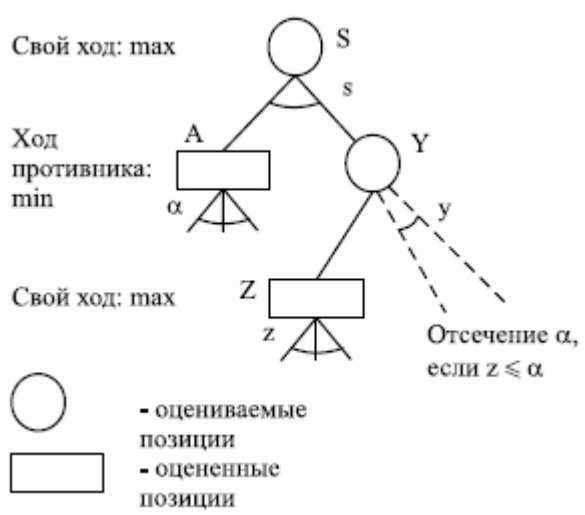

Рис. 1. - отсечение

Если мы захотим опуститься до узла Z, лежащего на уровне произвольной глубины, принадлежащей той же стороне, что и уровень S, то необходимо учитывать минимальное значение оценки β, получаемой на ходах противника.

Отсечение типа β можно выполнить всякий раз, когда оценка позиции, возникающая после хода противника, превышает значение β. Алгоритм поиска строится так, что оценки своих ходов и ходов противника сравниваются при анализе дерева с величинами α и β соответственно. В начале вычислений этим величинам присваиваются значения $+\infty$ и $-\infty$, а затем, по мере продвижения к корню дерева, находится оценка начальной позиции и наилучший ход для одного из противников.

Правила вычисления α и β в процессе поиска рекомендуются следующие:

1. у MAX вершины значение α равно наибольшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин;

2. у MIN вершины значение β равно наименьшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин.

Правила прекращения поиска:

3. можно не проводить поиска на поддереве, растущем из всякой MIN вершины, у которой значение β не превышает значения α всех ее родительских MAX вершин;

4. можно не проводить поиска на поддереве, растущем из всякой MAX вершины, у которой значение α не меньше значения β всех ее родительских MIN вершин.

На рис. 2 показаны $\alpha - \beta$ *отсечения* для конкретного примера. Таким образом, $\alpha - \beta$ -алгоритм дает тот же результат, что и *метод минимакса,* но выполняется быстрее.

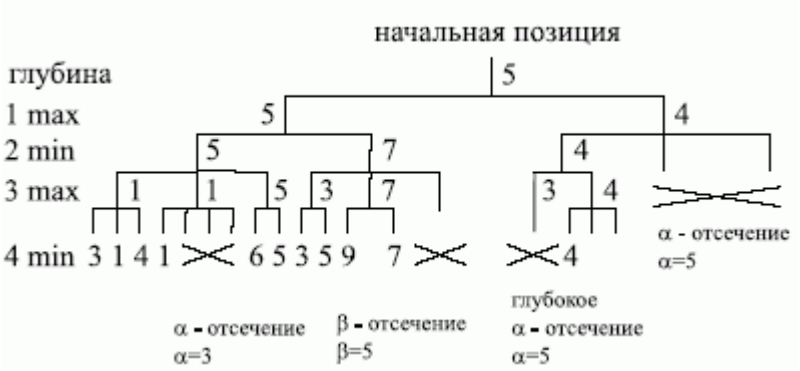

Рис. 2. a-b отсечение для конкретного примера

Использование *алгоритмов эвристического поиска* для поиска на графе И, ИЛИ выигрышной *стратегии* в более сложных задачах и играх (шашки, шахматы) не реален. По некоторым оценкам игровое дерево игры в шашки содержит 10^{40} вершин, в шахматах 10^{120} вершин. Если при игре в шашки для одной вершины требуется 1/3 наносекунды, то всего игрового времени потребуется 10^{21} веков. В таких случаях вводятся искусственные условия остановки, основанные на таких факторах, как наибольшая допустимая *глубина вершин* в дереве поиска или ограничения на время и объем памяти.

