Введение. Новые информационные технологии и Искусственный интеллект (ИИ)

Традиционные средства программного обеспечения ЭВМ и ИИ

Термин **Искусственный интеллект (ИИ)** – претенциозен, метафоричен.
Реальное содержание – повышение "интеллекта" ЭВМ; передача компьютеру некоторых функций человеческой интеллектуальной деятельности; создание помощника в решении интеллектуальных задач.

Более точно:
Искусственный интеллект – область исследований и прикладных разработок, направленных на создание программно-аппаратных средств, способных к решению таких задач, решение которых предполагает применение человеком своих интеллектуальных способностей.

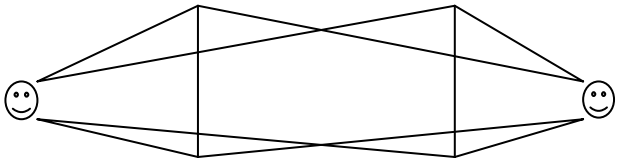
В МГУ представлены три аспекта исследований в области ИИ:
- искусственный интеллект ↔ интеллект человека (факультет психологии);
- искусственный интеллект ↔ математический аппарат (мехмат);
- искусственный интеллект ↔ программное обеспечение (ВМК).

Нас ИИ интересует именно в этом аспекте. Мы будем рассматривать проблемы ИИ в контексте создания программного обеспечения ЭВМ.

Традиционно основное внимание уделялось ***точности результатов*** работы вычислительных систем (ВС). Гораздо меньше внимания уделялось проблеме ***удобства*** работы с ВС.
60-е. 70-е гг. – ЭВМ окружена кондиционерами, обслуживается целой армией операторов.

УСЛОВИЯ РАБОТЫ "ДИКТУЕТ" МАШИНА.

Диполь Тыугу:



Связанные понятия: **интеллектуальный интерфейс, дружественный интерфейс**.

Дружественный интерфейс:
- естественные языковые конструкции и структуры меню (не требуется знание синтаксиса формальных языков общения с компьютером);
- «интуитивный» уровень взаимодействия с компьютером, не требующий длительного обучения (для профессионала – несколько часов);
- разнообразные средства общения, пригодные для пользователей различного уровня подготовки (командный язык, меню, пиктограммы, диаграммы и др.);
- для каждого уровня пользователя адекватные возможности в: меню, запросах, подсистеме помощи;
- работа в реальном времени (необходимая скорость в диалоге);
- использование манипулятора типа «мышь», «горячих клавиш», сенсорного экрана и др.;
- минимальное использование клавиатуры;
- «интеллектуальные» средства (устойчивость к ошибкам, широкое использование принципа «по умолчанию»).

Интеллектуальный интерфейс – совокупность программных и аппаратных средств, позволяющая конечному пользователю решать на компьютере характерные для его повседневной деятельности задачи без помощи посредников-программистов.
Расширение взаимодействия между человеком и компьютером с помощью:
- увеличения диапазона способов ввода и вывода;
- обогащения грамматики ввода и вывода;
- попытки кооперации с пользователем в достижении целей.

В идеале система должна иметь "модель мира задачи", над которой работают система и пользователь и которая близка модели этого мира в уме пользователя.

Достаточно реальная (и близкая) перспектива **– *речевой интерфейс*.**

Новые информационные технологии (сейчас этот термин трактуется шире) – технологии, которые должны обеспечить возможность применения ЭВМ конечным пользователем в сфере его профессиональной деятельности без помощи посредника-программиста.

Решение задач на ЭВМ (основные этапы):

Содержательная постановка задачи	Формальная постановка задачи	Разработка алгоритма	Написание программы	Получение программы на маш.яз.	Отладка, тестирование	Эксплуатация	Анализ результатов

Традиционные средства программного обеспечения помогают человеку на всех этапах кроме первого.
Системы ИИ должны быть способны помогать и на этапе содержательной постановки задачи, уточнения и необходимого пополнения содержательной постановки и ее формализации.

История: "романтический период" ИИ → серьезные научные исследования → практические задачи.

Тест Тьюринга.

50-е – 60-е гг. XX века – "романтический период" ИИ: "машинные стихи", "машинная музыка", машинный перевод, интеллектуальные игры (шашки, шахматы и др.);
60-е – 70-е гг. XX века – исследование методов решения задач (методов поиска решения);
70-е – 80-е гг. XX века – исследование методов представления знаний, нужных для решения задач;
80-е – 90-е гг. XX века – исследование методов приобретения знаний (передачи их от человека ЭВМ);
90-е гг. XX века – наше время – теоретическое осмысление, поиск новых идей и задач.

Постоянные проблемы:
Начальный уровень "знаний" системы ИИ, проблема ее обучения человеком и ее самообучения.
Общение человека с системой ИИ (языки общения, программно-аппаратные средства).
Инструментальные средства – языки программирования для задач ИИ.

Типы систем ИИ (в историческом аспекте):
Решатели задач;
Роботы;
Экспертные системы;
Агенты.

Интеллектуальная деятельность человека и ИИ, основные понятия психологии.

Будем использовать общенаучные термины: ***взаимодействие, отражение, информация, деятельность, сущность.***
Отметим, что ***информация*** – это:
структурная информация (мера неоднородности/разнообразия) и
отраженная информация (результат отражения разнообразия).
Психическое Отражение – информационное отражение, важную роль в котором играет субъективный фактор.
Психика – свойство человека, обеспечивающее возможность выявления и использования Информации о мире в процессе ***Человеческой Деятельности*** **(ЧД).**
Личность – человек как носитель Психики, как субъект Психической Деятельности.
(***организм*** – человек как биологическое существо; ***сома*** – телесная субстанция человека).
Психические явления – то, что происходит на уровне Личности.
Психическая Деятельность – информационная сторона деятельности человека.
Функции Психической Деятельности – перенос деятельности во внутренний план с целью регуляции ЧД

Психофизиологическая проблема – проблема связи психических явлений с лежащими в их основе физиологическими явлениями. Как физиологические явления (происходящие в органах чувств и в центральной нервной системе на основе физико-химических процессов) дают нам информацию о мире, воспринимаемом объекте?
Эта проблема очень актуальна, чрезвычайно сложна, слабо разработана.
"Я знаю все, но только не себя" /Ф.Вийон/.
Психосоматическая проблема – проблема связи психических явлений с соматическими/телесными.
Примеры влияния ПСИХИЧЕСКОЕ – СОМАТИЧЕСКОЕ:
психотерапия, гипноз, психогенные соматические заболевания (стрессовая язва желудка);
Примеры влияния СОМАТИЧЕСКОЕ – ПСИХИЧЕСКОЕ:
соматогенные психозы, психотропные вещества, генетико-биохимическая концепция шизофрении.

Особенности отражения сомы в Психике: не отражаются нейрофизиологические процессы, состояния организма отражаются не в виде образов, а в мотивационной форме – локализованной (боль, зуд) или не локализованной (усталость, голод).

Психические явления – по форме (с точки зрения длительности, повторяемости):
- процессы (примеры – восприятие содержания лекции, понимание смысла сказанной фразы);
- состояния (примеры – умственная усталость, бодрость и свежесть восприятия);
- свойства Личности (примеры – целеустремленность, любознательность).

Психические явления – по содержанию (с точки зрения особенностей информационных процессов):
формы Психического Отражения (эмоции, ощущения, мышление, чувства, воля, память);
внимание (внутренняя организация Психического Отражения);
психомоторика (объективация форм Психического Отражения в движениях) *сенсомоторика*, *идеомоторика*;
потребности (*биологические*, в том числе, инстинктивные; *социальные*, например, трудовые).

Потребность – Психическое Явление, побуждающее к деятельности, отражение нужды.
Мотив – Потребность, инициирующая некоторую Деятельность (мотив этой Деятельности).
Цель – возможный и планируемый Результат Деятельности.
Средства (как явления уровня Личности/Психики) – механизмы Психической Деятельности или психические модели средств ***предметной деятельности*** человека.
Результат (как явление уровня Личности/Психики) – Психическое Явление, отражающее реальный результат некоторой Деятельности, направленной на достижение определенной Цели.
Возможно несовпадение Цели и Результата Деятельности. Это – источник новой (иногда очень полезной для человека) информации.
Целенаправленность Человеческой Деятельности. Иерархическая структура Деятельности.

Про-Задача – двойка <Потребность, Цель>.
Задача – тройка <Потребность, Средства, Цель>.
Решенная-Задача – пятерка <Потребность, Средства, Цель, Деятельность, Результат>.

Продуктивная деятельность – Цель продуцирует решение (заставляет человека его придумывать). Это "чистое творчество".
Репродуктивная деятельность – Цель репродуцирует решение (готовое решение сразу же приходит в голову). Это нетворческая часть деятельности человека.
Эвристика – этот термин имеет два значения:
Эвристика – наука о творческом мышлении, изучающая формирование новых действий в новых ситуациях.
Эвристика – конкретный прием, облегчающий и/или ускоряющий поиск решения некоторой задачи.

Мышление (= Интеллект) – высшая форма Психического Отражения. Отражение по сфере сущностей, то есть **Понятийное Отражение.**
Понятие (об объекте) – Психическое Явление, отражающее ***сущность*** этого объекта.
Сущность – наиболее важные, глубинные характеристики предмета или явления, определяющие его свойства, поведение, развитие.
Понимание – выявление Сущности объекта.
Мышле'ние – способность человека к Понятийному Отражению.
Мы'шление – процесс Понятийного Отражения.

Известная триада психологии и педагогики: ***знания – умения – навыки***.
Знания – усвоенные Понятия.
Умения – способность выполнять новые действия в новых условиях.
Навыки – действия, автоматизировавшиеся в процессе их усвоения и выполнения.

Существует много различных теорий мышления, интеллекта.
Теория Интеллекта Жана Пиаже – одна из наиболее интересных теорий мышления и его развития.
Ж.Пиаже: но лишь один Интеллект «*тяготеет к тотальному равновесию, стремясь к тому, чтобы ассимилировать всю совокупность действительности и чтобы аккомодировать к ней действие, которое он освобождает от рабского подчинения изначальным "здесь" и "теперь"*».

Для нас **ИНТЕЛЛЕКТ**:
- целенаправленное планирование поведения в меняющейся проблемной среде;
- перенос деятельности во внутренний план вместо выполнения поведенческих актов;
- работа с понятийными моделями среды и себя (на основе понятийного отражения);
- скоординированная совокупность мыслительных/интеллектуальных операций – как абстрактных (метод рассуждения по аналогии), так и конкретных (способ решения определенного типа задач);

Осознаваемые и ***Неосознаваемые*** Психические Явления.

Сознание человека.
Вытеснение – неосознанное вытеснение (удаление) из сферы Сознания информации, вызывающей тревогу, отрицательные эмоции и т.п.
Установка – неосознанное внимание, ориентирующее Личность на определенную Деятельность.
Интуиция – появление в Сознании результатов неосознаваемых психических процессов.
Конформизм – неосознанное согласие с мнением большинства (даже абсурдным).
Сон (одна из его функций, есть и другие) – неосознаваемый анализ, обработка и запоминание (части) информации, полученной человеком в период бодрствования.

Учение Зигмунда Фройда (психоанализ) – интересная, но в целом ошибочная, теория неосознаваемых психических явлений.

Человек в Социальной Среде

Социальная позиция – место Личности в определенной социальной системе.
Социальная роль – нормативно одобренный способ поведения, ожидаемый от каждой личности, занимающей данную Социальную Позицию.
Межролевые конфликты. Внутриролевые конфликты.
Влияние Социальной Роли на поведение человека. Примеры.

Основные школы психологии мышления

Ассоциативная психология (XVIII-XIX вв.) – предшественники: Ньютон, Локк.
(мышление как ассоциации представлений)
(Общее в психологических теориях того времени: психическое = осознанное, психология = психология индивида, интроспекция, т.е. самонаблюдение – как главный метод исследования)
Гартли (Англия, XVIII в.): впервые ***ассоциация*** трактуется как универсальное понятие психологии, объясняющее всю психическую деятельность человека.

- механистический материализм, психофизиологический параллелизм: вибрации в периферической нервной системе → аналогичные вибрации в головном мозгу – база идей;
- детерминанты ассоциаций: смежность по времени, частота повторений;
- замечено, что если ощущения A, B, C ассоциируются с идеями a, b, c, то при появлении A могут возникнуть b, c;
- попытка объяснения бессознательного (головной мозг – осознанное, идеи; вне – ощущения);
- мотивация: удовольствие, страдание.

Беркли, Юм (Англия, XVIII в.): *ощущения* – единственный объект, другой познаваемой реальности нет.

- из ассоциаций удаляется физический субстрат;
- расширяется набор ассоциативных связей: рассматриваются ассоциации *по сходству*, *по контрасту*;
- ассоциации отрываются от реальных объектов.

Гербарт (Германия, XIX в.): ***представление*** – "первичное единство, возникающее в виде акции души, стремящейся (в противовес внешним воздействиям) к самосохранению"; представления следуют друг за другом вне зависимости от чего-либо внешнего;
апперцептивная масса – представления, силой которых в сознании (фокусе внимания) удерживаются некоторые представления;
Недостатки Ассоциативной психологии
Описательный характер.
Не вскрыты внутренние механизмы динамики потока ассоциаций; не объяснена целенаправленность мыслительной (интеллектуальной) деятельности человека, ее связь с предметной деятельностью.
В то же время, изучены некоторые аспекты ***ассоциаций*** – явлений, действительно играющих заметную роль в психической деятельности человека. Понятие ***ассоциация*** использовалось в рамках других школ психологии.

Вюрцбургская школа психологии мышления (Германия, XIX-XX вв.).
(мышление как действие)
Переход к экспериментальному интроспективному изучению мышления (комментируются психические феномены, испытуемые – *психологи*). Замысел – использовать интроспективный метод для описания в экспериментальных условиях мыслительных процессов.
Существенное достижение – изменение взглядов на мышление:
мышление – решение задач;
решаемые задачи – важны для человека;
необходим лабораторно-экспериментальный анализ мышления.
Ключевые понятия: ***задача***, ***представление цели***, ***детерминирующая тенденция*** (придает мышлению целенаправленный характер), ***установка*** – регулятор мыслительной деятельности у принявшего задачу, определяет ход мышления, регулирует (в соответствии с задачей) его содержание.
Экспериментальные исследования:

Опыты Марбе (1901 г.): сравнить веса предметов и *прокомментировать*, как выбирали.
Опыты Уатта и Мессера (1905 г.): решить арифметическую/логическую задачу и *проследить путь*, который привел к решению; в ассоциативном эксперименте *проследить*, какие психические процессы связывают стимул (исходное понятие) и реакцию (понятие, связанное с ним ассоциацией).
Обобщение результатов:
чувственно-образные феномены (прослеживаемые с помощью самонаблюдения) и ассоциации не определяют итоговую реакцию, мышление не сводится к ассоциациям;
в мышлении есть другое содержание, у мышления есть другие детерминанты;
основные факторы мышления находятся вне "непосредственно данного" (ощущений, интроспекции), мышление управляется не ассоциативными связями, а тем, что задано;
контекст мышления – ***задача*** (Уатт), ***детерминирующая тенденция*** (Ах), ***установка***.

Концепция Зельца (XX вв.).
(мышление как функционирование интеллектуальных операций)
Серьезный анализ начальных этапов решения задачи (*проблемный комплекс*); введение в рассмотрение *интеллектуальных операций*: ***антиципация*** (выявление отношений: известное ↔ искомое), ***дополнение комплекса***, ***абстракция***, ***репродукция сходства***.

Бихевиоризм (США, XIX-XX вв.).
(мышление как поведение)
Главный момент концепции: предметом психологии должно стать ***поведение***, только тогда возможно объективное исследование психической деятельности.
И.П.Павлов: *"Деловой американский ум, обращаясь к практике жизни, нашел, что важнее точно знать внешнее поведение человека, чем гадать об его внутреннем состоянии со всеми его комбинациями и колебаниями"*.
Торндайк:
- опыты с *проблемными ящиками* (дверь открывается изнутри, "испытуемые"– мыши, крысы);
- мышление можно изучать без обращения к идеям и другим явлениям сознания;
- ассоциативные связи (которые можно объективно изучать) – связи между *движениями и ситуациями*, *ситуациями и реакциями* (на эти ситуации);
- ассоциации могут возникать в результате "слепого поиска" решения, выбора удачного варианта, а затем укрепления и упрочения ассоциативных связей, т.е. ***научения*** (какая реакция R из нескольких возможных связана с ситуацией S);
- существуют "законы научения" (установления связи S ↔ R):
закон упражнения (R зависит от частоты, силы, длительности повторений ситуации S),
закон эффекта (выбирается R, сопровождающаяся приятными ощущениями),
закон ассоциативного сдвига (если S1 ↔ R и S1 встречается совместно с ситуацией S2, то возможно образование связи S2 ↔ R).
Общая схема поведения: исходный пункт – проблемная ситуация; организм противостоит ей как целое, активно действует в поисках выбора, выучивается путем упражнения.
Метод проб и ошибок (современные взгляды на его место в мышлении); "Мартышка и очки", "Дурная голова ногам покоя не дает" и т.п.
Уотсон:
- основа поведения – "стимул – реакция" (S ↔ R);
- все факты Сознания (Торндайк предлагал исключить их из рассмотрения) должны объясняться с позиций бихевиоризма – как реакции на раздражители;
- интеллект – поведение, направленное на решение задач путем отбора движений, оказавшихся удачными;
- в мышление включена внутренняя речь (*"человек мыслит гортанью"*).
Толмен, Халл, Скинер (необихевиоризм, субъективный бихевиоризм):
- существуют медиаторы поведения (М), это не фикция, а реальные факторы поведения (хотя их трудно изучать объективными методами; нужно рассматривать не двойку (S ↔ R), а S ↔ M ↔ R;
- закон упражнения нужно трактовать по-новому
результат – образование определенной познавательной структуры/картины/карты (например, у крысы при поиске пути в лабиринте формируется схема лабиринта, а не совокупность двигательных навыков);
- интеграторы поведения – центральные процессы, а не движения;
- при решении задачи важна структура задачи, а не шаблонные приемы решения.
Оценка концепции: предложены объективные методы экспериментального исследования; в сферу изучения включен анализ связи телесной реакции с материальным стимулом; введено понятие ***поведения***, но проведен общебиологический (а не собственно психологический) его анализ; в целом для концепции характерна биологизация человеческой деятельности; в ходе развития концепции пришлось отказаться от некоторых наиболее одиозных положений, сделан "шаг навстречу" оппонентам.

Когнитивная психология (XX вв.).
(мышление как процессы обработки информации)
Классическая работа **Миллер, Галантер, Прибрам** "Планы и структура поведения" (на рус.яз.- 1965 г.).
- мышление – решение задач;
- активность связана с приобретением, организацией и использованием знаний;

- исследование аналогий *мозг человека* ↔ *ЭВМ*, моделирование мышления на ЭВМ;
- связь мышления с познавательными (когнитивными) процессами.

Гештальтпсихология (XX вв.).
(мышление как переструктурирование ситуации)
"Гештальт" – образ, система. Объективно, **Гештальтпсихология** знаменует внедрение системного подхода в психологические исследования.
Вертгеймер: *"Имеются целостности, чье поведение не детерминируется поведением индивидуальных элементов, из которых они состоят, но где сами частные процессы детерминируются внутренней природой целого"*.
В основе экспериментальный анализ процессов восприятия, его обобщение:
Рубин, **Катц** – анализ зрительных и осязательных восприятий, формулировка законов ***константности***, ***прегнантности*** восприятия, ***транспозиции***, исследование феномена ***Фигуры и Фона***;
Вертгеймер, Келлер, Коффка – *φ-феномен* (воспринимается динамичное целое, а не соединение отдельных сенсорных элементов).
Келлер – попытка перестроить психологию по аналогии с современной физикой (Ньютон → Планк);
рассмотрение гештальтов трех уровней (физического, физиологического и психического) и признание (ошибочное) изоморфизма всех этих уровней;
знаменитые опыты с обезьянами, анализ ***инсайта*** (интуитивного озарения).
Вертгеймер, Келлер, Дункер – исследование продуктивного мышления; выделение мыслительных операций: ***реорганизация***, ***центрирование***, ***группировка***; фиксация отрицательного влияния привычного способа восприятия структурных отношений между компонентами задачи/проблемной ситуации на ее продуктивное решение.
Оценка концепции (достижения): внедрение системного подхода; новая экспериментальная практика – объект рассматривается как целостный, динамичный, трансформируемый чувственный образ; анализ продуктивного мышления и адекватных мыслительных операций.
Важные моменты во взглядах на решение задач: поле восприятия обретает новую структуру, адекватную проблемной ситуации, безразличные до этого предметы приобретают *функциональную ценность* средств решения задачи.