Многие из рассмотренных выше идей были использованы А. Ньюэллом, Дж. Шоу и Г. Саймоном в их программе *GPS*. Процесс работы *GPS* в общем воспроизводит *методы решения задач*, применяемые человеком: выдвигаются подцели, приближающие к решению; применяется *эвристический метод* (один, другой и т. д.), пока не будет получено решение. Попытки прекращаются, если получить решение не удается.

Программа *STRIPS* (STanford Research Institut *Problem* Solver) вырабатывает соответствующий *порядок действий робота* в зависимости от поставленной цели. Программа способна обучаться на опыте решения предыдущих задач. Большая часть *игровых программ* также обучается в процессе работы. Например, знаменитая шашечная программа Самюэля, выигравшая в 1974 году у чемпиона мира, "заучивала наизусть" выигранные партии и обобщала их для извлечения пользы из прошлого опыта. Программа HACHER Зуссмана, управляющая поведением робота, обучалась также и на ошибках.

Методы поиска решений на основе исчисления предикатов

Семантика исчисления предикатов обеспечивает основу для формализации логического вывода. Возможность *логически выводить* новые правильные выражения из набора истинных утверждений очень важна. Логически выведенные утверждения корректны, и они совместимы со всеми предыдущими интерпретациями первоначального набора выражений. Обсудим вышесказанное неформально и затем введем необходимую формализацию.

В исчислении высказываний основным объектом является переменное *высказывание* (*предикат*), истинность или ложность которого зависит от значений входящих в него переменных. Так, истинность предиката " x был физиком" зависит от значения переменной

x. Если x - П. Капица, то *предикат* истинен, если x - М. Лермонтов, то он ложен. На языке *исчисления предикатов* утверждение $\forall x(P(x) \supset Q(x))$ читается так: "для любого x если P(x), то имеет *место* и Q(x) ". Иногда его записывают и так: $\forall x(P(x) \rightarrow Q(x))$. Выделенное *подмножество* тождественно истинных формул (или *правильно построенных формул* - ППФ), истинность которых не зависит от истинности входящих в них высказываний, называется *аксиомами*.

В *исчислении предикатов* имеется множество правил вывода. В качестве примера приведем классическое правило отделения, или modus ponens :

(A, A -> B) / B

которое читается так "если *истинна формула* A и истинно, что из A следует B, то истинна и формула B ". Формулы, находящиеся над чертой, называются посылками вывода, а под чертой - заключением. Это *правило вывода* формализует основной закон дедуктивных систем: из истинных посылок всегда следуют истинные заключения. *Аксиомы* и правила вывода *исчисления предикатов* первого порядка задают основу формальной дедуктивной системы, в которой происходит формализация схемы рассуждений в *логическом программировании*. Можно упомянуть и другие правила вывода.

Modus tollendo tollens : Если из A следует B и B ложно, то и A ложно.

Modus ponendo tollens : Если A и B не могут одновременно быть истинными и A истинно, то B ложно.

Modus tollendo ponens : Если либо A, либо B является истинным и A не истинно, то B истинно.

Решаемая задача представляется в виде утверждений (аксиом) f_1, F_2... F_n исчисления предикатов первого порядка. Цель задачи B также записывается в виде утверждения, справедливость которого следует установить или опровергнуть на основании аксиом и правил вывода формальной системы. Тогда решение задачи (достижение цели задачи) сводится к выяснению логического следования (выводимости) целевой формулы B из заданного *множества* формул (аксиом) f_1, F_2... F_n. Такое выяснение равносильно доказательству общезначимости (тождественно-истинности) формулы

f_1& F_2&... & F_n -> B

или невыполнимости (тождественно-ложности) формулы

$$f_1 \vee F_2 \vee \cdots \vee F_n \vee \neg B$$

Из практических соображений удобнее использовать доказательство от противного, то есть доказывать невыполнимость формулы. На доказательстве от противного основано и ведущее правило вывода, используемое в логическом программировании, - ***принцип резолюции***. Робинсон открыл более сильное правило вывода, чем modus ponens, которое он назвал *принципом резолюции* (или правилом резолюции). При использовании *принципа резолюции* формулы *исчисления предикатов* с помощью несложных преобразований приводятся к так называемой дизъюнктивной форме, то есть представляются в виде набора дизъюнктов. При этом под дизъюнктом понимается дизъюнкция литералов, каждый из которых является либо предикатом, либо отрицанием предиката.

Приведем пример дизъюнкта:

$$\forall x(P(x, c_1) \supset Q(x, c_2)).$$

Пусть P - *предикат* уважать, c_1 - Ключевский, Q - *предикат* знать, c_2 - история. Теперь данный дизъюнкт отражает факт "каждый, кто знает историю, уважает Ключевского".

Приведем еще один пример дизъюнкта:

$$\forall x(P(x, c_1)P(x, c_2)).$$

Пусть P - *предикат* знать, c_1 - физика, c_2 - история. Данный дизъюнкт отражает *запрос* "кто знает физику и историю одновременно".

Таким образом, условия решаемых задач (факты) и целевые утверждения задач (запросы) можно выразить в дизъюнктивной форме *логики предикатов* первого порядка. В

дизъюнктах кванторы всеобщности $\forall, \exists$, обычно опускаются, а связки $\supset, \neg, \wedge$ заменяются на $\longrightarrow$ *импликацию*.

Вернемся к *принципу резолюции*. Главная идея этого правила вывода заключается в проверке того, содержит ли множество дизъюнктов R пустой (ложный) дизъюнкт. Обычно резолюция применяется с прямым или обратным методом рассуждения. *Прямой* метод из посылок A, A -> B выводит заключение B (правило modus ponens). Основной недостаток прямого метода состоит в его не направленности: повторное применение метода приводит к резкому росту промежуточных заключений, не связанных с целевым заключением. *Обратный вывод* является направленным: из желаемого заключения B и тех же посылок он выводит новое подцелевое заключение A. Каждый шаг вывода в этом случае связан всегда с первоначально поставленной целью. Существенный недостаток *метода резолюции* заключается в формировании на каждом шаге вывода *множества* резольвент - новых дизъюнктов, большинство из которых оказывается лишними. В связи с этим разработаны различные модификации *принципа резолюции*, использующие более эффективные *стратегии поиска* и различного рода ограничения на вид исходных дизъюнктов. В этом смысле наиболее удачной и популярной является система *ПРОЛОГ*, которая использует специальные виды дизъюнктов, называемых дизъюнктами Хорна.

Процесс доказательства *методом резолюции* (от обратного) состоит из следующих этапов:

1. Предложения или *аксиомы* приводятся к *дизъюнктивной нормальной форме*.
2. К набору аксиом добавляется отрицание доказываемого утверждения в дизъюнктивной форме.
3. Выполняется совместное разрешение этих дизъюнктов, в результате чего получаются новые основанные на них дизъюнктивные выражения (резольвенты).
4. Генерируется пустое выражение, означающее противоречие.
5. Подстановки, использованные для получения пустого выражения, свидетельствуют о том, что отрицание отрицания истинно.

Рассмотрим примеры применения *методов поиска решений на основе исчисления предикатов*. Пример "интересная жизнь". Итак, заданы утверждения 1-4 в левом столбце таблица Требуется ответить на вопрос: "Существует ли человек, живущий интересной жизнью?" В виде предикатов эти утверждения записаны во втором столбце таблицы. Предполагается, что $\forall X(smart(X) = \neg stupid(X))$ и $\forall Y(wealthy(Y) = \neg poor(Y))$. В третьем столбце таблицы записаны дизъюнкты.