Теория мышления Рубинштейна (XX вв., 50-е гг., СССР).
Задача теории мышления – исследование мышления как деятельности, в основе которой лежит взаимодействие субъекта и объекта.
Мышление – процесс, использующий механизмы анализа, синтеза, обобщения, абстракции; применение знаний зависит от хода мыслительного процесса.
Анализ через синтез (один из главных механизмов продуктивного мышления) – объект в процессе мышления включается в новые системы отношений (*синтез*), выступает в новых качествах, что дает возможность узнать его новые свойства, фиксируемые в новых понятиях (*анализ*); *"из объекта как бы вычерпывается все новое содержание"* (Примеры: опыты Секкея, урезанная шахматная доска).
Задача обучения – формирование продуктивного мышления.

Теория мышления Гальперина (XX вв., 50-е гг., СССР).
В основе – ***идея интериоризации***: *"предметное действие переносится во внутренний, умственный план, а затем ... во внутреннюю речь"*; умственная деятельность – последовательное, поэтапное отражение во все более сокращенном виде материальной деятельности человека.
Мышление – система (и процесс ее функционирования) интериоризованных операций.
Теория мышления – теория о поэтапном формировании умственных действий и методах обучения им.

Теория интеллекта Жана Пиаже (XX вв., Швейцария).
(Ж.Пиаже – один из наиболее крупных и уважаемых психологов, крупнейший специалист в области детской психологии, автор известных работ по экспериментальной психофизиологии; теория интеллекта Жана Пиаже – одна из наиболее интересных теорий мышления и его развития).
Основные аспекты (положения) Теории:
Интеллект определяется в контексте анализа поведения (взаимодействий: субъект ↔ внешний мир); интеллект – форма когнитивного/познавательного аспекта поведения, функциональное назначение которого – структурирование отношений между человеком/субъектом и средой.
Интеллект обладает адаптивной природой. Адаптация включает в себя ***ассимиляцию*** (усвоение данного материала существующими схемами поведения) и ***аккомодацию*** (приспособление этих схем к новым ситуациям). В интеллектуальной сфере – специфически функциональный характер адаптации.
Суть ***Интеллекта*** в его деятельной природе. Познавать объект – воздействовать на него, динамически его воспроизводить.
Интеллектуальная деятельность производна от материальных действий субъекта; ее элементы – интериоризованные действия. Они являются ***операциями*** – координируются между собой, образуя *обратимые, устойчивые* и вместе с тем *подвижные* целостные структуры.
Интеллект *"продолжает и завершает совокупность адаптивных процессов"*. Органическая адаптация *"обеспечивает лишь мгновенное, реализующееся в данном месте, а потому и весьма ограниченное равновесие"*. Простейшие когнитивные функции (восприятие, память и др.) *"продолжают это равновесие как в пространстве, так и во времени"*. Но лишь один интеллект "*тяготеет к тотальному равновесию, стремясь к тому, чтобы*

ассимилировать всю совокупность действительности и чтобы аккомодировать к ней действие, которое он освобождает от рабского подчинения изначальным «здесь» и «теперь»".

Психологическое развитие мыслительных операций (учение о ***Стадиальном развитии интеллекта***):

1. ***Сенсо-моторный период*** (ребенок в возрасте 0-2 года): действия еще не перенесены во внутренний план, начинают формироваться представления о константности предмета.
2. ***Дооперациональный период*** (2-7 лет): появляются язык (→ возможность интериоризации действия в мысли), осознание прошлого, способность мысленного разделения объекта на части и т.п.; отсутствует представление о законах сохранения (эксперименты с переливанием жидкости и др.).
3. ***Период конкретных операций*** (7-11 лет): появляются формальные операции (классификация: *орел* < *птица* < *животное* и др.); операции еще не объединены в единое целое; формируются представления о законах сохранения: вещества (7-8 лет), массы (8-9 лет), объема (9-11 лет) – эксперименты с *глиняной колбаской*.
4. ***Период формирования операций*** (11-15 лет) гипотетико-дедуктивные рассуждения (могут выдвинуть гипотезу, обосновать ее), формальные рассуждения (тест Белларда), операции начинают объединяться в целостные структуры.

Программное обеспечение работ по ИИ

Экспериментальный и эволюционный характер разработок систем ИИ, требования к программному обеспечению

Система Искусственного интеллекта (**Интеллектуальная Система** = **ИС**) – программно-аппаратный комплекс, способный к решению таких задач, решение которых предполагает применение человеком своих интеллектуальных способностей.
Создание ИС – процесс сложный, он предполагает моделирование интеллектуальной деятельности человека (а мы знаем, что она чрезвычайно сложна и слабо изучена)
→
приходится много экспериментировать, создавать все новые и новые варианты, версии (иногда очень сильно отличающиеся друг от друга)
→
нужны языки высокого/сверхвысокого уровня для быстрого прототипирования

Особенности задач ИИ (с точки зрения программирования):

1. сложные и динамически меняющиеся структуры данных;
2. большие по объему хранилища данных (базы знаний) и средства эффективной работы с ними;
3. символьные (в основном) данные;
4. модели, отражающие состояние проблемной среды;
5. переборные алгоритмы;
6. алгоритмы поиска по образцу;
7. гибкие структуры управления.

Языки, реально используемые в работах по ИИ:

экспертные системы-оболочки	- EMYCIN
лиспоподобные языки высокого уровня	- различные диалекты/версии Лиспа
языки логического программирования	- Пролог
языки объектно-ориентированного программирования	- SMALL-TALK
языки на основе фреймов	- KRL
универсальные языки программирования	- C/C++, Java, Паскаль
языки, ориентированные на доступ (используют специальные процедуры - ***демоны***, которые активируются при получении СООБЩЕНИЙ)	- LOOPS

Пример: ЭС моделирования воздушного боя SWIRL
(LOOPS) ***движущиеся объекты***: бомбардировщик,
истребитель,
ракета,
самолет системы AWACS
радары: радар зенитно-ракетного комплекса,
самолет системы AWACS
. . .
Самолет системы AWACS обнаружил самолет-нарушитель

→ посылка СООБЩЕНИЯ всем центрам дешифровки
→ посылка СООБЩЕНИЯ командному центру
→ посылка СООБЩЕНИЯ истребителям/зенитно-ракетным комплексам

Другие средства программной поддержки

Редакторы баз знаний:
штатные текстовые редакторы
автоматические журналы
средства синтаксического контроля
средства контроля содержания
Средства отладки: трассировка, остановы, автоматизация тестирования

Язык ЛИСП

В языке Лисп все действия описываются в виде *функций*. С точки зрения синтаксиса, обращения к функциям, как и обрабатываемые ими данные, представляют собой так называемые *S-выражения*. В простейшем случае S-выражением является *атом* (идентификатор или число), в более сложном – *список*, который представляет собой заключенную в круглые скобки последовательность элементов списка. Лисповские списки имеют рекурсивную структуру, поскольку элементом списка может быть произвольное S-выражение – как атом, так и список. Примеры списков:

(( ) (a b 1 (c)) class)
(45 89 (( ) ( )))
(AT ROBOT (a 6))
(1 2 # 4 5 6 7 8 3).

Пустой список `()` имеет в Лиспе и другое обозначение – `nil`, причем он одновременно относится и к атомам, и к спискам.

Некоторые S-выражения можно вычислять, такие выражения называются *формами*. Простейшими формами являются числа, идентификаторы `T` и `nil`, а также переменные с уже определенными значениями; значением такой формы служит соответственно само число, сам идентификатор `T` и `nil` или текущее значение переменной.

Формой является также обращение к функции, т.е. список вида

$(f\ a_1\ a_2\ \ldots\ a_n)$ $(n \geq 0)$

где f - имя функции, а a_i - ее аргументы, которые для *обычной функции* должны быть формами. В Лиспе есть и *специальные функции*, у которых может быть произвольное число аргументов и аргументы которых либо не вычисляются, либо вычисляются особым образом.

Программа на Лиспе представляет собой последовательность форм, и ее выполнение заключается в последовательном вычислении этих форм. Как правило, в начале программы определяются новые функции, а затем следуют обращения к ним.

Языка Лисп предоставляет большой набор *встроенных* (стандартных) *функций*, на основе которых составляется пользовательская программа. Ниже кратко описаны некоторые из встроенных функций. Все эти функции входят в наиболее известные версии Лиспа - *Common Lisp* и *MuLisp* [Семенов].

Отметим, что комментарием в Лиспе считается любой текст между двумя точками с запятой (;)

Определение новых функций

Для этого служит встроенная функция defun, к которой возможно любое из следующих обращений:

(defun f (lambda ($v_1\ v_2\ \ldots\ v_n$) e)) $(n \geq 0)$
(defun f ($v_1\ v_2\ \ldots\ v_n$) e).

Значением этой функциональной формы является имя определяемой функции, т.е. f. Побочным же эффектом вычисления этой формы будет определение новой лисповской функции с именем f, аргументами (формальными параметрами) v_i и телом e - формой, зависящей от v_i.

Отметим, что таким образом определяется обычная лисп-функция, т.е. функция с фиксированным количеством аргументов, которые всегда вычисляются при обращении к ней.

При последующем обращении в программе к новой функции:

$(f\ a_1\ a_2\ \ldots\ a_n)$

сначала вычисляются аргументы (фактические параметры) a_i, затем вводятся локальные переменные v_i, которым присваиваются значения соответствующих аргументов a_i, а далее вычисляется форма-тело e при этих значениях переменных v_i, после чего эти переменные уничтожаются. Вычисленное значение формы становится значением функции f.

Основные операции над списками

(car l)

Значением аргумента l должен быть непустой список, тогда значением функции является первый элемент (верхнего уровня) этого списка. Например: (car '(AT ROBOT (a 6))) → AT.

Примечание: символ ' имеет особое значение, он показывает, что следующее за ним S-выражение берется в том виде, в каком оно записано (значение его – как формы – не вычисляется) – см. ниже.

(cdr *l*)

Значением аргумента *l* должен быть непустой список, и значением функции является «хвост» этого списка, т.е. список, полученный отбрасыванием первого элемента. Например, (cdr '(AT ROBOT (a 6))) → (ROBOT (a 6)).

Кроме этих двух функций, называемых селекторами элементов списка, часто используются функции, эквивалентные определенной их суперпозиции. Имена всех таких функций начинаются на букву c, а заканчиваются на букву r, между ними же может стоять произвольная комбинация из не более чем пяти букв a и d, причем буква a означает наличие в суперпозиции функции car, а буква d – функции cdr . Сама же последовательность букв соответствует порядку следования функций car и cdr в эквивалентной суперпозиции.
Например, (caddr *l*) ≡ (car (cdr (cdr *l*))) .
Предполагается, что список-аргумент *l* всех этих функций, так же как и ниже описываемой функции nth, содержит необходимое число элементов (в противном случае вычисление программы прерывается сообщением об ошибке).

(nth *n l*)

Значением аргумента *n* должно быть положительное целое число (обозначим его *N*), а значением аргумента *l* – список. Значением функции является *N*-й от начала элемент этого списка. Например,
(nth 2 '(AT ROBOT (a 6))) → ROBOT.

(last *l*)

Функция выбирает последний (от начала) элемент списка, являющегося значением ее аргумента. Например, (last '(AT ROBOT (a 6))) → (a 6).

(cons *e l*)

В отличие от предыдущих функций эта функция является конструктором, т.е. строит новый список, который и выдает в качестве своего результата. Первым элементом этого списка будет значение аргумента *e*, а хвостом списка – значение аргумента *l*. Например, (cons '(A B C) '(D E F)) →
((A B C) D E F).

(append l_1 l_2)

Эта функция осуществляет конкатенацию двух списков, являющихся значением ее аргументов. Например, (append '(A B C) '(D E F)) → (A B C D E F).

(list e_1 e_2 . . . e_n) ($n \geq 1$)

Эта функция имеет произвольное количество аргументов, из значений которых она строит список (количество элементов на верхнем уровне результирующего списка равно количеству аргументов).

Например, если переменная X имеет своим значением список (p (q r)), а переменная Y – список (t), то
значение формы (cons X Y) равно ((p (q r)) t),
значение формы (append X Y) равно (p (q r) t),
значение формы (list X Y) равно ((p (q r)) (t)).

Арифметические функции

(length *l*)

Значением аргумента *l* должен быть список, функция вычисляет количество элементов (верхнего уровня) этого списка. К примеру, значение (length *X*) равно 2, если переменная X имеет значение (p(q r)).
Значениями аргументов нижеследующих функций должны быть числа, над которыми производятся арифметические операции.

(add1 *n*)

Функция прибавляет число 1 к числу-аргументу и выдает результат в качестве своего значения.

(sub1 *n*)

Эта функция вычитает 1 из значения своего аргумента и выдает результат в качестве своего значения.

(+ n_1 n_2)

Значением функции является сумма значений ее аргументов.

(- n_1 n_2)

Значением этой функции является разность значений ее аргументов.

(mod n_1 n_2)

Функция выполняет деление нацело значение первого аргумента на значение второго, и результат выдает в качестве своего значения.

(rem n_1 n_2)

Результат вычисления этой функции – остаток от деления первого числа на второе.

Предикаты

Предикатом обычно называется форма, значение которой рассматривается как логическое значение «истина» или «ложь». Особенностью языка Лисп является то, что «ложью» считается пустой список, записываемый как `()` или `nil`, а «истиной» – любое другое выражение (часто атом `T`).

(null *e*)

Эта функция проверяет, является ли значение ее аргумента пустым списком: если да, то значение функции равно T, иначе – `()`.

(eq e_1 e_2)

Функция сравнивает значения своих аргументов, которые должны быть атомами-идентификаторами. В случае их совпадения (идентичности) значение функции равно T, иначе – `()`.

(eql e_1 e_2)

В отличие от предыдущей функции, данная функция сравнивает значения своих аргументов, которыми могут быть не только атомы-идентификаторы, но и атомы-числа. Если они равны, то значение функции равно `T`, иначе – ( ).

(neq e_1 e_2)

Аналог предыдущей функции, но значения аргументов сравниваются на «не равно».

(equal e_1 e_2)

Функция производит сравнение двух произвольных S-выражений – значений своих аргументов. Если они равны (имеют одинаковую структуру и состоят из одинаковых атомов), то значение функции равно T, иначе – `()`.

(member *a l*)

Значением первого аргумента должен быть атом, а второго – список. Функция производит поиск заданного атома на верхнем его уровне заданного списка. В случае успеха поиска значение функции равно `T`, иначе – `()`.

(gt n_1 n_2) или ($> n_1$ n_2)

Значениями аргументов этой функции должны быть числа. Если первое из них больше второго, то значение функции равно `T`, иначе – `()`.

(lt n_1 n_2) или ($< n_1$ n_2)

Аналог предыдущей функции, но числа сравниваются на «меньше».

Логические функции

Так называются три функции, реализующие основные логические операции.

(not *e*)

Эта функция, реализующая «отрицание», является дубликатом функции null: если значение аргумента равно `()` («ложь»), то функция выдает результат T («истина»), а при любом другом значении аргумента выдает результат `()`.

(and e_1 e_2 ... e_k) (k≥1)

Это «конъюнкция». Функция по очереди вычисляет свои аргументы. Если значение очередного из них равно `()` («ложь»), то функция, не вычисляя оставшиеся аргументы, заканчивает свою работу со значением `()`, а иначе переходит к вычислению следующего аргумента. Если функция дошла до вычисления последнего аргумента, то с его значением она и заканчивает свою работу.

(or e_1 e_2 ... e_k) (k≥1)

Это «дизъюнкция». Функция по очереди вычисляет свои аргументы. Если значение очередного из них не равно `()` («ложь»), то функция, не вычисляя оставшиеся аргументы, заканчивает свою работу со значением этого аргумента, в противном случае она переходит к вычислению следующего аргумента. Если функция дошла до вычисления последнего аргумента, то с его значением она и заканчивает свою работу.

К числу логических функций можно отнести и лисповское условное выражение:

(cond (p_1 $e_{1,1}$ $e_{1,2}$ ... $e_{1,k1}$) … (p_n $e_{n,1}$ $e_{n,2}$ ... $e_{n,kn}$)) (n≥1, k_i≥1)

Функция cond последовательно вычисляет первые элементы своих аргументов – обращения к предикатам p_i. Если все они имеют значение `()` («ложь»), тогда функция заканчивает свою работу с этим же значением. Но если был обнаружен предикат p_i, значение которого отлично от `()`, т.е. он имеет значение «истина», тогда функция cond уже

не будет рассматривать остальные предикаты, а последовательно вычислит формы $e_{i,j}$ из этого i-го аргумента и со значением последнего из них закончит свою работу. Заметим, что поскольку значения предыдущих форм из этого аргумента нигде не запоминаются, то в качестве этих форм имеет смысл использовать только такие, которые имеют побочный эффект, например, функции присваивания значений переменным или печати.

Специальные функции

(quote *e*) или '*e*

Это функция блокировки вычислений: она выдает в качестве значения свой аргумент, не вычисляя его. Например, значением формы '(car (2)) является само выражение (car (2)).

(gensym)

Это функция генерации уникальных атомов (символов): при каждом обращении к ней выдается новый атом-идентификатор. Этот идентификатор получается склеиванием специального префикса и очередного номера (целого числа). Префикс и целое число, от которого начинается нумерация генерируемых атомов, могут быть установлены заранее, как, например, в языке MuLisp:

(setq *gensym-prefix* 'S) (setq *gensym-count* 2)

После этого при последовательных обращениях к функции gensym она будет выдавать атомы S2, S3, S4 и т.д.

Блочная и связанные с ней функции

(prog (v_1 v_2 … v_n) e_1 e_2 … e_k) ($n \geq 0$, $k \geq 1$)

Эту специальную функцию называют «блочной», поскольку ее вычисление напоминает выполнение блоков в других языках программирования. Вычисление функции начинается с того, что вводятся локальные переменные v_i, перечисленные в ее первом аргументе, и всем им в качестве начального значения присваивается пустой список `nil`. После этого функция последовательно вычисляет остальные свои аргументы – формы e_i. Вычислив последнюю из них, функция prog заканчивает работу со значением этой формы, уничтожив перед этим все свои локальные переменные v_i .

Вычисленные значения всех форм e_i, кроме последней, нигде не запоминаются, поэтому имеет смысл использовать в качестве них только функции с побочным эффектом. Некоторые из этих функций перечислены ниже.

В качестве одной из форм e_i может быть записан атом-идентификатор, в этом случае он не вычисляется, а трактуется как метка, на которую будет производиться переход внутри этого блока (функцией go).

(return *e*)

Это функция досрочного выхода из блока. Она может использоваться только внутри блочной функции prog, поскольку завершает вычисление ближайшей объемлющей блочной функции, устанавливая ее значение равным значению аргумента *e*.

(go *e*)

Функция реализует переход по метке. Аргумент ее не вычисляется, в качестве ее аргумента должен быть задан идентификатор – одна из меток ближайшей объемлющей блочной функции. Функция go полностью завершает вычисление той формы этой блочной функции, в которую она входит (на любом уровне), и осуществляет переход на вычисление формы, непосредственно следующей за указанной меткой.

(setq *v e*)

Это аналог оператора присваивания. В качестве аргумента *v* (он не вычисляется) должно быть задано имя переменной, существующей в данный момент. Функция присваивает этой переменной новое значение – вычисленное значение формы *e*. Это же значение является значением и самой функции setq, однако оно, как правило, не используется.

Следующие две особые функции используются для упрощения записи часто используемых конструкций (setq *V* (cdr *V*)) и (setq *V* (cons (*e V*)).

(pop *v*)

Аргументом этой функции (он не вычисляется) должно быть имя переменной, существующей в данный момент и имеющей своим значением непустой список. Хвост этого непустого списка присваивается в качестве нового значения указанной переменной, а также выступает в качестве значения самой функции pop.

(push *e v*)

В качестве второго аргумента этой функции (он не вычисляется) должно быть задано имя переменной, в качестве первого – произвольная форма. Функция вычисляет эту форму и строит новый список, первый элемент которого – вычисленное значение, а хвост – список, являющийся значением переменной *v*. Результирующий список становится новым значением переменной *v* и значением самой функции push.

Например, если переменная X имеет значение (d (e) g), а переменная U – значение (1 2), то значение формы (pop X) равно ((e) g), а значение (push U X) равно ((1 2) d (e) g).

Основным средством реализации циклических программ в Лиспе является рекурсия.
Рассмотрим примеры рекурсивных программ на Лиспе:
(listp *l*) – функция-предикат, проверяющая является ли значение ее аргумента списком (на верхнем уровне). Если да, то значение функции равно T, иначе – ().

```
(defun listp (lambda (x)
        (cond ((null x) T)
                ((atom x) ( ))
                ( T (listp (cdr x))) )))
```

(memb *a l*) - функция ищет атом, являющийся значением первого ее аргумента, в списке (на верхнем его уровне), являющемся значением второго аргумента. В случае успеха поиска значение функции равно T, иначе – ().

```
(defun memb (lambda (a l)
        (cond ((null l) nil)
                ((eq a (car l)) T)
                (T (memb a (cdr l)) ) ))
```

(out *a l*) - функция удаляет из списка, являющегося значением ее второго аргумента, все вхождения (на верхнем) атома, являющегося значением первого аргумента.

```
(defun out (lambda (a l)
        (cond ((null l) nil)
                ((eq a (car l)) (out a (cdr l)))
                (T (cons (car l) (out a (cdr l))))) ))
```

(equal e_1 e_2) – функция, сравнивающая два произвольных S-выражения – значения своих аргументов. Если они равны (имеют одинаковую структуру и состоят из одинаковых атомов), то значение функции равно T, иначе – ().

```
(defun equal (lambda (x y)
                (cond ((atom x) (eq x y))
                        ((atom y) ( ))
                        ((equal (car x) (car y)) (equal (cdr x) (cdr y)))
                       ( T ( ) ) )))
```

Язык Плэнер

Плэнер – один из потомков языка Лисп, созданный специально для реализации систем ИИ. Основные объекты языка (как и в Лиспе) – списки и атомы. Плэнер – очень громоздкий и сложный (для реализации и изучения) язык не нашедший, к сожалению, более или менее широкого практического применения.