Таблица 1. Интересная жизнь

Утверждения и заключение	**Предикаты**	**Предложения(дизъюнкты)**
1. Все небедные и умные люди счастливы	$\forall X(\neg poor(X) \wedge smart(X) \rightarrow happy(X))$	$poor(X) \wedge \neg smart(X) happy(X)$
2. Человек, читающий книги, - неглуп	$\forall Y(read(Y) \rightarrow smart(Y))$	$\neg read(Y) smart(Y)$
3. Джон умеет читать и является состоятельным человеком	$read(John) \wedge \neg poor(John)$	$3a\ read(John) 3b\ \neg poor(John)$
4. Счастливые люди живут интересной жизнью	$\forall Z(happy(Z) \rightarrow exciting(Z))$	$\neg happy(Z) \vee exciting(Z)$

5. Заключение: Существует ли человек, живущий интересной жизнью?	$\exists W(exciting(W))$	exciting(W)
6. Отрицание заключения	$\neg\exists W(exciting(W))$	$\neg exciting(W)$

Отрицание заключения имеет вид (строка 6): $\neg\exists W(exciting(W))$

Одно из возможных доказательств (их более одного) дает следующую последовательность резольвент:

1. $\neg happy(Z)$ резольвента 6 и 4
2. $poor(X) \wedge \neg smart(X)$ резольвента 7 и 1
3. $poor(Y) \wedge \neg read(Y)$ резольвента 8 и 2
4. $\neg read(John)$ резольвента 9 и 3b
5. *NIL* резольвента 10 и 3a

Символ *NIL* означает, что *база данных* выражений содержит противоречие и поэтому наше предположение, что не существует человек, живущий интересной жизнью, неверно.

В *методе резолюции* порядок комбинации дизъюнктивных выражений не устанавливался. Значит, для больших задач будет наблюдаться экспоненциальный рост числа возможных комбинаций. Поэтому в процедурах резолюции большое *значение* имеют также *эвристики* поиска и различные *стратегии*. Одна из самых простых и понятных *стратегий* - *стратегия* предпочтения единичного выражения, которая гарантирует, что резольвента будет меньше, чем наибольшее родительское *выражение*. Ведь в итоге мы должны получить *выражение*, не содержащее литералов вообще.

Среди других *стратегий* (*поиск в ширину* (breadth-first), *стратегия* "*множества* поддержки", *стратегия* линейной *входной* формы) *стратегия* "*множества* поддержки" показывает отличные результаты при поиске в больших пространствах дизъюнктивных выражений. Суть *стратегии* такова. Для некоторого набора исходных дизъюнктивных выражений S можно указать *подмножество* T, называемое множеством поддержки. Для реализации этой *стратегии* необходимо, чтобы одна из резольвент в каждом опровержении имела предка из *множества* поддержки. Можно доказать, что если S - невыполнимый набор дизъюнктов, а S-T- выполнимый, то *стратегия множества* поддержки является полной в смысле опровержения. Исследования, связанные с доказательством теорем и разработкой алгоритмов опровержения резолюции, привели к развитию языка логического программирования *PROLOG* (*Programming* in *Logic*). *PROLOG* основан на теории предикатов первого порядка. Логическая *программа* - это набор спецификаций в рамках формальной логики. Несмотря на то, что в настоящее время удельный *вес* языков *LISP* и *PROLOG* снизился и при решении задач ИИ все больше используются C, C++ и *Java*, однако многие задачи и разработка новых *методов решения задач* ИИ продолжают опираться на языки *LISP* и *PROLOG*. Рассмотрим одну из таких задач - задачу *планирования последовательности действий* и ее решение на основе теории предикатов.

Задачи планирования последовательности действий

Многие результаты в области ИИ достигнуты при решении "задач для робота". Одной из таких простых в постановке и интуитивно понятных задач является задача *планирования последовательности действий*, или задача построения планов.

В наших рассуждениях будут использованы примеры традиционной *робототехники* (современная *робототехника* во многом основывается на реактивном управлении, а не на планировании). Пункты плана определяют *атомарные* действия для робота. Однако при описании плана нет необходимости опускаться до микроуровня и говорить о датчиках, шаговых двигателях и т. п. Рассмотрим ряд предикатов, необходимых для работы

планировщика из мира блоков. Имеется некоторый робот, являющийся подвижной рукой, способной брать и перемещать кубики. Рука робота может выполнять следующие задания (U, V, W, X, Y,Z - переменные).