Вспомним «Особенности задач ИИ (с точки зрения программирования)»:

1. сложные и динамически меняющиеся структуры данных;
2. большие по объему хранилища данных (базы знаний) и средства эффективной работы с ними;
3. символьные (в основном) данные;
4. модели, отражающие состояние проблемной среды;
5. переборные алгоритмы;
6. алгоритмы поиска по образцу;
7. гибкие структуры управления.

Основные составляющие языка Плэнер:

- средства для работы со списками (*лисповская часть*, используется для работы со списками)
- плэнерская база данных (используется для моделирования ситуаций проблемной среды)
- встроенный режим возвратов (реализует один из основных методов перебора – бэктрекинг)
- аппарат работы с образцами (используется для анализа структуры списков)
- аппарат теорем (используется при планировании решения)

позволяют достаточно эффективно (и быстро) реализовать соответствующие возможности.

Три типа процедур языка Плэнер:

- функции;
- сопоставители (для работы с образцами);
- теоремы (процедуры, вызываемые по образцу).

Примеры конструкций и процедур Плэнера:

Обращение к функции:
[ELEM -2 (A B C D)] → C
Выдать второй (2), считая с конца списка (знак "минус" при 2), элемент списка (A B C D).

Сопоставление образца с выражением:
[IS (*X ПРИШЕЛ !*Y) (САША ПРИШЕЛ ИЗ МАГАЗИНА)] → T
Сопоставление успешно, то есть образец (*X ПРИШЕЛ !*Y) соответствует выражению (САША ПРИШЕЛ ИЗ МАГАЗИНА). Побочный эффект: переменная X получает значение САША, а переменная Y получает значение (ИЗ МАГАЗИНА)

Утверждение плэнерской базы данных:
(BOX A (3 7)) - ящик с названием A находится на плоском квадратном поле с пронумерованными клетками в третьей строке (по горизонтали) и в седьмом столбце (по вертикали).

Запись утверждения в плэнерскую базу данных:
[ASSERT (BOX A) (WITH COL RED)] – в базу данных записывается утверждение (BOX A), то есть "A является ящиком"; записывается также, что "цвет" (COL) этого ящика "красный" (RED).

Поиск утверждения в плэнерской базе данных:
[SEARCH (BOX [ ]) (TEST COL [NON GREEN])] – в базе данных ищется утверждение о некотором ящике, цвет которого не (NON) зеленый (GREEN); возможный ответ (результат поиска) – (BOX A).

Определение плэнерской теоремы:
```
[DEFINE ОСВОБОДИЛИ (ERASING (X)
                    (=ЗАНЯТ= *X)
    [ASSERT (=СВОБОДЕН= .X)])]
```
Если из базы данных вычеркивается утверждение о том, что "поверхность некоторого кубика X занята" (если так, то на него нельзя поставить другой кубик), то автоматически будет вызвана и выполнена эта теорема. Она запишет в базу данных утверждение: "поверхность этого кубика X освободилась" (теперь на него можно поставить другой кубик).

Перейдем к более детальному рассмотрению основных составляющих Плэнера.

Основные объекты языка, средства для работы со списками
Выражения:
- атомарные:
  - обращения к переменным
  - атомы: идентификаторы, числа, строки
- списковые:
  - L-списки (списки в круглых скобках: **(** и **)**)
  - P-списки (списки в квадратных скобках: **[** и **]**)
  - S-списки (списки в «уголках»: **<** и **>**)

Ограничители: <пробел>, **(** , **)** , **[** , **]** , **<** , **>**; цифры; буквы; специальные литеры **+** , **–** , **.** , ***** , **:** , **!** .
Префиксы: **.** , ***** , **:** - простое обращение к переменным; **!.** , **!*** , **!:** - сегментное обращение к переменным.
Простые формы:
- атомы (значением атома является сам этот атом),
- обращения к переменным с префиксом **.** ,
- обращения к константам с префиксом **:** ,
- L-списки (значением L-списка является список из значений его элементов),
- P-списки (обращения к процедурам).

Сегментные формы:
- обращения к переменным с префиксом **!.** ,
- обращения к константам с префиксом **!:** ,
- S-списки (сегментные обращения к процедурам).

В языке Плэнер, как и в Лиспе, программа и обрабатываемые ею данные строятся из *выражений*, к которым относятся: *атомы*, *обращения к переменным*, состоящие из префикса и имени переменной (например, .X) и *списки* всех трех видов. Некоторые выражения (*формы*) можно вычислять, получая в результате значения (ими являются выражения). Программа на Плэнере представляет собой последовательность форм, ее выполнение заключается в вычислении этих форм.

В языке имеется много встроенных (стандартных) функций.

Определение новых функций

Для определения новой функции следует обратиться к встроенной функции define:

[define *f dexp*]

Вычисление функции define в качестве побочного эффекта приводит к появлению в программе новой функции с именем *f* и т.н. определяющим выражением *dexp*, которое должно иметь вид

(lambda (v_1 v_2 ... v_n) e) (n≥0)

где v_i - формальные параметры новой функции, а e - форма, зависящая от v_i.

При последующем обращении к этой новой функции

[f a_1 a_2 ... a_n]

сначала вычисляются аргументы (фактические параметры) a_i, затем вводятся локальные переменные v_i, которым присваиваются значения соответствующих аргументов a_i, и далее вычисляется форма e при этих значениях переменных v_i, после чего эти переменные уничтожаются. Значение данной формы становится значением функции f при аргументах a_i.

Операции над списками

[elem n l]

Значением аргумента n должно быть ненулевое целое число (обозначим его N), а значением второго аргумента - список (L) с любыми скобками. Значением функции является N-й от начала элемент списка L, если N>0, или |N|-й от конца элемент этого списка, если N<0. В случае, когда в качестве n явно указано число, имя функции в обращении можно опустить; например, [1 .X] - это сокращение от [elem 1 .X], т.е. выделяется первый от начала элемент списка, являющего значением переменной X.

[rest n l]

Значением аргумента n должно быть ненулевое целое число (N), а значением второго аргумента - список (L) с любыми скобками. Значением функции является результат отбрасывания N первых элементов списка L, если N>0, или |N|-й последних его элементов, если N<0.

[head n l]

В такой же ситуации значением функции является список из N первых элементов списка L, если N>0, или |N|-й последних его элементов, если N<0.

Пример: [rest 2 [head 5 (1 2 3 4 5 6)]] → (3 4 5)

Если требуется вычислить список в круглых скобках

(e_1 e_2 ... e_n) (n≥0)

т.е. если он рассматривается как форма, то все его элементы должны быть формами. Значением такого списка является список (с круглыми скобками) из значений его элементов. Например, если переменная X имеет значение (a b), то значением списка (.X c [-1 .X]) является список ((a b) c b).

В таких списках можно использовать *сегментные формы* (сегментные обращения к переменным и сегментные обращения к процедурам, <elem 5 .X>). Сегментная форма вычисляется как обычная (простая) форма, но затем у ее значения, если это список, отбрасываются внешние скобки, и полученная таким образом последовательность элементов поставляется вместо сегментной формы. Например, если переменная X имеет значение (a (b c)), то:

(5 .X ( )) → (5 (a (b c)) ( )),
(5 !.X ( )) → (5 a (b c) ( )),
(!.X !.X) → (a (b c) a (b c)).

Если переменная X имеет значение (1 2), а переменная Y – значение (3 4):

(.X !.Y) → ((1 2) 3 4) – на Лиспе эти действия задаются так: (cons X Y)
(!.X !.Y) → (1 2 3 4) – на Лиспе эти действия задаются так: (append X Y)

Арифметические функции

[+ n_1 n_2 ... n_k] (k≥2)

Значениями аргументов должны быть числа. Результат вычисления функции – их сумма.

[- n_1 n_2]

Значениями обоих аргументов должны быть числа. Значение функции – их разность.

[max n_1 n_2 ... n_k] (k≥2)

Значениями аргументов должны быть числа. Результат вычисления функции – их максимум.

[min n_1 n_2 ... n_k] (k≥2)

Значениями аргументов должны быть числа. Результат вычисления функции – их минимум.

Предикаты

Предикатом называется форма, значение которой рассматривается как логическое значение «истина» или «ложь». При этом в Плэнере «ложью» считается пустой список `()`, а «истиной» – любое другое выражение (чаще всего в роли «истины» выступает атом `T`).

[num *e*]
Эта функция-предикат проверяет, является ли числом значение ее аргумента. Если да, то значение функции равно `T` («истина»), иначе – `()` («ложь»).

[empty *e*]
Функция проверяет, является ли значение ее аргумента пустым списком (с любыми скобками). Если да, то значение функции равно `T`, иначе – `()`.

[eq e_1 e_2]
Функция сравнивает значения своих аргументов. Если они равны, то значение функции – `T` , иначе – `()`.

[neq e_1 e_2]
Аналог, но значения аргументов сравниваются на «не равно».

[memb e l]
Значением функции является номер первого вхождения формы E в список L.

[gt n_1 n_2], [ge n_1 n_2], [lt n_1 n_2] – сравнение, соответственно, на «больше», «больше или равно» и «меньше».

Предикаты, проверяющие состояние переменных

В Плэнере переменные (формальные параметры процедур или локальные переменные блоков) могут находиться в одном из трех состояний:

1) переменная не описана (в некоторой функции встретилась нелокальная переменная, которая не описана в динамически объемлющих процедурах/блоках),
2) переменная описана, но не имеет значения,
3) переменная описана и имеет некоторое значение.

Для анализа таких ситуаций используются предикаты: [bound *v*] и [hasval *v*].
Первый из этих предикатов проверяет, является ли значение его параметра описанной (в динамически объемлющих процедурах/блоках) переменной. Если да, возвращается значение `T`, иначе – `()`.
Значением аргумента предиката hasval должно быть имя переменной, существующей в данный момент. Функция проверяет, имеет ли сейчас эта переменная какое-либо значение или нет. Если имеет, то функция возвращает результат `T`, а иначе - результат `()`.

Функции для работы со списками свойств идентификаторов

Идентификаторы (и некоторые другие объекты) Плэнера могут иметь т.н. *списки свойств* – списки пар: *свойство-значение*. Если идентификатор является именем некоторого объекта проблемной среды (например, ящика, который может перемещаться роботом), в списке свойств могут быть указаны цвет этого ящика, его вес, линейные размеры и т.п.

Для задания списка свойств используется функция [plist *i pl*]. При выполнении обращения к ней идентификатор I получает список свойств PL = (ind_1 val_1 ... ind_n val_n). Отметим, что отсутствие некоторого свойства эквивалентно тому, что это свойство присутствует в списке со значением `()`.
Функция [put i ind val] позволяет изменить значение указанного свойства IND идентификатора I, а функция [get i ind?] – получить весь список свойств I (если параметр ind не задан), или же узнать значение указанного свойства.

Логические функции

Функции «отрицание», «конъюнкция», «дизъюнкция» и «условное выражение» аналогичны соответствующим лисп-функциям:

[not *e*]
[and e_1 e_2 ... e_k] (k≥1)
[or e_1 e_2 ... e_k] (k≥1)
[cond (p_1 $e_{1,1}$ $e_{1,2}$ ... $e_{1,k1}$) … (p_n $e_{n,1}$ $e_{n,2}$ ... $e_{n,kn}$)] (n≥1, k_i≥1)

Блочная и связанные с ней функции

[prog (v_1 v_2 ... v_n) e_1 e_2 ... e_k] (n≥0, k≥1)

Эту функцию называют «блочной», поскольку ее вычисление напоминает выполнение блоков в других языках программирования. Вычисление функции начинается с того, что вводятся локальные переменные, перечисленные в ее первом аргументе. Если v_i – идентификатор (имя), вводится локальная переменная с этим именем и без начального значения. Если же v_i – пара *(id val)*, то вводится локальная переменная с именем *id* и начальным значением *val* (выражение *val* при этом не вычисляется). После этого функция последовательно вычисляет остальные свои аргументы – формы e_i, которые принято называть операторами. Вычислив последний из них, функция с его значением заканчивает работу, уничтожив при этом свои локальные переменные.

Вычисленные значения всех операторов (кроме последнего) нигде не запоминаются, поэтому имеет смысл в качестве таких операторов использовать только функции/операторы с побочным эффектом. Некоторые из этих функций перечислены ниже.

[set *v e*]

Это аналог оператора присваивания. Сначала вычисляются оба аргумента, причем значением аргумента *v* должно быть имя переменной, существующей в данный момент. Функция присваивает этой переменной новое значение – значение аргумента *e*. Это же значение является значением функции set, но оно, как правило, не используется.

[return *e*]

Это оператор выхода из блока. Функция return может использоваться только внутри функции prog, поскольку она завершает вычисление ближайшей объемлющей блочной функции, объявляя ее значением значение своего аргумента *e*.

[go *e*]

Это оператор перехода. Отметим, что если в качестве оператора функции prog указан идентификатор, то он трактуется как метка. Значением аргумента функции go должен быть идентификатор – одна из меток ближайшей объемлющей блочной функции. Функция go полностью завершает выполнение того оператора этой блочной функции, в который она входит (на любом уровне), и передает управление на оператор, следующий за этой меткой.

В качестве операторов можно использовать составной оператор: [do e_1 e_2 ... e_k] (k≥1),
а также операторы цикла (loop, while, until, for). Мы рассмотрим только первый вид оператора цикла:

[loop *x l* e_1 e_2 ... e_k]

При выполнении этого оператора вводится параметр цикла *x*, локализованный внутри loop, затем *x* поочередно получает в качестве значения очередной элемент списка L (слева направо), к которому применяются операторы e_1 e_2 ... e_k.

Пример: [set L (1 2 3))]
[prog ((R ( ))) [loop E .L [set R (.E !.R)] .R]]

- описанный блок переворачивает входной список (значение переменной L), помещая в конец списка R (в начальный момент пустого) элементы L: сначала первый, затем второй, затем третий. Результатом работы prog будет значение переменной R на момент завершения цикла – список (3 2 1).

Константы

Как и в Лиспе, константами в Плэнере называются глобальные перемеенные. Они вводятся так: [cset c v] – константе с именем C присваивается V.

Функции ввода-вывода

В Плэнере имеется достаточно богатый набор функций ввода-вывода, включающий: функции для работы с файловой системой; функции ввода и вывода символов, текста, строк; функции установки цвета фона и символов, управления курсором и др.
Ниже приводится фрагмент программы, использующей некоторые средства ввода-вывода:

```
[open screen get]     файл screen открывается на ввод
[active screen get]   файл screen переводится в состояние "активный файл ввода"
[cset a [read]]           считывается очередное выражение (в данном случае, набираемое на клавиатуре)
                          и его значение присваивается константе a; пусть это выражение – (N O T).
[cset b [rev :a]]         константе b присваевается (T O N).
[open ff put]         файл с именем ff открывается на вывод
[active ff put]           файл ff переводится в состояние "активный файл вывода"
[print :b]                в файл ff "печатается" (записывается) выражение (T O N).
```

Специальные функции

В Плэнере имеется более десятка т.н. специальных функций, реализующих взаимодействие с операционной средой, интерпретатором (перехват синтаксических ошибок с целью их анализа в программе пользователя), а также преобразование типов. Несколько примеров:

[catch e1 e2]

Функция вычисляет E1 и, если вычисление закончилось успешно, с этим значением заканчивает свою работу. Если же в процессе вычисления возникла ошибка, вычисляется значение второго параметра.

[exit e fn n?]

Выход из функции FN со значением E, если параметр n задан, то выход из N+1 динамически вложенных обращений к функции FN.

[exit ( ) ( )]

Частный случай – выход на верхний уровень программы.

[time]

Значение функции – время от начала решения задачи.

[clock]

Дата и астрономическое время.

[atl a]

Функция преобразует идентификатор A в список составляющих его символов. Например, [atl PLAN] → (P L A N).

Образцы и сопоставители

Для анализа данных в Плэнере наряду с традиционными способами (применение функций для работы со списками, логических функций, предикатов, блоков) можно использовать:

- функцию is, реализующую сопоставление *образца* с выражением,
- *сопоставители* – особые процедуры (к которым можно обращаться только в образцах), проверяющие те или иные свойства выражения и его соответствие образцу в целом или отдельному элементу образца.

Соответствующие возможности Плэнера мы рассмотрим кратко и в основном на примерах.

Образец – выражение, которое используется как шаблон при анализе другого выражения (задает структуру и отдельные компоненты этого выражения).
Если выражение удовлетворяет требованиям шаблона, то говорят, что оно *соответствует* образцу, если нет, то *не соответствует*. В первом случае сопоставление образца с выражением *удачно*, во втором – *неудачно*.

Обращение к функции is выглядит так:
[is pat e], где pat – образец, e – выражение (сопоставление производится с E – значением e).
Алгоритм сопоставления в общем случае сложен, он предполагает возвраты. Важно, что удачное сопоставление может сопровождаться побочными эффектами – переменным, входящим в состав образца могут присваиваться в качестве значений соответствующие фрагменты выражения. Это позволяет резко сократить запись сложных действий по анализу данных и выделению важных для работы фрагментов.

Примеры (считаем, что используемые переменные описаны вне обращений к is, например, в каком- то объемлющем блоке):

[is (*X .X) (b b)] → T — сопоставление удачно, X (побочный эффект) получает значение b отметим, что при сопоставлении первого элемента образца (*X) с первым элементом выражения (b) переменная X получила значение b;
второй элемент образца (.X) требует, чтобы рассматривалось текущее значение переменной X (префикс *точка*), т.е. как раз b; поэтому удачным будет и сопоставление второго элемента образца со вторым элементом выражения.

[is (*X .X) (b (b))] → ( ) — сопоставление неудачно;

[is (*X !*Y) (1 2 3)] → T — сопоставление удачно; X:= 1, Y:= (2 3)

[is (*X *Y) (1 2 3)] → T — сопоставление неудачно (образец описывает список длиной два).

Обращение к ***сопоставителю*** внешне выглядит как обращение к функции. Однако: значение не вырабатывается, проверяется соответствие выражения/подвыражения образцу/элементу образца, фактические параметры (аргументы) обычно уточняют вид проверки.
Пример: [list n] - сопоставитель, соответствующий списку из N элементов

[is [list 2] (a b)] → T — сопоставление удачно;

[is [list 2] (a b c)] → ( ) — сопоставление неудачно;

[is (a <list 2> a) (a b b a)] → T — сопоставление удачно (сегмент b b соответствует сегментному обращению к сопоставителю list;

[is (a [list 2] a) (a (b b) a)] → T — сопоставление удачно.

В Плэнере есть около 30 встроенных сопоставителей, в том числе: логические (типа and и or), типа cond, типа prog (вводят локальные переменные).
Сопоставитель [ ] соответствует любому выражению, сопоставитель < > - любому сегменту.

Некоторые встроенные сопоставители

Сопоставители (без параметров): id (идентификатор), num (число), var. (обращение к переменной с префиксом "."), var*, var:, var!., var!*, var!:, atomic (атомарное выражение) – проверяют тип выражения.
[aut pat_1 ... pat_n] – сопоставитель "ИЛИ"; сопоставление с некоторым выражением успешно, если выражение соответствует одному из указанных образцов.
[et pat_1 ... pat_n] – сопоставитель "И"; сопоставление с некоторым выражением успешно, если выражение соответствует каждому из указанных образцов.
[when pat_1 $pat_{1,1}$ $pat_{1,2}$ ... $pat_{1,k1}$) … (pat_n $pat_{n,1}$ $pat_{n,2}$ ... $pat_{n,kn}$)] – "УСЛОВНЫЙ" сопоставитель; если выражение соответствует образцу pat_i, то применяется сопоставитель [et $pat_{i,1}$ $pat_{i,2}$ ... $pat_{i,ki}$].
[star *pat*] – сопоставитель соответствует списку, каждый элемент (верхнего уровня) которого соответствует образцу *pat*.

Определение новых сопоставителей

[define <имя> (kappa <список параметров> pat)]
Пример:
[define pal (kappa ( )
[aut ( ) [list 1] [same (X) (*X <pal> .X)]])]
- рекурсивный сопоставитель pal соответствует любому списку-палиндрому; в нем использованы: логический сопоставитель aut (аналог or), известный нам сопоставитель list, сопоставитель-блок same (вводящий локальную переменную X). Сопоставитель pal проверяет (последовательно): не является ли выражение пустым списком; списком длиной 1; списком, первый и последний элементы которого совпадают, а «середина» удовлетворяет требованиям pal.

Плэнерская база данных

База данных Плэнера не имеет никакого отношения к базам данных как объекту традиционного программного обеспечения (реляционным или сетевым, СУБД). Она похожа на т.н. *доску объявлений*.

Плэнерская база данных – это область памяти системы программирования на Плэнере, в которой хранятся *утверждения*. Утверждение представляет собой произвольный L-список, семантику которого (как и семантику всего наполнения базы данных) определяет пользователь. Рекомендуется использовать базу данных для моделирования динамически меняющейся проблемной среды: текущий набор утверждений отражает текущее состояние проблемной среды, запись или вычеркивание утверждений соответствует происходящим в среде изменениям. Динамика базы данных отражает динамику поиска /планирования решения задач.

Основные операции над базой данных: запись утверждения, вычеркивание утверждение (по образцу), поиск утверждения (по образцу).

Запись

[assert asrt with? rec? else?] – функция assert служит для записи явно заданного (в виде L-списка) утверждения asrt в базу данных. Факультативный параметр with задает список свойств записываемого утверждения (он имеет такую же структуру, что и список свойств идентификатора, но связан с утверждением в целом). Факультативный параметр rec указывает, какие теоремы следует вызвать при записи данного утверждения (некоторые теоремы могут вызываться в этот момент автоматически). Факультативный параметр else содержит рекомендации по поводу того, что следует делать, если записать указанное утверждение в базу данных не удалось (например, потому, что такое утверждение в базе данных уже хранится).
Пример: пусть переменная X имеет значение green
[assert (box A) (with col .X) (else)] - параметр rec в этом примере опущен
При выполнении этого обращения к функции assert происходит следующее:

1. в базу данных записывается утверждение (box A) - имеется ящик A,
2. с ним связывается список свойств (col green) - его цвет – зеленый,
3. если утверждение записать не удалось, то (в соответствии с конкретным видом последнего параметра – else – вырабатывается неуспех.

Поиск

[search pat test?] – основная функция для поиска в базе данных утверждений по образцу pat. Параметр test (факультативный) позволяет задать требования, предъявляемые к списку свойств утверждения.
Функция search ставит развилку; ищет утверждение, соответствующее образцу; проверяет его список свойств (если он не удовлетворяет требованиям параметра test, ищется другое утверждение соответствующее образцу). Если подходящее (соответствует образцу, удовлетворяет требованиям test, параметр test не задан) утверждение найдено, оно является результатом обращения к search; развилка не уничтожается, следовательно, если в динамике до этого обращения к search «доберется» неуспех, начнется поиск новых утверждений, соответствующих pat. Если

найти утверждение, соответствующее pat и удовлетворяющее test, не удалось, развилка отменяется и вырабатывается неуспех.

Пример: пусть после выполнения обращения к assert из предыдущего примера мы задаем:
[search (box [ ]) (test col [non red])]
Результат поиска – утверждение (box A).

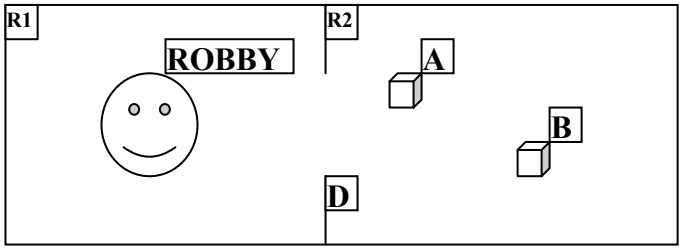
Поиск утверждений можно вести поэтапно – находить утверждения, «частично соответствующие» образцу (функция candidates, обращение к которой имеет вид [candidates pat type?]), а затем более детально анализировать полученный «список кандидатов». Можно искать только одно подходящее утверждение (функция search1).

Вычеркивание

[erase pat test?] – функция для удаления из базы данных утверждений, соответствующих образцу pat и удовлетворяющих требованиям параметра test (если он задан).

Пример: стереть утверждение (box A) со списком свойств (col green) можно так:
[erase (box [ ]) (test col [non red])]

Плэнерская база данных, отображающая состояние проблемной среды:



(room R1)
(room R2)
(conn R1 R2 D)
(door D)
(box A) – (col green)
(box B) – (col red)
(at ROBBY R1)
(at A R2)
(at B R2)

В этой ситуации можно выполнять операции поиска:
[search (at ROBBY R1)] → T
[search (at ROBBY R2)] → ( )
[search (at ROBBY *X)] → T, X:= R1
[find all (Y) .Y [search (room *Y)]] → (R1 R2)

Режим возвратов

В язык Плэнер встроен т.н. *режим возвратов*, который упрощает реализацию различных поисковых алгоритмов, использующих перебор вариантов. Суть этого режима в следующем. В любом месте программы может быть установлена т.н. *развилка*, от которой возможно несколько вариантов продолжения работы программы. Выбирается один из вариантов, и программа продолжает свою работу. Если затем окажется, что этот вариант неуспешен, вырабатывается т.н. *неуспех*, по которому программа автоматически «откатывается» к последней (по времени) развилке. При этом отменяются все изменения (в значениях переменных и т.п.), произведенные на неуспешном пути, и в этой развилке выбирается следующий вариант, после чего программа снова «идет вперед». Если в развилке уже не оказалось нерассмотренных альтернатив, то неуспех возвращает программу к предыдущей развилке.

В каких местах программы ставить развилки и с какими альтернативами, считать ли выбранный путь вычисления неуспешным и когда вырабатывать неуспех – за все это отвечает автор программы. Встроенный же режим возвратов обеспечивает запоминание мест развилок и то, какие альтернативы в них еще не рассматривались, обеспечивает возврат программы по неуспеху к последней развилке и отмену ранее произведенных изменений в значениях переменных.

Ниже перечислены некоторые из встроенных функций языка Плэнер, позволяющих реализовать режим возвратов.

[among *e*]

Значением аргумента должен быть список. Если этот список пуст, функция вырабатывает неуспех, который автоматически возвращает программу к предыдущей ее развилке. Иначе функция запоминает развилку, альтернативами которой является то, что функция в качестве своего значения может выдать любой элемент из этого списка. Вначале функция выдает как свое значение первый элемент списка, завершает на этом работу, и программа продолжает свои вычисления. Но если позже в программе возникнет неуспех, который вернет ее к данной развилке, то функция возобновит свою работу и теперь как свое значение выдаст второй элемент списка, после чего программа снова «идет вперед». И так далее, пока в списке остаются нерассмотренные элементы. После выдачи в качестве своего значения последнего элемента списка, функция уничтожает свою развилку и потому последующий неуспех уже не будет здесь остановлен.

[alt e_1 e_2 ... e_n]

Функция ставит развилку, *i*-я альтернатива которой – вычисление формы e_i. Если вычислить значение удалось, функция заканчивает работу со значением E_i. Перед ввычислением e_n развилка уничтожается.

[fail]

Эта функция вырабатывает неуспех, по которому программа автоматически возвращается к последней (по времени) развилке. (Если развилок нет, то вычисление всего выражения самого верхнего уровня программы считается окончившимся неуспешно.)

[pset *v e*]

Это аналог функции set, т.е. переменной, имя которой является значением аргумента *v*, присваивается новое значение – значение аргумента *e*. Однако, если присваивание, осуществленное функцией set, отменяется при неуспехе, то действие функции pset при неуспехе не будет отменено. Функция pset (и ей подобные) применяется, когда надо сохранить информацию, полученную на неуспешном пути вычисления программы, для последующих путей.

Для управления режимом возвратов помимо использования процедур, результаты которых не отменяются при неуспехе, в Плэнере есть и другие средства:
- уничтожение развилок и/или обратных операторов,
- использование *именованных* развилок.

Пример плэнер-программы, решающей переборную задачу на основе бэктрекинга:
Пусть задан список L положительных целых чисел. Нужно подобрать набор чисел из L (они могут повторяться), сумма которых равна заданному числу N.

```
[define sum (lambda (L N) [prog ( K (M ( )) (S 0))
        A       [set K [among .L]] [set M ( .K !.M)] [set s [+ .K .S]]
                [cond   ([eq .S .N] .M]
                        ([lt .S .N] [go A])
                        (T [fail])] ])]
```

Трассировка выполнения программы при L = (6 3 2 1) и N = 5:
Вход: K – без значения, M = ( ), S = 0

[among (6 3 2 1)] → 6		[among (6 3 2 1)] → 3
[set K 6]	обр.оператор [unassign K]	[set K 3]
[set M (6)]	обр.оператор [set M ( )]	[set M (3)]
[set S 6]	обр.оператор [set S 0]	[set S 3]
S > 5 → неуспех		S < 5 → переход по метке A и новый вызов among: [among (6 3 2 1)] → 6
неуспех	[among (6 3 2 1)] → 3	[among (6 3 2 1)] → 2
	неуспех	S = 5, выход со значением M.

Отметим, что развилки в первом и втором обращениях к функции among остаются. Если в описанное обращение к функции sum откуда-то извне «придет» неуспех, вычисление может возобновиться (при этом будут выбираться не исследованные ранее альтернативы). Например:
[prog (X) [set X [sum (6 3 2 1) 5]] [cond ([neq [length .X 3]] [fail])] .X] → (1 1 3).
Напечатать (поочередно) все решения рассматриваемой задачи можно с помощью такой конструкции:

```
[prog ( ) [alt ( ) [return T]]
          [print [sum (6 3 2 1) 5]] [fail]]
```

А собрать все решения (в списке Y) и затем выдать этот результат на печать можно так:

```
[prog (X (Y ( )) ) [alt ( ) [return .Y]]
                   [set X [sum (6 3 2 1) 5]]
                   [pset Y (!.Y .X)] [fail]]
```

Теоремы

В языке существуют три типа теорем:
- «целевые» (типа conseq),
- «при записи» (типа antec),
- «при вычеркивании» (типа erasing).

Целевые теоремы используются при планировании решения задач.
Если плэнерская база данных может рассматриваться как модель проблемной ситуации, то набор целевых теорем – как набор средств решения соответствующей задачи. Отобранный и упорядоченный набор теорем может трактоваться как план решения: отдельная теорема из этого набора описывает некоторое элементарное действие (перемещение робота из одной точки в другую, применение некоторой формулы интегрирования и т.п.). Примечательно, что мы можем не знать имена теорем, перебираемых в ходе планирования решения задачи и/или попавших в окончательный вариант плана решения. Автоматически выбираются такие теоремы, которые приводят к достижению цели, описываемой в образце этой теоремы. Вызов теорем происходит в сочетании с режимом возвратов; распространение неуспеха может влиять на процесс планирования решения.
Все теоремы – любого из трех указанных типов – пользователь должен определять сам (встроенных теорем в языке нет).

Пример определения целевой теоремы:

```
[define TRAN-R (conseq (x y)
```

```
        (at  R  *y)
    [search1  (AT  R  *x)]
    [erase     (AT  R   .x)]
    [assert    (AT  R   .y)] )]
```

Эта теорема (с именем TRAN-R) описывает перемещение робота (R) из точки x в точку y. Теорема может быть вызвана по образцу – (at R *y) – в ситуации, когда ставится цель «робот R должен попасть в некоторую точку проблемной среды», скажем в точку G (такая целевая ситуация описывается выражением (at R G), которое соответствует образцу теоремы). Добиться этого можно, применив данную теорему (выполнив соответствующее ей действие в предметном мире) или, возможно, какие-то другие теоремы из числа описанных в программе.

Тело этой теоремы предписывает:

- найти точку, в которой находится R,
- вычеркнуть из базы данных утверждение о том, что R находится в этой точке,
- записать в базу данных утверждение о новом местонахождении R.

Вызов целевых теорем осуществляется с помощью функции achive (или goal):

[achive pat rec?] - pat – образец, rec? – факультативная рекомендация;

[goal pat test? rec?] - pat – образец, test? – факультативный набор требований к списку свойств утверждения, rec? – факультативная рекомендация;

Функция goal перед тем, как начать вызов теорем, проверяет, не представлена ли целевая ситуация в базе данных в виде утверждения (это означает, что цель на самом деле достигнута, никакого вызова теорем, никакого планирования решения не нужно).

Факультативный параметр rec (рекомендации) позволяет влиять на процесс перебора теорем, отдавать приоритет некоторым теоремам, учитывать их «стоимость» и т.п. Более того, можно "редактировать" (динамически менять) рекомендации с учетом попыток вызова других теорем.

Примеры рекомендаций:

(use T1 [ ]) - вызвать теорему с именем T1, а если вызов неуспешен, вызывать все остальные,

(try [NOT T3]) - вызывать все теоремы кроме T3,

(use1) - если найдена одна успешная теорема, отменить все развилки.

Пример определения теоремы типа «при вычеркивании»:

```
[define   КУБИК_ОСВ (erasing  (X)
                    (=ЗАНЯТ=  *X)
          [assert  (=СВОБОДЕН=  .X)])]
```

Если из базы данных вычеркивается утверждение о том, что "поверхность некоторого кубика X занята" (если так, то на него нельзя поставить другой кубик), то автоматически будет вызвана и выполнена эта теорема. Она запишет в базу данных утверждение: "поверхность этого кубика X освободилась" (теперь на него можно поставить другой кубик).

Вызов теорем типа «при вычеркивании» осуществляется с помощью функции change – либо явно, либо неявно (из функции erase):

[change pat rec?]

Пример определения теоремы типа «при записи»:

```
[define  ПРИШЕЛ  (antec  (X  Y  Z)
                  (*X  =ПРИШЕЛ=  *Y)
          [search1 (.X  =НАХОДИТСЯ_В=  *Z)]
          [erase   (.X  =НАХОДИТСЯ_В=  .Z)]
          [assert  (.X  =НАХОДИТСЯ_В=  .Y)])]
```

Если в базу данных записывается утверждение о том, что "X пришел в Y (из Z)", то автоматически будет вызвана и выполнена эта теорема. Она найдет в базе данных утверждение о прежнем местонахождении X, вычеркнет это утверждение и запишет, что "X находится вY".

Вызов теорем типа «при записи» осуществляется с помощью функции draw – опять же либо явно, либо неявно (из функции assert):

[draw pat rec?]

Решение задач и искусственный интеллект

Двумя составными элементами процесса решения задач в теории искусственного интеллекта являются *представление* (формализация) задач и собственно решение – *поиск*. Мы рассмотрим два подхода к решению задач и, соответственно, два способа представления – подход с использованием пространства состояний и подход, основанный на редукции задач. Для обоих подходов описываются используемые алгоритмы поиска решения. Важной особенностью большинства этих алгоритмов является использование

эвристической информации. *Эвристикой* обычно принято называть любое правило, стратегию, прием, существенно помогающий решению некоторой задачи. В области искусственного интеллекта и теории поиска под эвристической информацией понимается все то, что относится к конкретной решаемой задаче и служит более быстрому ее решению.

Представление задач в пространстве состояний

Основные понятия

Типичным представителем класса задач, для которых подходит представление в пространстве состояний, является головоломка, известная как игра в пятнадцать – см. рис.1(а). В ней используется пятнадцать пронумерованных (от 1 до 15) подвижных фишек, расположенных в клетках квадрата 4×4. Одна клетка этого квадрата всегда остается пустой, так что одну из соседних с ней фишек можно передвинуть на место этой пустой клетки, изменив тем самым местоположение пустой клетки. Заметим, что более простым вариантом этой головоломки является игра в восемь (квадрат 3×3 и восемь фишек) – пример соответствующей задачи показан на рис.1(б).

На рис.1(а) изображены две конфигурации фишек. В головоломке требуется преобразовать первую, или начальную, конфигурацию во вторую, или целевую конфигурацию. Решением этой задачи будет подходящая последовательность сдвигов фишек, например: передвинуть фишку 8 вверх, фишку 6 влево и т.д.

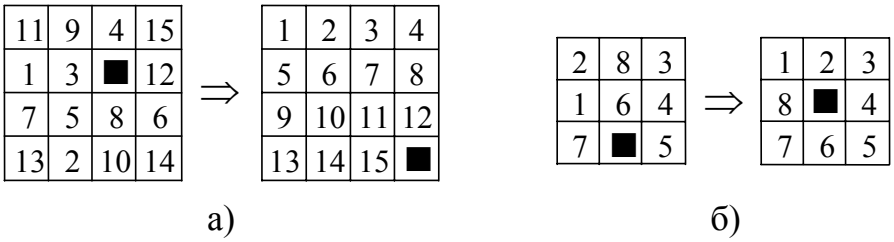


Рис. 1. Игра в пятнадцать и игра в восемь фишек

Важной особенностью класса задач, к которому принадлежит игра в пятнадцать (восемь) фишек, является наличие в задаче точно определенной начальной ситуации и точно определенной цели. Имеется также некоторое множество операций, или ходов, переводящих одну конфигурацию в другую. Именно из таких ходов состоит искомое решение задачи, которое можно в принципе получить методом проб и ошибок. Действительно, отправляясь от начальной ситуации, можно построить конфигурации, возникающие в результате выполнения возможных в этой ситуации ходов, затем построить множество конфигураций, получающихся после применения следующего хода, и так далее – пока не будет достигнута целевая конфигурация.

Введем теперь основные понятия, используемые при формализации задачи в пространстве состояний. Центральным из них является понятие ***состояния*** задачи. Например, для игры в пятнадцать (или в восемь) состояние — это просто некоторая конкретная конфигурация фишек.

Среди всех состояний задачи выделяются ***начальное состояние*** и ***целевое состояние***, в совокупности определяющие задачу, которую надо решить – примеры их приведены на рис.1.

Другим важным понятием является понятие ***оператора***, т.е. допустимого хода в задаче. Оператор преобразует одно состояние в другое, являясь по сути функцией, определенной на множестве состояний и принимающей значения из этого множества. Для игры в пятнадцать или в восемь удобнее выделить четыре оператора, соответствующие перемещениям пустой клетки (фишки-«пустышки») влево, вправо, вверх, вниз. В некоторых случаях оператор может оказаться неприменимым к какому-то состоянию: например, операторы сдвига вправо и вниз неприменимы, если пустая клетка расположена в правом нижнем углу. Значит, в общем случае оператор является частично определенной функцией отображения состояний.

В терминах состояний и операторов ***решение задачи*** есть определенная последовательность операторов, преобразующая начальное состояние в целевое. Решение задачи ищется в ***пространстве состояний*** – множестве всех состояний, достижимых из начального состояния при помощи заданных операторов. Например, в игре в пятнадцать или в восемь пространство состояний состоит из всех конфигураций фишек, которые могут быть образованы в результате возможных перемещений фишек.

Пространство состояний можно представить в виде направленного графа, вершины которого соответствуют состояниям, а дуги (ребра) – применяемым операторам. Указанные в виде стрелок направления соответствуют движению от вершины-аргумента применяемого оператора к результирующей вершине. Тогда решением задачи будет путь в этом графе, ведущий от начального состояния к целевому. На рис.2 показана часть пространства состояний для игры в пятнадцать: в каждой вершине помещена та конфигурация фишек, которую она представляет. Все дуги между вершинами являются двунаправленными, поскольку в этой головоломке для

любого оператора есть обратный ему (точнее, множество операторов состоит из двух пар взаимно-обратных операторов: влево-вправо, вверх-вниз).

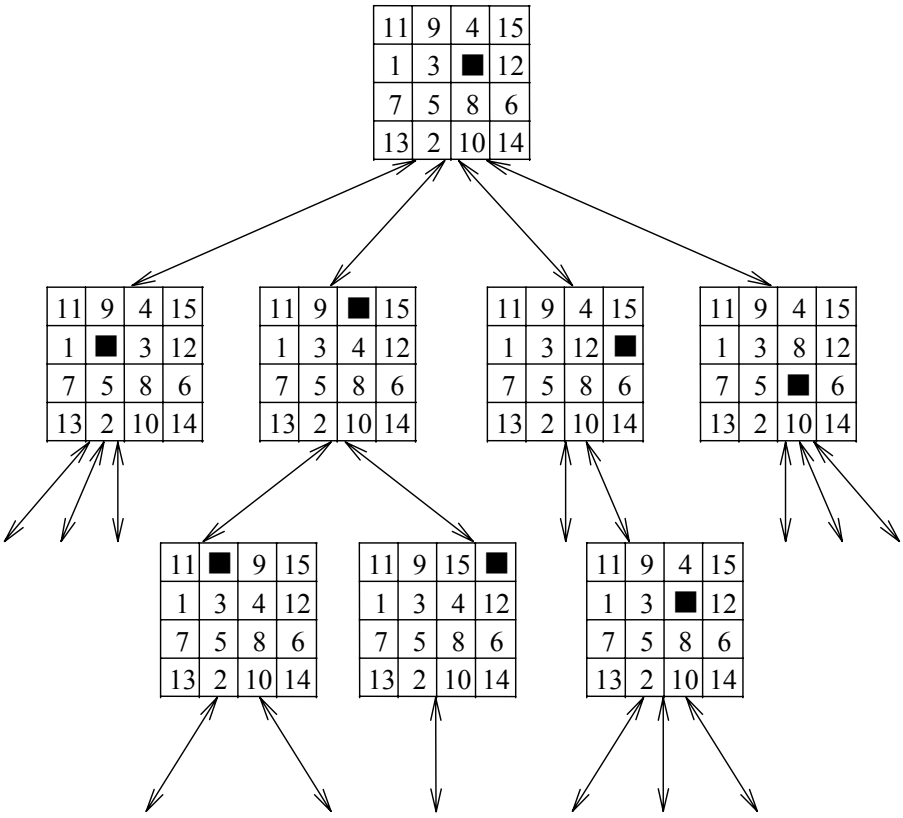


Рис. 2. Часть пространства состояний для игры в пятнадцать

Пространства состояний могут быть большими и даже бесконечными, но в любом случае предполагается счетность множества состояний.