goto(X,Y,Z) перейти в местоположение X, Y, Z
pickup(W) взять блок W и держать его
putdown(W) опустить блок W в некоторой точке
stack(U,V) поместить блок U на верхнюю грань блока V
unstack(U,V) убрать блок U с верхней *грани* блока V

Состояния мира описываются следующим *множеством предикатов* и отношений между ними.

on(X,Y) блок X находится на верхней *грани* блока Y
clear(X) верхняя грань блока X пуста
gripping(X) *захват* робота удерживает блок X
gripping() *захват* робота пуст
ontable(W) блок W находится на столе

Предметная область из мира кубиков представлена на рис. 3 в виде начального и целевого состояния для решения задачи *планирования*. Требуется построить *последовательность действий робота*, ведущую (при ее реализации) к достижению целевого состояния.

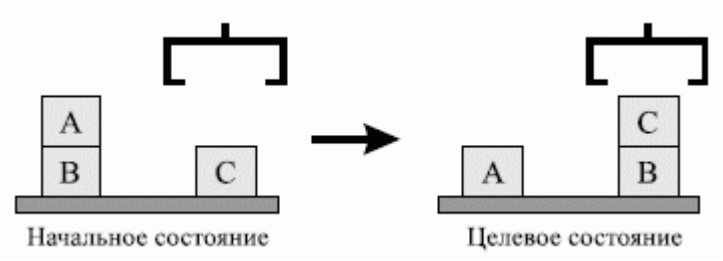

Рис.3. Начальное и целевое состояния задачи из мира кубиков

Состояния мира кубиков представим в виде предикатов. Начальное состояние можно описать следующим образом:

start = [handempty, ontable(b),
ontable(c), on(a,b), clear(c),
clear(a)]

где: handempty означает, что рука робота Робби пуста.

Целевое состояние записывается так:

goal = [handempty, ontable(a),
ontable(b), on(c,b), clear(a),
clear(c)]

Теперь запишем правила, воздействующие на состояния и приводящие к новым состояниям.

$$(\forall X)(pickup(X) \rightarrow (gripping(X) \leftarrow (gripping() \wedge clear(X) \wedge ontable(X))))$$

$$(\forall X)(putdown(X) \rightarrow ((gripping() \wedge ontable(X) \wedge clear(X)) \leftarrow gripping(X)))$$

$$(\forall X)(\forall Y)(stack(X,Y) \rightarrow ((on(X,Y) \wedge gripping() \wedge clear(X)) \leftarrow (clear(Y) \wedge gripping(X))))$$

$$(\forall X)(\forall Y)(unstack(X,Y) \rightarrow ((clear(Y) \wedge gripping(X)) \leftarrow (on(X,Y) \wedge clear(X) \wedge gripping())))$$

Прежде чем использовать эти правила, необходимо упомянуть о проблеме границ. При выполнении некоторого действия могут изменяться другие предикаты и для этого могут использоваться *аксиомы* границ - правила, определяющие инвариантные предикаты. Одно из решений этой проблемы предложено в системе *STRIPS*.

В начале 1970-х годов в Стэнфордском исследовательском институте (Stanford Research Institute *Planning System*) была создана система *STRIPS* для *управления роботом*. В *STRIPS* четыре оператора pickup, putdown, stack, unstack описываются тройками элементов. Первый элемент тройки - множество предусловий (П), которым удовлетворяет мир до применения оператора. Второй элемент тройки - *список* дополнений (Д), которые являются результатом применения оператора. Третий элемент тройки - *список* вычеркиваний (В), состоящий из выражений, которые удаляются из описания состояния после применения оператора.

Ведя рассуждения для рассматриваемого примера от начального состояния, мы приходим к *поиску в пространстве состояний*. Требуемая последовательность действий (план достижения цели) будет следующей:

unstack(A,B), putdown(A), pickup(C), stack(C,B)

Для больших графов (сотни состояний) *поиск* следует проводить с использованием оценочных функций.