Таким образом, в подходе к решению задачи с использованием пространства состояний задача рассматривается как тройка (S_I, O, S_G), где

S_I – начальное состояние;

O – конечное множество операторов, действующих на не более чем счетном множестве состояний;

S_G – целевое состояние.

Дальнейшая формализация решения задачи с использованием пространства состояний предполагает выбор некоторой конкретной *формы* описания состояний задачи. Для этого могут применяться любые подходящие структуры – строки, массивы, списки, деревья и т.п. Например, для игры в пятнадцать или восемь наиболее естественной формой описания состояния будет двумерный массив. Заметим, что от выбора формы описания состояния зависит в общем случае сложность задания операторов задачи, которые должны быть также определены при формализации задачи в пространстве состояний.

Если для игры в пятнадцать средством формализации выступает язык программирования Лисп или Паскаль, то операторы задачи могут быть описаны в виде четырех соответствующих функций языка. При использовании же продукционного языка, эти операторы задаются в виде правил продукций вида: «исходное состояние → результирующее состояние» (см. [Нильсон73, раздел 2.2]).

В рассмотренных на рис.1 примерах задач искомое целевое состояние задается явно, т.е. известно местоположение каждой фишки в целевой конфигурации. В более сложных случаях игры может быть несколько целевых состояний, либо же целевое состояние может быть определено неявно, т.е. охарактеризовано некоторым свойством, например, как состояние, в котором сумма номеров фишек в верхнем ряду не превосходит 10. В подобных случаях свойство, которому должно удовлетворять целевое состояние, должно быть описано исчерпывающим образом, к примеру, путем задания булевской функции, реализующей проверку нужного свойства состояния задачи.

Итак, для представления задачи в пространстве состояний необходимо определить следующее:

- форму описания состояний задачи и описание начального состояния;
- множество операторов и их воздействий на описания состояний;
- множество целевых состояний или же описание их свойств.

Перечисленные составляющие задают неявно граф-пространство состояний, в котором необходимо найти решение задачи. Заметим попутно, что, в отличие от такого неявного способа задания графа, при явном способе задания все вершины и дуги графа должны быть перечислены, например, с помощью таблиц.

Решение задачи в пространстве состояний подразумевает просмотр неявно заданного графа, для чего необходимо преобразование в явную форму достаточно большой его части, включающей искомую целевую вершину. Действительно, просмотр осуществляется как последовательный ***поиск***, или ***перебор*** вершин, в ***пространстве состояний***. В исходной точке процесса к начальному состоянию применяется тот или иной оператор и строится новая вершина-состояние, а также дуги, связывающие ее с корневой вершиной. На каждом последующем шаге поиска к одной из уже полученных вершин-состояний применяется допустимый оператор и строится еще одна вершина графа и связывающие дуги. Если очередная построенная вершина соответствует целевому состоянию, то процесс поиска завершается.

Примеры пространств состояний

Разберем два характерных примера представления задач в пространстве состояний, показывающих, что такое представление возможно для различных типов задач. Подчеркнем заранее, что предлагаемые ниже представления, хотя и являются достаточно естественными, все же не являются единственно допустимыми в этих задачах, возможны и другие варианты.

Вообще, от выбора представления, т.е. рассмотренных выше составляющих, зависит размер пространства состояний, а значит, и эффективность поиска в нем. Очевидно, желательны представления с малыми пространствами состояний, но нахождение удачных представлений, сужающих пространство поиска, требует обычно некоторого дополнительного анализа решаемой задачи.

Рассмотрим формализацию в пространстве состояний известной задачи о коммивояжере. Коммивояжер, располагая картой дорог, соединяющей 7 городов (см. рис.3), должен построить свой маршрут так, чтобы, выехав из города A, он посетил каждый из других шести городов B, C, D, E, H, G в точности по одному разу и затем вернулся в исходный город. В другом, более сложном варианте задачи требуется также, чтобы маршрут имел минимальную протяженность.

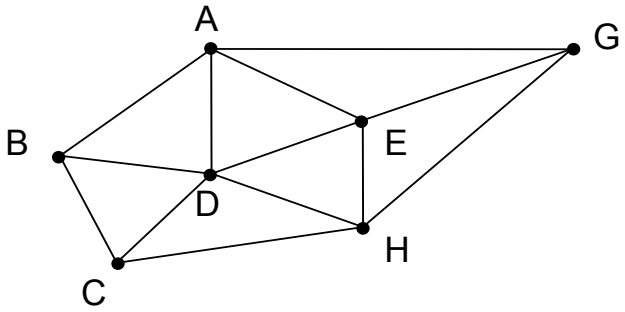


Рис. 3. Задача о коммивояжере: схема дорог между городами

Состояние решаемой задачи можно задать как список городов, которые коммивояжер уже посетил к текущему моменту. Тогда возможным состояниям соответствуют списки из элементов A, B, C, D, E, H, G без повторений, исключение составляет только элемент-город A, он может встретиться в списке дважды – в начале списка и его конце. Пример списка-состояния – (A B C H). Начальное же состояние определяется как список (A), а целевое – как любой допустимый список, начинающийся и кончающийся элементом A.

Для определенных таким образом состояний операторы задачи могут соответствовать перемещениям между городами (т.е. дугам графа рис.3) – получаем таким образом 13 операторов. На рис.4 показана в виде графа часть получающегося пространства состояний, включающая решающий путь. Этот граф является деревом – действительно, при применении любого оператора список уже посещенных городов может только удлиниться, поэтому в процессе построения новых вершин не может возникнуть вершина, встречавшаяся прежде.

Операторы в этой задаче могут быть определены и другим образом. Например, введем для каждого из семи городов оператор переезда в этот город (из любого другого). Такой оператор применим к некоторому описанию состояния, если он соответствует карте дорог и в список-описание не включен город, в который надо произвести переезд. Например, оператор переезда в город B неприменим к состоянию (A B C D), но применим к состоянию (A D).

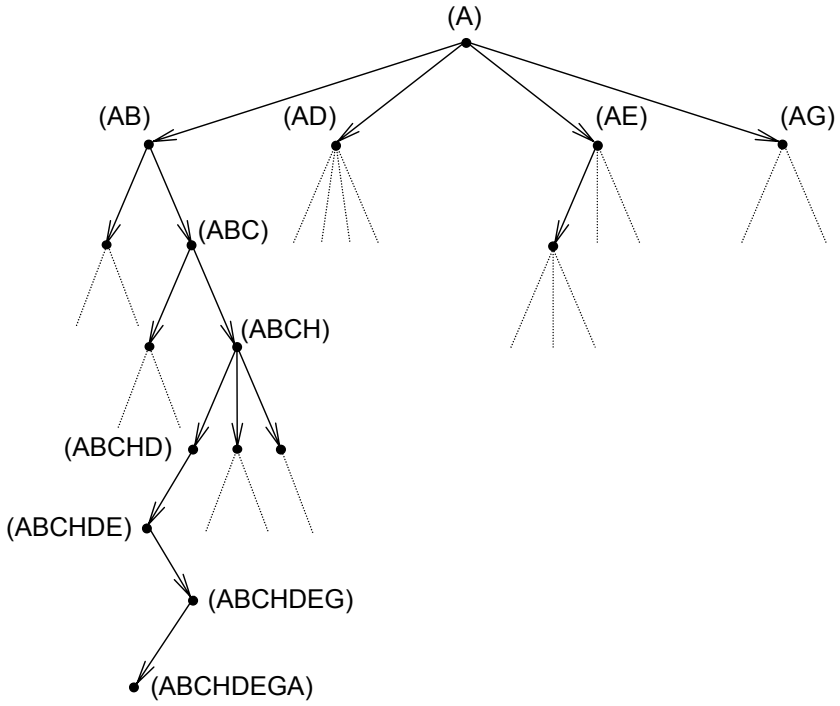


Рис. 4. Задача о коммивояжере: часть пространства состояний

Обратимся теперь к известной задаче об обезьяне и банане, простейшую формулировку которой мы и рассмотрим. В комнате находятся обезьяна, ящик и связка бананов, которая подвешена к потолку настолько высоко, что обезьяна может до нее дотянуться, только встав на ящик. Нужно найти последовательность действий, которые позволят обезьяне достать бананы. Предполагается, что обезьяна может ходить по комнате, двигать по полу ящик, взбираться на него и хватать бананы.

Ясно, что описание состояния этой задачи должно включать следующие сведения: местоположение обезьяны в комнате – в горизонтальной плоскости пола и по вертикали (т.е. на полу она или на ящике), местоположение ящика на полу и наличие у обезьяны бананов. Все это можно представить в виде 4-элементного списка *(ПолОб, ВертОб, ПолЯщ, Цель),* где

ПолОб – положение обезьяны на полу (это может быть двухэлементный вектор координат);

ПолЯщ – положение ящика на полу;

ВертОб – это символ П или Я в зависимости от того, где находится обезьяна, на полу или на ящике;

Цель – это число 0 или 1 в зависимости от того, достала ли обезьяна бананы или нет.

Зафиксируем также три следующие значимые точки в плоскости пола:

$Т_о$ – точка первоначального местоположения обезьяны;

$Т_я$ – точка первоначального расположения ящика;

$Т_б$ – точка пола, расположенная непосредственно под связкой бананов.

Тогда начальное состояние задачи описывается списком ($Т_о$, П, $Т_я$, 0), а целевое состояние задается как любой список, последний элемент которого равен 1.

Операторы в этой задаче можно определить так:

1) *Перейти* (*W*) – обезьяна переходит в точку *W* плоскости пола;
2) *Передвинуть* (*V*) – обезьяна передвигает ящик в точку *V* пола;
3) *Взобраться* – обезьяна взбирается на ящик;
4) *Схватить* – обезьяна хватает связку бананов.

Условия применимости и действие этих четырех операторов легко задать в виде правил продукций вида:

аргумент оператора $\rightarrow$ *результат оператора*,

причем *X, Y, Z, W, V* – это переменные:

1. *Перейти* (*W*) : (*X*, П, *Y*, *Z*) $\rightarrow$ (*W*, П, *Y*, *Z*)
2. *Передвинуть* (*V*) : (*X*, П, *X*, *Z*) $\rightarrow$ (*V*, П, *V*, *Z*)
3. *Взобраться* : (*X*, П, *X*, *Z*) $\rightarrow$ (*X*, Я, *X*, *Z*)
4. *Схватить* : ($Т_б$, Я, $Т_б$, 0) $\rightarrow$ ($Т_б$, Я, $Т_б$, 1)

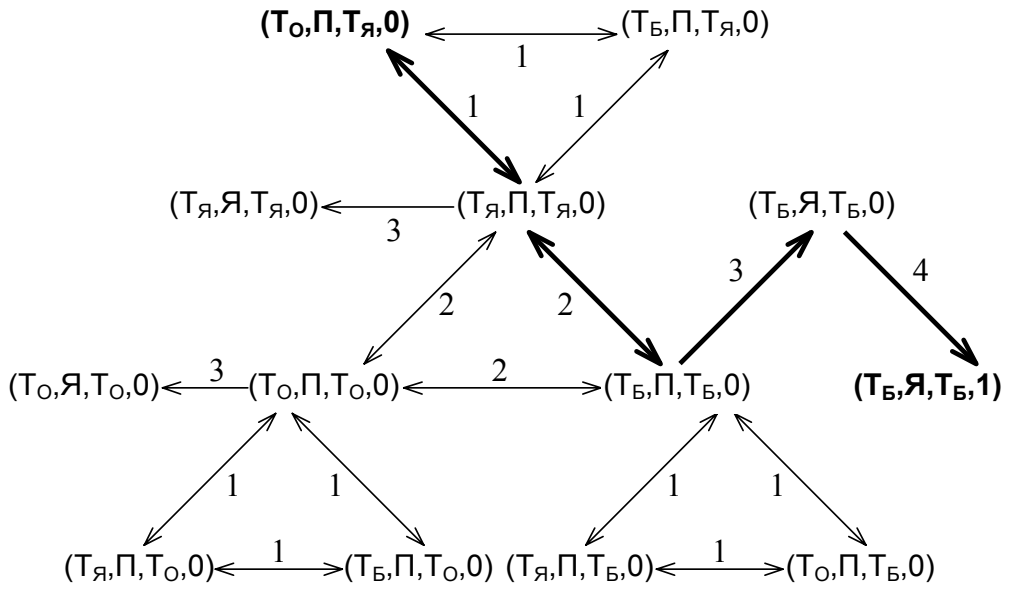


Рис. 5. Пространство состояний в задаче об обезьяне

Если считать, что для решения задачи значимы лишь вышеупомянутые точки пола $т_о$, $т_я$, $т_б$, тогда получим пространство состояний задачи, изображенное на рис.5. Это пространство содержит только 13 состояний, дуги графа-пространства помечены порядковым номером применяемого оператора. Пространство содержит четыре цикла хождения обезьяны между тремя значимыми точками (с ящиком или без него). В пространстве есть также две тупиковые ветви – когда обезьяна залезает на ящик, но не под связкой бананов. Жирными дугами (стрелками) показан решающий путь, состоящий из четырех операторов:

Перейти ($Т_я$); *Передвинуть* ($Т_б$); *Взобраться*; *Схватить*.

Рассмотренный пример показывает, сколь важен для успешного и эффективного решения задачи выбор определенного представления. Такое небольшое по размерам пространство состояний получено, в частности, вследствие того, что игнорировались все точки пола, кроме трех, соответствующих первоначальному расположению обезьяны, ящика и бананов.

Мощным приемом сужения пространств состояний является применение так называемых *схем состояний* и *схем операторов*, в которых для описаний состояний и операторов используются переменные. Тем самым схема состояния описывает целое множество состояний, а не только одно, а схема оператора определяет все множество действий некоторого типа. В рассмотренном нами представлении задачи об обезьяне использовались схемы операторов, но не схемы состояний. Другое представление этой задачи с использованием как схем состояний, так и схем операторов приведено в [Нильсон73, раздел 2.7].

Разновидности поиска

Как уже отмечалось, поиск в пространстве состояний базируется на последовательном построении (переборе) вершин графа состояний – до тех пор, пока не будет обнаружено целевое состояние. Введем несколько терминов, которые будем использовать для описания различных алгоритмов поиска.

Вершину графа, соответствующую начальному состоянию, естественно назвать ***начальной*** вершиной, а вершину, соответствующую целевому состоянию – ***целевой***. Как и ранее, вершины, непосредственно следующие за некоторой вершиной, т.е. получившиеся в результате применения к ней допустимых операторов, будем называть *дочерними*, а саму исходную вершину – *родительской*. Основной операцией, выполняемой при поиске на графе, будем считать ***раскрытие вершины***, что означает порождение (построение) всех ее дочерних вершин, путем применения к соответствующему описанию состояния задачи всех допустимых операторов.

Поиск в пространстве состояний можно представить как процесс постепенного раскрытия вершин и проверки свойств порождаемых вершин. Важно, что в ходе этого процесса должны сохраняться *указатели* от всех возникающих дочерних вершин к их родительским. Именно эти указатели позволят восстановить путь назад к начальной вершине после того, как будет построена целевая вершина. Этот путь, взятый в обратном направлении, точнее, последовательность операторов, соответствующих дугам этого пути, и будет искомым решением задачи.

Вершины и указатели, построенные в процессе поиска, образуют поддерево всего неявно определенного при формализации задачи графа-пространства состояний. Это поддерево называется ***деревом поиска***.

Процедуры поиска в пространстве состояний различаются несколькими характеристиками, основными из которых являются:

- использование эвристической информации;
- порядок раскрытия (перебора) вершин;
- полнота просмотра пространства состояний;
- направление поиска.

В соответствии с первой характеристикой различают ***слепой*** и ***эвристический*** поиск. В первом случае местонахождение целевой вершины в пространстве состояний никак не влияет на порядок, в котором раскрываются (перебираются) вершины. В противоположность слепому поиску, эвристический поиск использует априорную, эвристическую информацию об общем виде пространства состояний и/или о том, где в этом пространстве расположена цель, поэтому для раскрытия обычно выбирается наиболее перспективная вершина. В общем случае это позволяет сократить возникающий перебор вершин.

Два основных подвида слепого поиска, различающиеся порядком раскрытия вершин, – это ***поиск вширь*** и ***поиск вглубь***.

Как слепой, так и эвристический поиск могут характеризоваться полнотой просмотра пространства состояний. При ***полном*** переборе осуществляется исчерпывающий просмотр графа-пространства состояний, тем самым гарантируется нахождение решения, если таковое существует. В случае ***неполного*** поиска просматривается лишь некоторая, хотя и существенная часть пространства, и если эта часть не содержит целевых вершин, то искомое решение задачи найдено не будет.

В соответствии с направлением различают ***прямой*** поиск, ведущийся от начальной вершины к целевой, ***обратный*** поиск, ведущийся в направлении от целевой вершины к начальной, и ***двунаправленный*** поиск, при котором чередуются прямой и обратный поиск. Обратный поиск возможен в случае обратимости операторов задачи. Чаще используются (отчасти, в силу их простоты) алгоритмы прямого поиска, они и рассматриваются в данном пособии.

Слепой поиск

Слепые алгоритмы поиска вширь (*breadth_first_search*) и поиска вглубь (*depth_first_search*) различаются тем, какая вершина выбирается для очередного раскрытия. В алгоритме перебора вширь вершины раскрываются в том порядке, в котором они строятся. В алгоритме же перебора вглубь прежде всего раскрываются те вершины, которые были построены последними.

Сначала рассмотрим эти алгоритмы для пространств, являющихся деревьями (корнем дерева является начальная вершина). Затем покажем, как алгоритмы следует модифицировать для поиска на произвольных графах. Организовать перебор в деревьях проще, так как при построении нового состояния (и соответствующей вершины) можно быть уверенным в том, что такое состояние никогда раньше не строилось и не будет строиться в дальнейшем.

Описываемые алгоритмы используют списки **`Open`** и **`Closed`**, состоящие соответственно из раскрытых и нераскрытых вершин пространства состояний. Имя **`Current`** обозначает вершину, раскрываемую в текущий момент поиска. При формулировке алгоритмов предполагаем, что начальная вершина не является целевой.

Поиск вширь

Базовый **алгоритм поиска вширь** состоит из следующей последовательности шагов:

Шаг 1. Поместить начальную вершину в список нераскрытых вершин **`Open`**.

Шаг 2. Если список **`Open`** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к следующему шагу.

Шаг 3. Выбрать первую вершину из списка **`Open`** (назовем ее **`Current`**) и перенести ее в список раскрытых вершин **`Closed`**.

Шаг 4. Раскрыть вершину **`Current`**, образовав все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в любом порядке) в конец списка **`Open`** и построить указатели, ведущие от этих вершин к родительской вершине **`Current`**.

Шаг 5. Проверить, нет ли среди дочерних вершин целевых. Если есть хотя бы одна целевая вершина, то завершить поиск и выдать решение задачи, получающееся просмотром указателей назад от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

Конец алгоритма.

Основу этого алгоритма составляет цикл последовательного раскрытия (шаги 2-5) концевых вершин (листьев) дерева перебора, хранящихся в списке **`Open`**. Описанный алгоритм поиска вширь является полным. Можно также показать, что при переборе вширь непременно будет найден самый короткий путь к целевой вершине – при условии, что этот путь вообще существует. Если же решающего пути нет, то в случае конечного пространства состояний алгоритм завершит работу (с сообщением о неудаче поиска), в случае же бесконечного пространства алгоритм не остановится.