Поиск решений в системах продукций

Поиск решений в *системах продукций* наталкивается на проблемы выбора правил из *конфликтного множества*, как это указывалось в предыдущей лекции. Различные варианты решения этой проблемы рассмотрим на примере ЭСО *CLIPS*, на которой нам предстоит в 7 лекции разработать исследовательский прототип ЭС. Правила в ЭС, кроме фактора уверенности эксперта, имеют приоритет выполнения (salience). ***Конфликтное множество (КМ)*** - это список всех правил, имеющих удовлетворенные условия при некотором, текущем состоянии списка фактов и объектов и которые еще не были выполнены. Как отмечалось ранее, *конфликтное множество* это простейшая *база целей*. Когда активизируется новое правило с определенным приоритетом, оно помещается в *список* правил *КМ* ниже всех правил с большим приоритетом и выше всех правил с меньшим приоритетом. Правила с высшим приоритетом выполняются в первую *очередь*. Среди правил с одинаковым приоритетом используется определенная *стратегия*.

CLIPS поддерживает семь *стратегий* разрешения конфликтов.

Стратегия глубины (depth *strategy*) является *стратегией* по умолчанию (*default strategy*) в *CLIPS*. Только что активизированное правило помещается поверх всех правил с таким же приоритетом. Это позволяет реализовать *поиск в глубину*.

Стратегия ширины (breadth *strategy*) - только что активизированное правило помещается ниже всех правил с таким же приоритетом. Это, в свою *очередь*, реализует *поиск в ширину*.

LEX стратегия - активация правила, выполненная более новыми образцами (фактами), располагается перед активацией, осуществленной более поздними образцами. Например, как это указано в таблица ниже.

MEA *стратегия* - *сортировка* образцов не производится, а осуществляется только упорядочение правил по первым образцам, как это показано в столбце 3 таблица.

Таблица. Результаты применения *LEX* и MEA стратегий

Исходный набор правил	**Правила, отсортированные *LEX***	**Правила, отсортированные MEA**
rule-6: f-1,f-4	rule-6: f-4,f-1	rule-2: f-3,f-1
rule-5: f-1,f-2,f-3	rule-5: f-3,f-2,f-1	rule-3: f-2,f-1
rule-1: f-1,f-2,f-3	rule-1: f-3,f-2,f-1	rule-6: f-1,f-4
rule-2: f-3,f-1	rule-2: f-3,f-1	rule-5: f-1,f-2,f-3
rule-4: f-1,f-2	rule-4: f-2,f-1	rule-1: f-1,f-2,f-3
rule-3: f-2,f-1	rule-3: f-2,f-1	rule-4: f-1,f-2

Стратегия упрощения (*simplicity strategy*) - среди всех правил с одинаковым приоритетом только что активизированное правило располагается выше всех правил с равной или большей определенностью (*specificity*). Определенность правила задается количеством сопоставлений в левой части правил плюс количество вызовов функций. Логические функции не увеличивают определенность правила.

Стратегия усложнения (*complexity strategy*) - среди всех правил с одинаковым приоритетом только что активизированное правило располагается выше всех правил с равной или большей определенностью.

Случайная *стратегия* (*random strategy*) - каждой активации назначается случайное число, которое используется для определения местоположения среди активаций с определенным приоритетом.

Подход на основе *стратегий поиска решений в продукционных ЭС* известен достаточно давно. Весьма популярная в начале 90-х годов ЭСО GURU (ИНТЕР-ЭКСПЕРТ) также использовала подобные *механизмы* управления *стратегиями поиска*. Возможность смены *стратегии* в ходе решения задачи программным образом и накопление опыта, какие *стратегии* дают лучшие результаты для определенных классов задач, позволяет получить эффективные *механизмы* поиска решений в *СПЗ* на основе продукций.

Завершая данную лекцию, следует отметить, что существуют различные *методы поиска решений* в *семантических сетях*, например, метод обхода семантической сети - мультипарсинг. Данный метод оригинален тем, что позволяет параллельно "вести" по графу несколько маркеров и, тем самым, распараллеливать процесс поиска информации в семантической сети, что увеличивает скорость поиска. Эти методы используются, как правило, при представлении текста в виде объектно-ориентированной семантической сети и в данной лекции не рассматриваются.