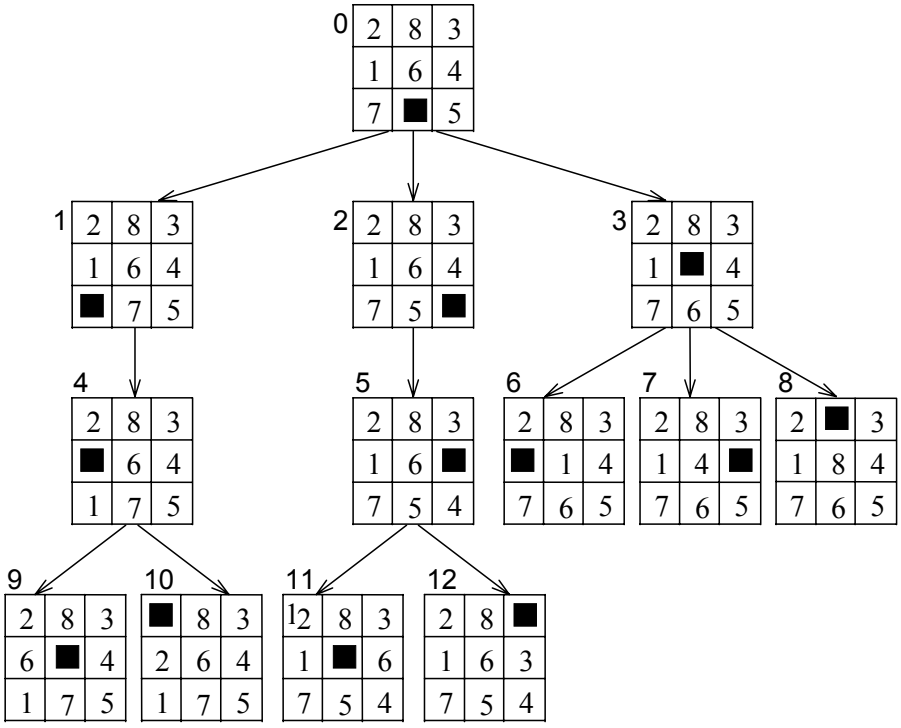


Рис.13. Поиск вширь для игры в восемь

На рис.13 показана часть дерева, построенного в результате применения алгоритма поиска вширь к некоторой начальной конфигурации игры в восемь, причем выполнение алгоритма прервано после построения первых 12 вершин (при этом раскрыто 6 вершин). В вершинах дерева помещены соответствующие описания состояний. Эти вершины перенумерованы в том порядке, в котором они были построены в ходе поиска. На следующем шаге цикла алгоритма будет раскрываться одна из вершин с номерами 6, 7 или 8, поскольку они расположены в начале списка нераскрытых вершин.

Считаем, что порядок построения дочерних вершин соответствует следующему зафиксированному порядку перемещения пустой клетки («пустышки»): влево/вправо/вверх/вниз. Предполагается также, что используемая алгоритмом операция раскрытия вершин организована таким образом, что она не порождает никакое состояние, идентичное состоянию в уже построенной вершине, являющейся родительской для раскрываемой вершины. Тем самым в дереве перебора нет дублирования одного и того же состояния в вершинах, имеющих общую соседнюю вершину.

В приведенном примере алгоритм поиска вглубь, сформулированный для пространств, являющихся деревьями, применялся к пространству состояний, являющемуся графом (в котором могут быть циклы). В некоторых случаях это допустимо, т.е. алгоритм находит решение, если оно есть, и заканчивает работу. Причем и при поиске в пространствах-деревьях, и при поиске в пространствах-графах, построенная алгоритмом структура из вершин и указателей всегда образует дерево (дерево перебора), поскольку указатели от дочерних вершин ссылаются только на одну порождающую вершину. Но в случае поиска на произвольном графе (и в этом – отличие от деревьев-пространств) одно и тоже состояние может быть продублировано в разных частях построенного дерева поиска: граф как бы разворачивается в дерево путем дублирования некоторых его частей. Например, по принятому предположению об операции раскрытия в игре в восемь исключалось только повторное возникновение состояний, уже встречавшихся два шага вверх по дереву перебора, другие же, более далекие друг от друга повторы одного и того же состояния остаются возможными. В общем случае, вследствие многократного дублирования вершин (из-за циклов в графе) возможно зацикливание базового алгоритма поиска вширь.

Поиск вглубь

Для формулировки алгоритма поиска вглубь необходимо определить понятие ***глубины вершины*** в дереве поиска. Это можно сделать следующим образом:

- глубина *корня* дерева равна нулю;
- глубина каждой *некорневой* вершины на единицу больше глубины ее родительской вершины.

В алгоритме перебора вглубь раскрытию в первую очередь подлежит вершина, имеющая наибольшую глубину. Такой принцип может привести к не завершающемуся процессу – это происходит, если пространство состояний бесконечно, и поиск вглубь пошел по бесконечной ветви дерева, не содержащей целевой вершины. Поэтому необходимо то или иное ограничение поиска, самый распространенный способ – ограничить глубину просмотра дерева. Это означает, что в ходе перебора можно строить только такие вершины, глубина которых не превышает некоторую заданную ***граничную глубину***. Тем самым, раскрытию в первую очередь подлежит вершина наибольшей глубины, но только если она расположена выше фиксированной границы. Соответствующий алгоритм поиска называется ***ограниченным поиском вглубь***.

Основные шаги базового **алгоритма ограниченного поиска вглубь** (с граничной глубиной **D**) таковы:

Шаг 1. Поместить начальную вершину в список нераскрытых вершин **Open**.

Шаг 2. Если список **Open** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к следующему шагу.

Шаг 3. Выбрать первую вершину из списка **Open** (назовем ее **Current**) и перенести ее в список раскрытых вершин **Closed**.

Шаг 4. Если глубина вершины **Current** равна граничной глубине **D**, то перейти к шагу 2, в ином случае перейти к следующему шагу.

Шаг 5. Раскрыть вершину **Current**, построив все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в произвольном порядке) в начало списка **Open** и построить указатели, ведущие от этих вершин к родительской вершине **Current**.

Шаг 6. Если среди дочерних есть хотя бы одна целевая вершина, то завершить поиск и выдать решение задачи, получающееся просмотром назад указателей от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

Конец алгоритма.

Приведенное описание очень похоже на описание алгоритма поиска вширь, разница заключается только в ограничении глубины (шаг 4) и в участке списка **Open**, куда помещаются построенные дочерние вершины (шаг 5).

Поскольку глубина поиска ограничена, то будучи примененным к деревьям-пространствам состояний, описанный базовый алгоритм поиска вглубь всегда заканчивает работу. В отличие от алгоритма поиска вширь, он осуществляет неполный поиск, поскольку целевая вершина может располагаться ниже граничной глубины (в таком случае она не будет обнаружена).

На рис.14 показана часть дерева перебора, построенного алгоритмом поиска вглубь при граничной глубине равной 4. В качестве начального состояния взята та же самая конфигурация игры в восемь, что и в примере на рис.13. В ходе поиска также построено 12 вершин (раскрыто 7), вершины перенумерованы в том порядке, в котором они были построены. Как нетрудно убедиться, сравнивая два указанных рисунка, алгоритмами поиска вширь и вглубь построены разные деревья поиска. Видно, что в алгоритме поиска в глубину сначала идет поиск вдоль одного пути, пока не будет достигнута установленная граничная глубина, затем рассматриваются альтернативные пути той же или меньшей глубины, которые отличаются от первого пути лишь последней (концевой) вершиной, после чего рассматриваются пути, отличающиеся последними двумя вершинами, и т.д.

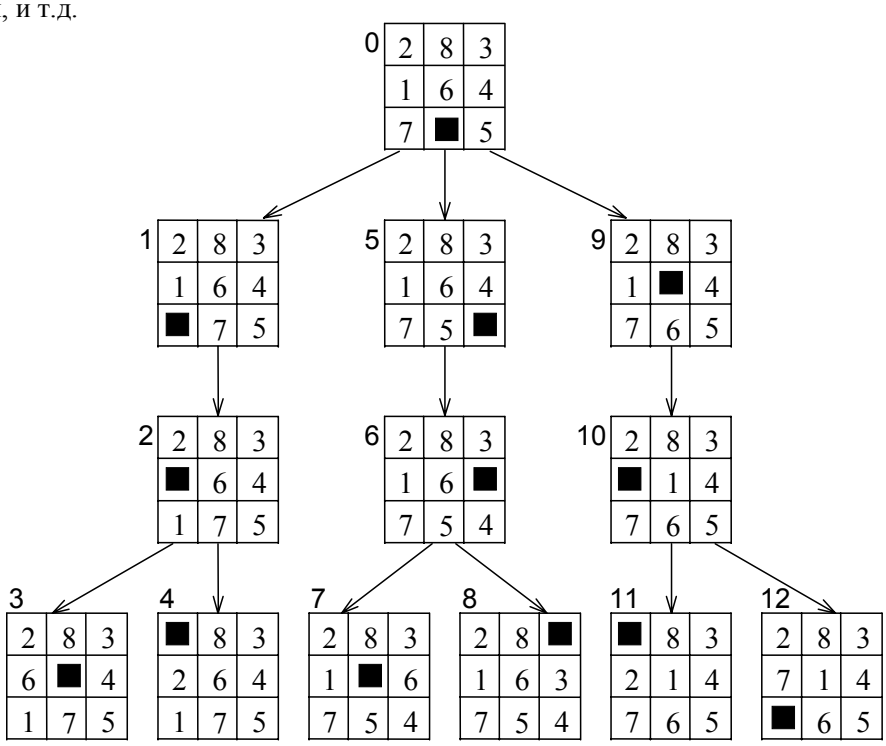


Рис.14. Поиск вглубь для игры в восемь

Слепой перебор и бектрекинг

Если продолжить выполнение алгоритмов перебора вширь и вглубь для рассмотренного на рис.13 и 14 начального состояния игры в восемь, с целью поиска конфигурации, заданной на рис.1(б), то она будет найдена на глубине 5, при этом алгоритмом поиска вширь будет раскрыто 26 и построено 46 вершин, а

алгоритмом поиска вглубь – соответственно 18 и 35 вершин (см. [Нильсон73, раздел 3.2] или [Нильсон85, раздел 2.4]).

В целом, алгоритмы поиска вширь и вглубь сравнимы по эффективности (в частности, по количеству построенных вершин). В то же время алгоритм поиска вглубь может оказаться предпочтительнее – в тех случаях, когда он начат с ветви дерева, содержащей целевое состояние, решение задачи будет обнаружено раньше, чем при поиске вширь.

Подчеркнем, что, также как и при переборе вширь, при переборе вглубь формируется именно дерево, а не граф поиска, даже если пространство состояний представлялось графом с циклами. В последнем случае, однако, дерево перебора может содержать дубликаты состояний и возможно зацикливание алгоритма. Если, к примеру, в графе две вершины являются друг для друга дочерними, то они будут многократно дублироваться в списке **`Open`**, что приводит к зацикливанию.

Чтобы избежать такого дублирования вершин в случае перебора на графах общего вида, необходимо внести некоторые очевидные изменения в описанные базовые алгоритмы поиска вширь и вглубь.

В алгоритме перебора вширь следует дополнительно проверять, не находится ли каждая вновь построенная вершина (точнее, соответствующее описание состояния) в списках **`Open`** и **`Closed`** по той причине, что она уже строилась раньше в результате раскрытия какой-то другой вершины. Если это так, то такую вершину не надо снова помещать в список **`Open`** (таким образом, разрывается цикл графа-пространства и обрывается соответствующая ветвь дерева перебора). В алгоритме же ограниченного поиска вглубь, кроме указанной проверки, может оказаться необходимым пересчет глубины порожденной дочерней вершины, уже имеющейся либо в списке **`Open`**, либо в списке **`Closed`**.

Усовершенствованный таким образом алгоритм поиска вширь всегда завершит работу в случае существования решения, а усовершенствованный алгоритм поиска вглубь закончится в любом случае, независимо от существования решения. В отсутствующем разделе 2.4 настоящей главы приводится реализация этих алгоритмов на языке программирования Лисп [Семенов]. Немаловажно, что алгоритмы слепого перебора описаны нами в форме, пригодной для их программирования с использованием любого языка, не только языка программирования задач искусственного интеллекта.

Алгоритм поиска вглубь демонстрирует также способ решения поисковых задач, называемый ***бектрекингом*** (*backtracking*). Этот способ предлагает определенную организацию перебора всех возможных вариантов решения задачи, число которых может быть велико.

Суть бектрекинга состоит в том, чтобы в каждой точке процесса решения задачи, где существует несколько априори равноправных альтернативных вариантов (путей) дальнейшего продолжения, выбрать один из них и следовать ему, предварительно запомнив другие альтернативы – для того, чтобы в случае неуспешности выбранного варианта-пути вернуться в точку выбора и выбрать для продолжения решения другой возможный вариант-путь. В общем случае в процессе решения возможно возникновение многих подобных точек выбора, называемых обычно *точками бектрекинга* или *развилками*; к каждой из таких точек может потребоваться *возврат* для выбора других вариантов решения.

В базовом алгоритме поиска вглубь по существу проводится бектрекинг. Действительно, запоминание всех альтернатив продолжения поиска (нераскрытых вершин) осуществляется в списке **`Open`**, на шаге 3 производится выбор варианта-альтернативы, а возврат к этому шагу для выбора следующей альтернативы осуществляется на шагах 4 и 5.

Некоторые языки для задач искусственного интеллекта, как, например, Пролог [Братко] и Плэнер [Пильщиков] имеют специальный встроенный механизм для реализации бектрекинга. Это означает, что запоминание точек бектрекинга – самих альтернатив и связанной с ними информации, а также реализация возвратов к нужным точкам (с восстановлением всей операционной обстановки в этой точке) возложены на интерпретатор языка, т.е. делается автоматически. От программиста требуется лишь определение точек бектрекинга с нужными альтернативами и инициация в необходимый момент процесса возврата. Заметим попутно, что язык Плэнер, по сравнению с Прологом, предлагает более гибкие средства управления бектрекингом, краткое описание этих средств находится в разделе 4.2 пособия.

Встроенный механизм бектрекинга позволяет упростить разработку поисковой программы, укорачивая ее текст – это демонстрирует приведенная в разделе 2.4 плэнерская функция `DEPTH_FIRST_SEARCH`, реализующая поиск вглубь (без ограничения глубины).

В целом алгоритмы слепого перебора являются неэффективными методами поиска решения, и в случае нетривиальных задач их невозможно использовать из-за большого числа порождаемых вершин. Действительно, если L – длина решающего пути, а B – количество ветвей (дочерних вершин) у каждой вершины, то в общем случае для нахождения решения надо исследовать B^L путей, ведущих из начальной вершины. Эта величина растет экспоненциально с ростом длины решающего пути, что приводит к ситуации, называемой комбинаторным взрывом.

Один из способов повышения эффективности поиска связан с использованием информации, отражающей специфику решаемой задачи и позволяющей более целенаправленно двигаться к цели. Такая информация обычно называется *эвристической*, а соответствующие алгоритмы и методы поиска – эвристическими.

Эвристический поиск

Идея, лежащая в основе эвристического поиска, состоит в том, чтобы с помощью эвристической информации оценивать перспективность нераскрытых вершин пространства состояний (с точки зрения достижения цели) и выбирать для продолжения поиска наиболее перспективную вершину.

Самый распространенный способ использования эвристической информации – введение так называемой ***эвристической оценочной функции***. Эта функция определяется на множестве вершин пространства состояний и принимает числовые значения. Значение эвристической оценочной функции Est(V) может интерпретироваться как перспективность раскрытия вершины (иногда – как вероятность ее расположения на решающем пути). Обычно считают, что меньшее значение Est(V) соответствует более перспективной вершине, и вершины раскрываются в порядке увеличения (точнее, неубывания) значения оценочной функции.

Алгоритм эвристического поиска

Последовательность шагов формулируемого ниже базового ***алгоритма эвристического (упорядоченного) перебора*** похожа на последовательность шагов алгоритмов слепого перебора, отличие заключается в использовании эвристической оценочной функции. После порождения нового состояния производится его оценивание (т.е. вычисление значения этой функции). Списки открытых и закрытых вершин содержат как вершины, так и их оценки, которые и используются для упорядочения поиска.

В цикле каждый раз для раскрытия выбирается наиболее перспективная концевая вершина дерева перебора. Как и в случае алгоритмов слепого поиска, множество порождаемых алгоритмом вершин и указателей образует дерево, в листьях которого находятся нераскрытые вершины.

Предполагаем, что исследуемое алгоритмом пространство состояний представляет собой дерево. Укажем основные шаги базового **алгоритма эвристического поиска** (*best_first_search*):

Шаг 1. Поместить начальную вершину в список нераскрытых вершин **`Open`** и вычислить ее оценку.

Шаг 2. Если список **`Open`** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к шагу 3.

Шаг 3. Выбрать из списка **`Open`** вершину с минимальной оценкой (среди вершин с одинаковой минимальной оценкой выбирается любая) и перенести эту вершину (назовем ее **`Current`**) в список **`Closed`**.

Шаг 4. Если **`Current`** – целевая вершина, то завершить поиск и выдать решение задачи, получающееся просмотром указателей от нее к начальной вершине, в противном случае перейти к шагу 5.

Шаг 5. Раскрыть вершину **`Current`**, построив все ее дочерние вершины. Если таких вершин нет, то перейти к шагу 2, в ином случае – к шагу 6.

Шаг 6. Для каждой дочерней вершины вычислить оценку (значение оценочной функции), поместить все дочерние вершины в список **`Open`** и построить указатели, ведущие от этих вершин к родительской вершине **`Current`**. Перейти к шагу 2.

Конец алгоритма.

Заметим, что поиск вглубь можно рассматривать как частный случай эвристического поиска с оценочной функцией Est(V) = d(V) , а поиск вширь – с оценочной функцией Est(V) = 1/d(V), где d(V) – глубина вершины V.

Чтобы модифицировать рассмотренный алгоритм для перебора на произвольных графах-пространствах состояний, необходимо предусмотреть в нем реакцию на случай построения дочерних вершин, которые уже имеются либо в списке раскрытых, либо в списке нераскрытых вершин.

В принципе, эвристическая оценочная функция может зависеть не только от внутренних свойств оцениваемого состояния (т.е. свойств входящих в описание состояния элементов), но и от характеристик всего пространства состояний, например, от глубины местонахождения оцениваемой вершины в дереве перебора или других свойств пути к этой вершине. Поэтому значение оценочной функции для вновь построенной дочерней вершины, входящей в список `Open` или `Closed`, может понизиться, и надо скорректировать старую оценку вершины, заменив ее на новую, меньшую. Если вновь построенная вершина с меньшей оценкой входит в список `Closed`, необходимо вновь поместить ее в список `Open`, но с меньшей оценкой. Потребуется также изменить направления указателей от всех вершин списков `Open` и `Closed`, оценка которых уменьшилась, направив их к вершине `Current`.

Впрочем, если оценочная функция учитывает только внутренние характеристики вершин-состояний, то для предотвращения зацикливания требуется более простая модификация алгоритма – надо просто исключить дублирование состояний в списках `Open` и `Closed`, оставляя в них лишь по одному состоянию (именно этот простой способ и реализован во всех лисп-алгоритмах поиска раздела 2.4).

Проиллюстрируем работу алгоритма эвристического поиска опять же на примере игры в восемь (для той же начальной ситуации, что на рис. 13 и 14, и целевого состояния, указанного на рис.1(б)). Воспользуемся в качестве оценочной следующей простой функцией:

$$Est1(V) = d(V) + k(V)$$

где: d(V) – глубина вершины V, или число ребер дерева на пути от этой вершины к начальной вершине;

k(V) – число фишек позиции-вершины V, стоящих не на «своем» месте (фишка стоит не на «своем» месте, если ее позиция отлична от позиции в целевом состоянии).

На рис. 15 показано дерево, построенное алгоритмом эвристического перебора с использованием указанной оценочной функции. Оценка каждой вершины приведена рядом с ней внутри кружка. Отдельно стоящие цифры, как и раньше, показывают порядок, в котором строились вершины. Двойной рамкой обведена найденная целевая вершина, она построена двенадцатой.

Видно, что поскольку каждый раз выбор вершины с минимальной оценкой производится внутри всего построенного к текущему моменту дерева перебора, то раскрываемые друг за другом вершины могут располагаться в отдаленных друг от друга частях дерева. Применяемая оценочная функция такова, что при прочих равных преимущество имеет менее глубокая вершина.

Решение задачи длиною в пять ходов найдено в результате раскрытия 6 и построения 13 вершин – это существенно меньше, чем при использовании слепого перебора (соответствующие числа были: для поиска вширь – 26 и 46 вершин, для поиска вглубь – 18 и 35 вершин). Таким образом, использование эвристической информации приводит к существенному сокращению перебора.

Эффективность алгоритмов поиска может быть оценена при помощи такого показателя, как *целенаправленность*. Он вычисляется по формуле

$$P = L / N,$$

где L – длина найденного пути до цели (она равна глубине целевой вершины);

N – общее число вершин, построенных в ходе перебора.

Легко заметить, что P=1, если строятся только вершины решающего пути, а в остальных случаях P<1. Вообще, эта величина тем меньше, чем больше строится бесполезных вершин. Таким образом, этот критерий показывает, насколько дерево, построенное при переборе, вытянуто, а не кустисто. Для рассмотренных примеров работы алгоритмов для игры в восемь этот показатель равен соответственно: эвристический поиск – 5/13, поиск вширь – 5/46, поиск вглубь – 5/35,

Ясно, что алгоритм эвристического поиска с хорошо подобранной оценочной функцией находит решение задачи быстрее алгоритмов слепого перебора. Однако подбор удачной эвристической функции, существенно сокращающей поиск, – наиболее трудный момент при формализации задачи, часто подходящая оценочная функция выявляется только экспериментально.

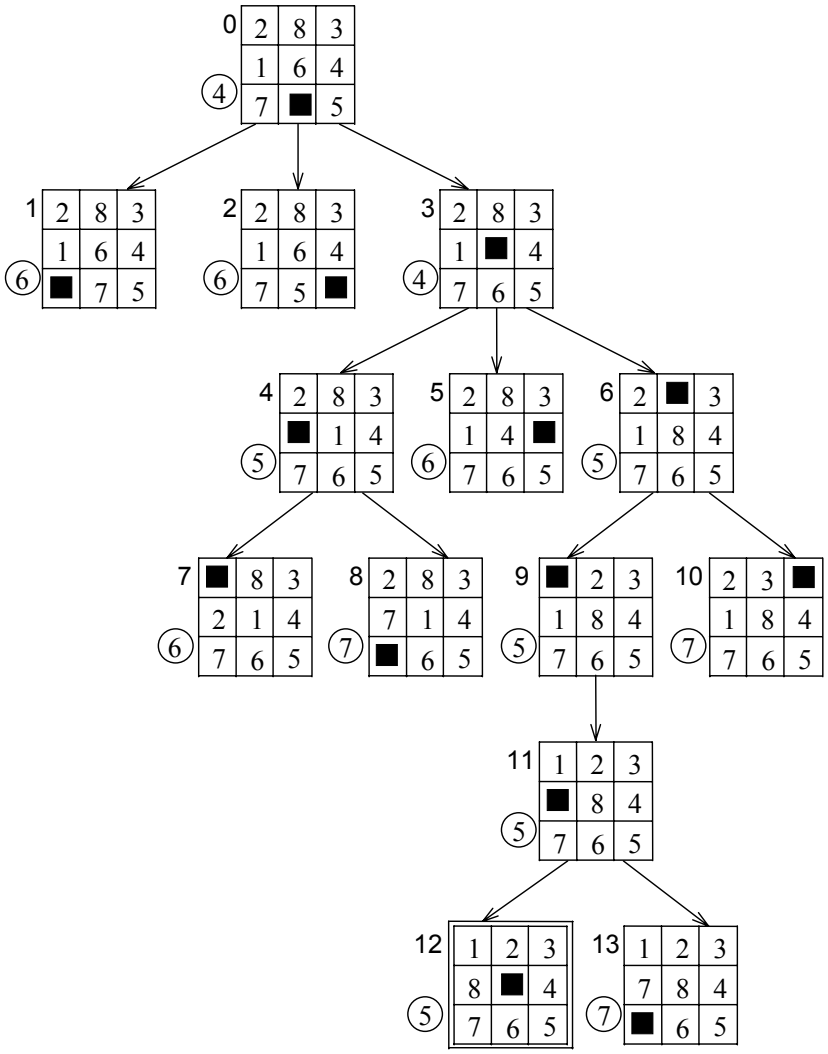


Рис. 15. Эвристический поиск для игры в восемь

В одной и той же задаче можно сравнивать различные оценочные функции по их *эвристической силе*, т.е. по тому, насколько они убыстряют поиск, делают его эффективным. Заметим, что эвристическая сила функции должна учитывать общий объем вычислительных затрат при поиске, поэтому кроме числа раскрытых и построенных вершин важен и такой показатель, как сложность вычисления самой оценочной функции.

Для игры в восемь можно предложить еще одну эвристическую функцию:

$$Est2(V) = d(V) + s(V) .$$

Первое слагаемое $d(V)$ этой функции имеет тот же смысл, что и для функции $Est1$. Второе слагаемое получается подсчетом для каждой из восьми фишек суммы двух расстояний – по вертикали и горизонтали – между клетками, где находится фишка в оцениваемом и целевом состояниях, а затем вычислением общей суммы $s(V)$ таких расстояний для всех восьми фишек. Тем самым, $s(V)$ выражает «суммарное расстояние» всех фишек от их целевого положения.

Например, для начальной конфигурации на рис.15 расстояние текущего положения фишки с номером 8 от ее положения в целевой конфигурации равно 1 и по вертикали, и по горизонтали, а сумма их равна 2. Общая же сумма таких расстояний для всех фишек равна 5 (фишки 3, 4, 5, 7 стоят уже на «своем» месте, поэтому их вклад в суммарное расстояние равен 0). Интуитивно ясно, и это можно показать на примерах, что новая эвристическая функция имеет большую эвристическую силу, т.е. более эффективно направляет поиск к цели.

Допустимость алгоритма эвристического поиска

Важным является вопрос, может ли алгоритм эвристического перебора с оценочной функций общего вида (на которую не накладывается никаких ограничений) гарантировать нахождение решающего пути за конечное число шагов в тех случаях, когда решение существует (как это гарантирует алгоритм поиска вширь). Понятно, что такой уверенности нет, прежде всего, для задач с бесконечными пространствами состояний. Вообще же, нередка ситуация, когда эвристика, сильно сокращающая перебор для одних задач (начальных и целевых состояний), в то же время для других задач либо не уменьшает перебор (решение задачи может искаться даже дольше, чем с использованием слепого поиска), либо вовсе не обеспечивает обнаружение решающего пути.

Математическое исследование алгоритма эвристического поиска, прежде всего – условий, гарантирующих нахождение им решения, было проведено для эвристических оценочных функций специального вида и для более сложной задачи, чем рассматриваемая до сих пор задача поиска произвольного решающего пути до целевой вершины.

Предположим, что на множестве дуг пространства состояний определена функция стоимости:

$c(V_A, V_B)$ – стоимость дуги-перехода от вершины V_A к вершине V_B .

Определим также ***стоимость*** любого пути в графе-пространстве как сумму стоимостей входящих в путь дуг. Пусть целью поиска будет не просто нахождение решающего пути, а нахождение ***оптимального решающего пути*** – решающего пути с минимальной стоимостью.

Предположим также, что эвристическая оценочная функция $Est(V)$ построена таким образом, чтобы оценивать стоимость оптимального решающего пути, идущего из начальной вершины к одной из целевой вершин, при условии, что этот путь проходит через вершину V. Тогда значение оценочной функции можно представить в виде суммы двух слагаемых:

$$Est(V) = g(V) + h(V) \qquad (*)$$

где $g(V)$ – оценка оптимального пути от начальной вершины до вершины V,

$h(V)$ – оценка оптимального пути от вершины V до целевой вершины.

Если в процессе поиска уже построена вершина V, то путь до нее найден, и его стоимость может быть вычислена. Найденный путь не обязательно оптимален (возможно, существует более дешевый, еще не найденный путь из начальной вершины в V), однако стоимость найденного пути может быть использована в качестве оценки искомого пути минимальной стоимости из начальной вершины до V, т.е. в качестве первого слагаемого $g(V)$ эвристической функции. Второе же слагаемое $h(V)$ может быть предложено исходя из эвристических соображений, свойственных конкретной решаемой задаче, как некоторая характеристика-оценка текущей вершины V (близости ее к цели). Таким образом, собственно эвристическая информация будет воплощена только во втором слагаемом оценочной функции.

Разновидность алгоритма эвристического поиска, применяемого для поиска оптимального решающего пути и использующего при этом оценочную функцию указанного выше вида (*), известна в литературе как ***А-алгоритм*** [Нильсон85, раздел 2.4]. Были доказаны важные свойства этого алгоритма, прежде всего, утверждение о его допустимости.

Алгоритм перебора называют ***допустимым*** (или *состоятельным*), если для произвольного графа он всегда заканчивает свою работу построением оптимального пути к цели, при условии, что такой путь существует.

Пусть $h^*(V)$ – стоимость оптимального пути из произвольной вершины V в целевую вершину.

Верна следующая **теорема о допустимости А-алгоритма**:

А-алгоритм, использующий некоторую эвристическую функцию вида (*), где

$g(V)$ – стоимость пути от начальной вершины до вершины V в дереве перебора, а

$h(V)$ – эвристическая оценка оптимального пути из вершины V в целевую вершину,

является допустимым, если $h(V) \le h^*(V)$ для всех вершин V пространства состояний.

А-алгоритм эвристического поиска с функцией $h(V)$, удовлетворяющей этому условию, получил название ***А*-алгоритма*** [Нильсон73, разделы 3.7-3.9; Лорьер, раздел 5.6].

Практическое значение этой теоремы в том, что для допустимости А-алгоритма достаточно найти какую-либо нижнюю грань функции $h^*(V)$ и использовать ее в качестве $h(V)$ – тогда оптимальность найденного алгоритмом решения будет гарантирована.

Если взять тривиальную нижнюю грань, т.е. установить $h(V)=0$ для всех вершин пространства состояний, то допустимость будет обеспечена. Однако этот случай соответствует полному отсутствию какой-нибудь эвристической информации о задаче, и оценочная функция Est не имеет никакой эвристической силы, т.е. не сокращает возникающий перебор. А*-алгоритм ведет себя при этом аналогично алгоритму поиска вширь.

Точнее, при $Est(V)=g(V)$, где $g(V)$ – стоимость пути от начальной вершины до вершины V, мы получаем алгоритм, известный как ***алгоритм равных цен***. Алгоритм равных цен представляет собой более общий вариант метода перебора вширь, при котором вершины раскрываются в порядке возрастания стоимости $g(V)$, т.е. в первую очередь раскрывается вершина из списка нераскрытых вершин, для которой величина g имеет наименьшее значение.

Если же, кроме того, положить стоимость $c(V_A,V_B)=0$ для всех дуг пространства состояний, то А*-алгоритм просто превращается в неэффективный слепой поиск вширь.

Обе предложенные для игры в восемь эвристические функции $Est1(V)$ и $Est2(V)$ удовлетворяют условию допустимости А*-алгоритма. Первое их слагаемое $d(V)$ есть стоимость пути к вершине V при стоимости всех дуг $c(V_A,V_B)=1$. Функции отличаются лишь вторым слагаемым, и можно показать, что значение второй функции всегда (т.е. для всех состояний), больше значения первой функции: $Est1(V) \le Est2(V)$, что равнозначно $k(V) \le s(V)$.

Действительно, во второй функции вклад каждой фишки в общую оценку-сумму $s(V)$ либо равен 0 (фишка стоит уже на «своем» месте), либо не меньше 1 (в противном случае), в первой же функции этот вклад в $k(V)$ соответственно либо равен 0, либо равен 1.

Из последнего неравенства следует, что условие допустимости достаточно доказать только для второй функции $Est2$. Справедливость нужного условия $s(V) \le h^*(V)$ следует из следующего соображения. Если бы фишки не мешали друг другу и могли двигаться до «своего» места по кратчайшему пути, как если бы других фишек на квадрате не было, то сумма длин таких путей для всех фишек была бы в точности равна значению $s(V)$. На самом же деле фишки часто не могут двигаться по кратчайшей траектории из-за того, что на ней расположены другие фишки, поэтому длина (стоимость) оптимального решения $h^*(V)$ будет не меньше $s(V)$.

Заметим, что функция $s(V)$ не учитывает должным образом трудность обмена местами двух соседних фишек, а поэтому ее эвристическая сила в принципе может быть повышена. В ряде случаев эвристическая сила некоторой оценочной функции может быть повышена просто путем умножения на положительную константу, большую единицы, однако иногда такое повышение приводит к потере допустимости алгоритма. Например, если для игры в восемь в качестве второй составляющей эвристической функции взять $h(V)=2 \cdot s(V)$, то в ряде случаев такая функция будет убыстрять поиск и позволит решать более трудные задачи, но условие допустимости перестанет выполняться (так как для начального состояния на рис.15: $h^*(V) \le 2 \cdot s(V)$).

Если неравенство $h_1(V) \le h_2(V)$ верно для всех вершин пространства состояний, не являющихся целевыми, А*-алгоритм, использующий эвристическую составляющую $h_2(V)$, называется *более информированным*, чем А*-алгоритм с функцией $h_1(V)$. Показано [Нильсон73, раздел 3.9], что если эти функции статичны (т.е. не изменяются в процессе поиска), то более информированный алгоритм раскрывает всегда меньшее число вершин, прежде чем находит путь минимальной стоимости. Это значит, что более информированный алгоритм осуществляет более целенаправленный, а значит, более эффективный поиск целевой вершины. Таким образом, понятие информированности отражает один из аспектов понятия эвристической силы оценочной функции при поиске в пространстве состояний.

Итак, желательно подбирать такую эвристическую функцию $h(V)$, которая была бы нижней границей $h^*(V)$ (чтобы гарантировать допустимость алгоритма) и которая была бы как можно ближе к $h^*(V)$ (чтобы обеспечить эффективность поиска). Заметим, что существуют задачи, для которых нельзя найти оценочную функцию, обеспечивающую во всех случаях как эффективность, так и допустимость эвристического поиска. Поэтому часто приходится использовать эвристические функции, сокращающие поиск во многих случаях, но не гарантирующие нахождение оптимального решающего пути.

В идеальном случае, когда известна сама оценка $h^*(V)$, и она используется в качестве $h(V)$, А*-алгоритм находит оптимальный решающий путь сразу, без раскрытия ненужных вершин.

Алгоритм «подъема на холм»

Сильным упрощением базового алгоритма эвристического поиска с произвольной оценочной функцией является алгоритм ***«подъема на холм»*** [Нильсон85, раздел 1.1.4]. Этот алгоритм при раскрытии каждой вершины производит упорядочение (по значению оценочной функции) порожденных дочерних вершин, и выбирает для последующего раскрытия дочернюю вершину с наименьшей оценкой, а не вершину с наименьшей оценкой среди всех нераскрытых вершин дерева поиска, как в базовом алгоритме эвристического поиска. Очевидно, что такой локальный выбор среди только что построенных дочерних вершин реализовать гораздо проще, чем глобальный выбор вершины во всем дереве перебора. Реализация этого алгоритма на языке Плэнер приводится в следующем разделе настоящей главы.

Идея этого алгоритма аналогична идее известного вне области искусственного интеллекта метода «подъема на гору», применяемого для поиска максимума (или минимума) функции. Согласно этому методу, для того чтобы, в конечном счете, найти максимум функции, на каждом шаге метода производится движение в направлении наибольшей крутизны функции. Для определенного класса функций (имеющих единственный максимум и некоторые другие свойства роста) такое использование локальной информации, т.е. знания направления наиболее крутого подъема в текущей точке, позволяет найти глобальное решение, т.е. максимум функции.

В алгоритме «подъема на холм», применяемом для поиска в пространстве состояний, роль функции метода «подъема на гору» играет эвристическая оценочная функция, взятая с обратным знаком. Поиск продолжается всегда от той дочерней вершины, которая имеет меньшее значение эвристической функции (при этом случай, когда вершин с одинаковой минимальной оценкой несколько, является нежелательным).

Важно, что алгоритм «подъема на холм» дает тот же результат, что и базовый алгоритм эвристического поиска в тех случаях, когда оценочная функция обладает определенными свойствами, в частности, имеет один (глобальный) экстремум. Алгоритм становится несостоятельным, если у эвристической функции имеется несколько локальных экстремумов. Бывают и другие случаи бесперспективности «подъема на холм»: если поверхность-множество значений функции имеет равнинный участок (как горное плато) или же участки узкого и длинного возвышения (в виде горного хребта), и процесс поиска вывел как раз на них. Таким образом, рассматриваемый алгоритм имеет ограниченную применимость, но иногда возникающие проблемы можно разрешить, построив более подходящую эвристическую функцию.

Поиск на игровых деревьях

Деревья игры. Поиск выигрышной стратегии

Будем рассматривать класс ***игр двух лиц с полной информацией***. В таких играх участвуют два игрока, которые поочередно делают свои ходы. В любой момент каждому игроку известно все, что произошло в игре к этому моменту и что может быть сделано в настоящий момент. Игра заканчивается либо выигрышем одного игрока (и проигрышем другого), либо ничьей.

Таким образом, в рассматриваемый класс не попадают игры, исход которых зависит хотя бы частично от случая – большинство карточных игр, игральные кости, «морской бой» и проч. Тем не менее класс достаточно широк: в него входят такие игры, как шахматы, шашки, реверси, калах, крестики-нолики и другие игры.

Для формализации и изучения игровых стратегий в классе игр с полной информацией может быть использован подход, основанный на редукции задач. Напомним, что при этом должны быть определены следующие составляющие: форма описания задач и подзадач; операторы, сводящие задачи к подзадачам; элементарные задачи; а также задано описание исходной задачи.

Наиболее интересной представляется задача поиска выигрышной стратегии для одного из игроков, отправляясь от некоторой конкретной позиции игры (не обязательно начальной). При использовании подхода, основанного на редукции задач, выигрышная стратегия ищется в процессе доказательства того, что игра может быть выиграна. Аналогично, поиск ничейной стратегии, исходя из некоторой конкретной позиции, ведется в процессе доказательства того, что игра может быть сведена к ничьей.

Ясно, что описание решаемой задачи должно содержать описание конфигурации игры, для которой ищется нужная стратегия. Например, в шашках игровая позиция включает задание положений на доске всех шашек, в том числе дамок. Обычно описание конфигурации содержит также указание, кому принадлежит следующий ход.

Пусть именами игроков будут ПЛЮС и МИНУС. Будем использовать следующие обозначения:

X^S или Y^S – некоторая конфигурация игры, причем индекс S принимает значения + или –, указывая, кому принадлежит следующий ход (т.е. в конфигурации X^+ следующий ход должен делать игрок ПЛЮС, а в X^- – игрок МИНУС);

$W(X^S)$ – задача доказательства того, что игрок ПЛЮС может выиграть, исходя из конфигурации X^S;

$V(X^S)$ – задача доказательства того, что игрок МИНУС может выиграть, отправляясь от конфигурации X^S.

Рассмотрим сначала игровую задачу $W(X^S)$. Операторы сведения (редукции) этой задачи к подзадачам определяются исходя из ходов, допустимых в проводимой игре:

* Если в некоторой конфигурации X^+ очередной ход делает игрок ПЛЮС, и имеется N допустимых ходов, приводящих соответственно к конфигурациям X_1^-, X_2^-, ..., X_N^-, то для решения задачи $W(X^+)$ необходимо решить по крайней мере одну из подзадач $W(X_i^-)$. Действительно, так как ход выбирает игрок ПЛЮС, то он выиграет игру, если хотя бы один из ходов ведет к выигрышу – см. рис.16(а).

* Если же в некоторой конфигурации Y^- ход должен сделать игрок МИНУС и имеется K допустимых ходов, приводящих к конфигурациям Y_1^+, Y_2^+, ..., Y_K^+, то для решения задачи $W(Y^-)$ требуется решить каждую из возникающих подзадач $W(Y_i^+)$. Действительно, поскольку ход выбирает МИНУС, то ПЛЮС выиграет игру, если выигрыш гарантирован ему после любого хода противника – см. рис.16(б).

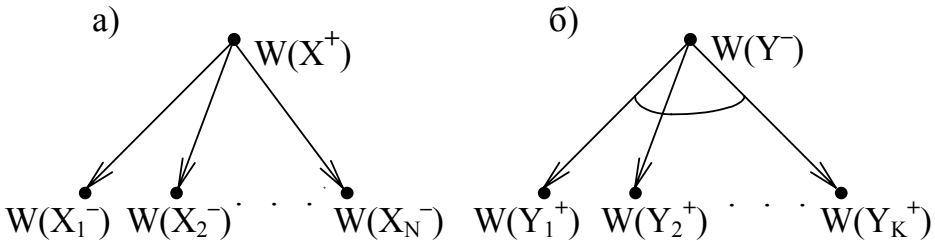


Рис. 16. Редукция игровых задач

Последовательное применение к исходной конфигурации игры данной схемы редукции порождает И/ИЛИ-дерево (И/ИЛИ-граф), которое называют ***деревом*** (***графом***) ***игры***. Дуги игрового дерева соответствуют ходам игроков, вершины – конфигурациям игры, причем листья дерева – это позиции, в которых игра завершается выигрышем, проигрышем или ничьей. Некоторые листья являются заключительными вершинами, соответствующими элементарным задачам – позициям, выигрышным для игрока ПЛЮС. Заметим, что для конфигураций, где ход принадлежит ПЛЮСу, в игровом дереве получается ИЛИ-вершина, а для позиций, в которых ходит МИНУС, получается И-вершина.

Цель построения игрового дерева или графа – получение решающего поддерева для задачи $W(X^S)$, показывающего, как игрок ПЛЮС может выиграть игру из позиции X^S независимо от ответов противника. Для этого могут быть применены разные алгоритмы поиска на И/ИЛИ-графах. Решающее дерево заканчивается на позициях, выигрышных для ПЛЮСа, и содержит полную стратегию достижения им выигрыша: для каждого возможного продолжения игры, выбранного противником, в дереве есть ответный ход, приводящий к победе.

Для задачи $V(X^S)$ схема сведения игровых задач к подзадачам аналогична: ходам игрока ПЛЮС будут соответствовать И-вершины, а ходам МИНУСа – ИЛИ-вершины, заключительные же вершины будут соответствовать позициям, выигрышным для игрока МИНУС.

Конечно, подобная редукция задач применима и в случае, когда нужно доказать существование ничейной стратегии в игре. При этом определение элементарной задачи должно быть соответствующим образом изменено.

Рассмотрим в качестве иллюстрации простую игру, называемую «последний проигрывает». Два игрока поочередно берут по одной или две монеты из кучки, первоначально содержащей семь монет. Игрок, забирающий последнюю монету, проигрывает.

На рис.17 показан полный граф игры для задачи $V(7^+)$, жирными дугами на нем выделен решающий И/ИЛИ-граф, который доказывает, что второй игрок (т.е. игрок МИНУС, начинающий вторым), всегда может выиграть. Конфигурация игры описана как число оставшихся в текущий момент монет, также указание, кому принадлежит следующий ход. Заключительные вершины, соответствующие элементарной задаче $V(1^+)$, т.е. выигрышу игрока МИНУС, в графе подчеркнуты.

Представленная в решающем графе выигрышная стратегия может быть сформулирована словесно так: если в очередном ходе игрок ПЛЮС берет одну монету, то в следующем ходе МИНУС должен взять две, а если ПЛЮС берет две монеты, то МИНУС должен забрать одну. Отметим, что для аналогичной задачи $W(7^+)$ решающий граф построить не удастся (начальная вершина неразрешима); таким образом, у игрока ПЛЮС нет выигрышной стратегии в этой игре.

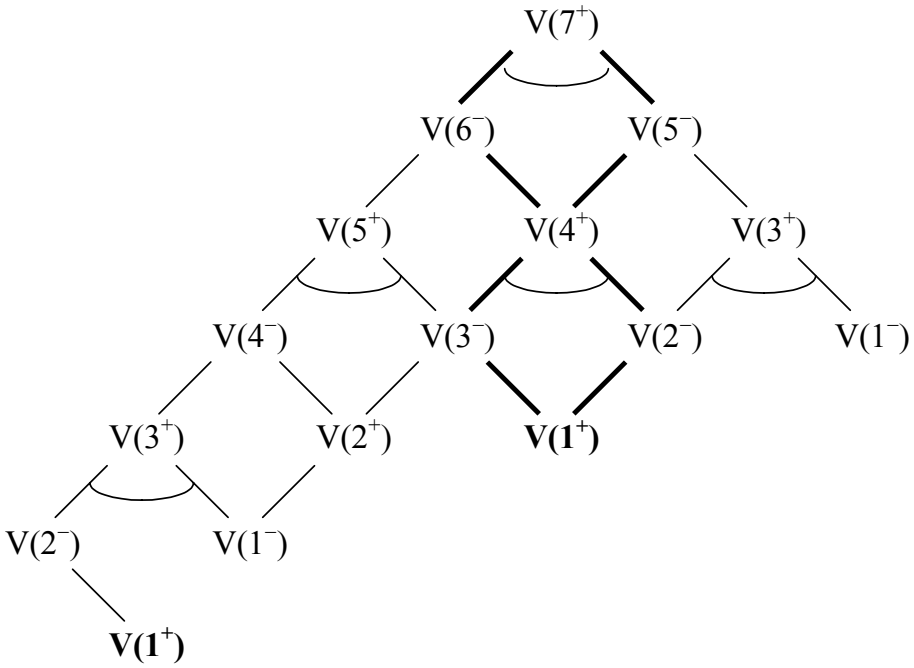


Рис. 17. Граф игры "последний проигрывает"

В большинстве игр, представляющих интерес, таких как шашки и шахматы, построить полные решающие деревья и найти полные игровые стратегии не представляется возможным. Например, для шашек число вершин в полном игровом дереве оценивается величиной порядка 10^{40}, и просмотреть такое дерево практически нереально. Алгоритмы же упорядоченного перебора с применением эвристик не настолько уменьшают просматриваемую часть дерева игры, чтобы дать существенное, на несколько порядков, сокращение времени поиска.

Тем не менее для неполных игр в шашки и шахматы (например, для эндшпилей), как и для всех несложных игр, таких как «крестики-нолики» на квадрате небольшого размера, можно успешно применять алгоритмы поиска на И/ИЛИ-графах, позволяющие обнаруживать выигрышные и ничейные игровые стратегии.

Рассмотрим, к примеру, игру «крестики-нолики» на квадрате 3×3. Игрок ПЛЮС ходит первым и ставит крестики, а МИНУС – нолики. Игра заканчивается, когда составлена либо строка, либо столбец, либо диагональ из крестиков (в этом случае выигрывает ПЛЮС) или ноликов (выигрывает МИНУС). Оценим размер полного дерева игры: начальная вершина имеет 9 дочерних вершин, каждая из которых в свою очередь имеет 8 дочерних; каждая вершина глубины 2 имеет 7 дочерних и т.д. Таким образом, число концевых вершин в дереве игры равно 9!=362880, но многие пути в этом дереве обрываются раньше на заключительных вершинах. Значит, в этой игре возможен полный просмотр дерева и нахождение выигрышной стратегии. Однако ситуация изменится при существенном увеличении размеров квадрата или же в случае неограниченного поля игры.

В таких случаях, как и во всех сложных играх, вместо нереальной задачи поиска полной игровой стратегии решается, как правило, более простая задача − поиск для заданной позиции игры *достаточно хорошего* первого хода.

Минимаксная процедура

С целью поиска достаточно хорошего первого хода просматривается обычно часть игрового дерева, построенного от заданной конфигурации. Для этого применяется один из переборных алгоритмов (вглубь, вширь или эвристический) и некоторое искусственное окончание перебора вершин в игровом дереве: например, ограничивается время перебора или же глубина поиска.

В построенном таким образом частичном дереве игры оцениваются его концевые вершины (листья), и по полученным оценкам определяется наилучший ход от заданной игровой конфигурации. При этом для оценивания концевых вершин полученного дерева используется так называемая ***статическая оценочная функция***, а для получения оценки остальных вершин – корневой (начальной) и промежуточных между корневой и концевыми – используется так называемый ***минимаксный принцип***.

Статическая оценочная функция, будучи применена к некоторой вершине-конфигурации игры, дает числовое значение, оценивающее различные достоинства этой игровой позиции. Например, для шашек могут учитываться такие (статические) элементы конфигурации игры, как продвинутость и подвижность шашек, количество дамок, контроль ими центра и проч. По сути, статическая функция вычисляет эвристическую оценку шансов на выигрыш одного из игроков. Для определенности будем рассматривать задачу выигрыша игрока ПЛЮС и соответственно поиска достаточно хорошего его первого хода от заданной конфигурации.

Будем придерживаться общепринятого соглашения, по которому значение статической оценочной функции тем больше, чем больше преимуществ имеет игрок ПЛЮС (над игроком МИНУС) в оцениваемой позиции. Очень часто оценочная функция выбирается следующим образом:

- статическая оценочная функция положительна в игровых конфигурациях, где игрок ПЛЮС имеет преимущества;
- статическая оценочная функция отрицательна в конфигурациях, где МИНУС имеет преимущества;
- статическая оценочная функция близка к нулю в позициях, не дающих преимущества ни одному из игроков.

Например, для шашек в качестве простейшей статической функции может быть взят перевес в количестве шашек (и дамок) у игрока ПЛЮС. Для игры «крестики-нолики» на фиксированном квадрате возможна такая статическая оценочная функция:

$$E(P) = \begin{cases} +\infty, & \text{если P есть позиция выигрыша игрока ПЛЮС} \\ -\infty, & \text{если P есть позиция выигрыша игрока МИНУС} \\ (N_L^+ + N_C^+ + N_D^+) - (N_L^- + N_C^- + N_D^-) & \text{в остальных случаях} \end{cases}$$

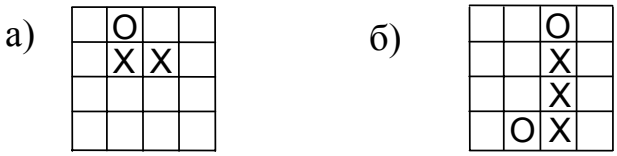
где: $+\infty$ – очень большое положительное число;

$-\infty$ – очень маленькое отрицательное число;

N_L^+, N_C^+, N_D^+ – соответственно число строк, столбцов и диагоналей, «открытых» для игрока ПЛЮС (т.е. где он еще может поставить три выигрышных крестика подряд);

N_L^-, N_C^-, N_D^- – аналогичные числа для игрока МИНУС.

На рис.18 приведены две игровые позиции (на квадрате 4×4) и соответствующие значения статической оценочной функции.



E (P) = 8 - 5 = 3 E (P) = 6 - 4 = 2

Рис. 18. Примеры статических оценок позиций игры "крестики-нолики"

Подчеркнем, что с помощью статической оценочной функции оцениваются только концевые вершины дерева игры, для оценок же промежуточных вершин, как и начальной вершины, используется минимаксный принцип, основанный на следующей простой идее. Если бы игроку ПЛЮС пришлось бы выбирать один из нескольких возможных ходов, то он выбрал бы наиболее сильный ход, т.е. ход, приводящий к позиции с наибольшей оценкой. Аналогично, если бы выбирать ход пришлось игроку МИНУС, то он выбрал бы ход, приводящий к позиции с наименьшей оценкой.

Сформулируем теперь сам минимаксный принцип:

* ИЛИ-вершине дерева игры приписывается оценка, равная максимуму оценок ее дочерних вершин;
* И-вершине игрового дерева приписывается оценка, равная минимуму оценок ее дочерних вершин.

Минимаксный принцип положен в основу ***минимаксной процедуры***, предназначенной для определения наилучшего (более точно, достаточно хорошего) хода игрока исходя из заданной конфигурации игры S при фиксированной глубине поиска N в игровом дереве. Предполагается, что игрок ПЛЮС ходит первым (т.е. начальная вершина есть ИЛИ-вершина). Основные этапы этой процедуры таковы:

1. Дерево игры строится (просматривается) одним из известных алгоритмов перебора (как правило, алгоритмом поиска вглубь) от исходной позиции S до глубины N ;
2. Все концевые вершины полученного дерева, т.е. вершины, находящиеся на глубине N, оцениваются с помощью статической оценочной функции;
3. В соответствии с минимаксным принципом вычисляются оценки всех остальных вершин: сначала вычисляются оценки вершин, родительских для концевых, затем - родительских для этих родительских вершин и т.д.; такое оценивание вершин продолжается при движении снизу вверх по дереву поиска до тех пор, пока не будут оценены вершины, дочерние для начальной вершины, т.е. для исходной конфигурации S;
4. Среди вершин, дочерних к начальной, выбирается вершина с наибольшей оценкой: ход, который к ней ведет, и есть искомый наилучший ход в игровой конфигурации S.

На рис.19 показано применение минимаксной процедуры для дерева игры, построенного до глубины N=3. Концевые вершины не имеют имен, они обозначены своими оценками – значениями статической оценочной функции. Числовые индексы имен остальных вершин показывают порядок, в котором эти вершины строились алгоритмом перебора вглубь. Рядом с этими вершинами находятся их минимаксные оценки, полученные при движении в обратном (по отношению к построению дерева) направлении. Таким образом, наилучший ход – первый из двух возможных.

На данном игровом дереве выделена ветвь (последовательность ходов игроков), представляющая так называемую *минимаксно-оптимальную* игру, при которой каждый из игроков всегда выбирает наилучший для себя ход. Заметим, что оценки всех вершин этой ветви дерева совпадают, и оценка начальной вершины равна оценке концевой вершины этой ветви.

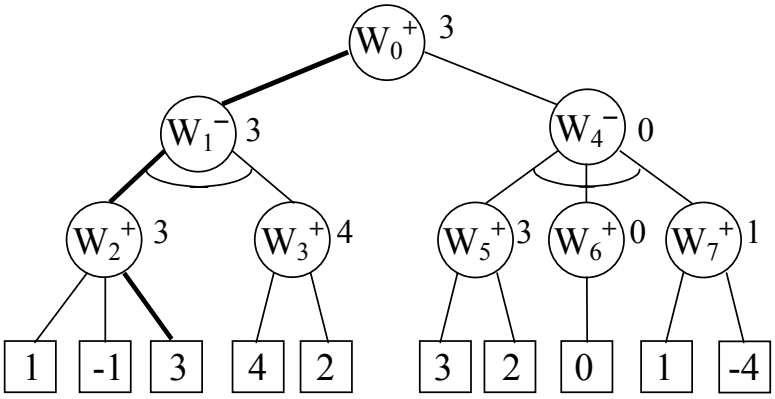


Рис. 19. Игровое дерево, построенное минимаксной процедурой

В принципе статическую оценочную функцию можно было бы применить и к промежуточным вершинам, и на основе этих оценок осуществить выбор наилучшего первого хода, например, сразу выбрать ход, максимизирующий значение статической оценочной функции среди вершин, дочерних к исходной. Однако считается, что оценки, полученные с помощью минимаксной процедуры, более надежные меры относительного достоинства промежуточных вершин, чем оценки, полученные прямым применением статической оценочной функции. Действительно, минимаксные оценки основаны на просмотре игры вперед и учитывают разные особенности, которые могут возникнуть в последующем, в то время как простое применение оценочной функции учитывает лишь свойства позиции как таковой. Это отличие статических и минимаксных оценок существенно для «активных», динамичных позиций игры (например, в шашках и шахматах к ним относятся конфигурации, в которых возникает угроза взятия одной или нескольких фигур). В случае же так называемых «пассивных» (спокойных) позиций статическая оценка может мало отличаться от оценки по минимаксному принципу.

Альфа-бета процедура

В минимаксной процедуре процесс построения частичного дерева игры отделен от процесса оценивания вершин, и это приводит к тому, что в целом минимаксная процедура оказывается неэффективной стратегией поиска хорошего первого хода. Для повышения эффективности поиска необходимо вычислять оценки (статические и минимаксные) вершин одновременно с построением игрового дерева и не рассматривать вершины, которые хуже уже построенных вершин. Это приводит к так называемой ***альфа-бета процедуре*** поиска наилучшего первого хода от заданной позиции, в основе которой лежит достаточно очевидное соображение для сокращения поиска: если есть два варианта хода одного игрока, то худший в ряде случаев можно сразу отбросить, не выясняя, насколько в точности он хуже.

Рассмотрим сначала идею работы альфа-бета процедуры на примере игрового дерева, приведенного на рис.19. Дерево игры строится до глубины N=3 алгоритмом перебора вглубь. Причем сразу, как это становится возможным, вычисляются не только статические оценки концевых вершин, но и *предварительные* минимаксные оценки промежуточных вершин. Предварительная оценка определяется соответственно как минимум или максимум уже известных к настоящему моменту оценок дочерних вершин. В общем случае, эта оценка может быть получена при наличии оценки хотя бы одной дочерней вершины. В ходе дальнейшего построения дерева игры и получения новых оценок вершин предварительные оценки постепенно уточняются, опять же по минимаксному принципу.

Пусть таким образом построены вершины W_1^-, W_2^+ и первые три конечные вершины (листья) – см. рис.20. Эти листья оценены статической функцией, и вершина W_2^+ получила точную минимаксную оценку 3, а вершина W_1^- – предварительную оценку 3. Далее при построении и раскрытии вершины W_3^+ статическая оценка первой ее дочерней вершины дает величину 4, которая становится предварительной оценкой самой вершины W_3^+. Эта предварительная оценка будет потом, после построения второй ее дочерней вершины, пересчитана. Причем согласно минимаксному принципу оценка может только увеличиться (поскольку подсчитывается как максимум

оценок дочерних вершин), но даже если она увеличится, это не повлияет на оценку вершины W_1^-, поскольку последняя при уточнении по минимаксному принципу может только уменьшаться (так как равна минимуму оценок дочерних вершин). Следовательно, можно пренебречь второй дочерней вершиной для W_3^+, не строить и не оценивать ее, поскольку уточнение оценки вершины W_3^+ не повлияет на оценку вершины W_1^-. Такое сокращение поиска в игровом дереве называется ***отсечением*** ветвей.

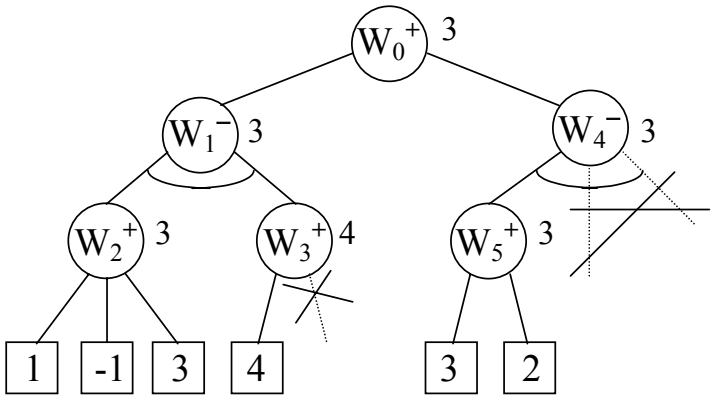


Рис. 20. Игровое дерево, построенное альфа-бета процедурой

Продолжим для нашего примера процесс поиска в глубину с одновременным вычислением предварительных (и точных, где это возможно) оценок вершин вплоть до момента, когда построены уже вершины W_4^- , W_5^+ и две дочерних последней, которые оцениваются статической функцией. Исходя из оценки первой дочерней вершины начальная вершина W_0^+, соответствующая исходной позиции игры, к этому моменту уже предварительно оценена величиной 3. Вершина W_5^+ получила точную минимаксную оценку 3, а ее родительская W_4^- получила пока только предварительную оценку 3. Эта предварительная оценка вершины W_4^- может быть уточнена в дальнейшем, но в соответствии с минимаксным принципом возможно только ее уменьшение, и это уменьшение не повлияет на оценку вершины W_0^+, поскольку последняя, опять же согласно минимаксному принципу, может только увеличиваться. Таким образом, построение дерева можно прервать ниже вершины W_4^-, отсекая целиком выходящие из нее вторую и третью ветви (и оставляя ее оценку предварительной).

На этом построение игрового дерева заканчивается, полученный результат – лучший первый ход – тот же самый, что и при минимаксной процедуре. У некоторых вершин дерева осталась неуточненная, предварительная оценка, однако этих приближенных оценок оказалось достаточно для того, чтобы определить точную минимаксную оценку начальной вершины и наилучший первый ход. В то же время произошло существенное сокращение поиска: вместо 17 вершин построено только 11 и вместо 10 обращений к статической оценочной функции понадобилось всего 6.

Обобщим рассмотренную идею сокращения перебора. Сформулируем сначала правила вычисления оценок вершин дерева игры, в том числе предварительных оценок промежуточных вершин, которые для удобства будем называть ***альфа-*** и ***бета-величинами***:

- концевая вершина игрового дерева оценивается статической оценочной функцией сразу, как только она построена;
- промежуточная вершина предварительно оценивается по минимаксному принципу, как только стала известна оценка хотя бы одной из ее дочерних вершин; каждая предварительная оценка пересчитывается (уточняется) всякий раз, когда получена оценка еще одной дочерней вершины;
- предварительная оценка ИЛИ-вершины (альфа-величина) полагается равной наибольшей из вычисленных к текущему моменту оценок ее дочерних вершин;
- предварительная оценка И-вершины (бета-величина) полагается равной наименьшей из вычисленных к текущему моменту оценок ее дочерних вершин.

Укажем очевидное следствие этих правил вычисления: альфа-величины не могут уменьшаться, а бета-величины не могут увеличиваться.

Сформулируем теперь правила прерывания перебора, или отсечения ветвей игрового дерева:

Перебор можно прервать ниже любой И-вершины, бета-величина которой не больше, чем альфа-величина одной из предшествующих ей ИЛИ-вершин (включая корневую вершину дерева);

Перебор можно прервать ниже любой ИЛИ-вершины, альфа-величина которой не меньше, чем бета-величина одной из предшествующих ей И-вершин.

В случае А говорят, что имеет место ***альфа-отсечение***, поскольку отсекаются ветви дерева, начиная с ИЛИ-вершин, которым приписана альфа-величина, а в случае В – ***бета-отсечение***, поскольку отсекаются ветви, начинающиеся с бета-величин.

Важно, что рассмотренные правила работают в ходе построения игрового дерева вглубь; это означает, что предварительные оценки промежуточных вершин появляются лишь по мере продвижения от концевых вершин дерева вверх к корню и реально отсечения могут начаться только после того, как получена хотя бы одна точная минимаксная оценка промежуточной вершины.

В рассмотренном на рис.20 примере первое прерывание перебора было бета-отсечением, а второе – альфа-отсечением. Причем в обоих случаях отсечение было неглубоким, поскольку необходимая для соблюдения соответствующего правила отсечения альфа- или бета-величина находилась в непосредственно предшествующей к точке отсечения вершине. В общем же случае она может находиться существенно выше отсекаемой ветви – где-то на пути от вершины, ниже которой производится отсечение, к начальной вершине дерева, включая последнюю.

После прерывания перебора предварительные оценки вершин в точках отсечения остаются неуточненными, но, как уже отмечалось, это не препятствует правильному нахождению предварительных оценок всех предшествующих вершин, как и точной оценки корневой вершины и ее дочерних вершин, а значит, и искомого наилучшего первого хода.

На приведенных выше правилах вычисления оценок вершин и выполнения отсечений (всюду, где это возможно) основана альфа-бета процедура, являющаяся более эффективной реализацией минимаксного принципа. В принципе, используя математическую индукцию и рассуждая, как показано на примере, несложно доказать следующее

Утверждение:

Альфа-бета процедура всегда приводит к тому же результату (наилучшему первому ходу), что и простая минимаксная процедура той же глубины.

Важным является вопрос о том, насколько в среднем альфа-бета процедура эффективнее обычной минимаксной процедуры. Нетрудно заметить, что количество отсечений в альфа-бета процедуре зависит от степени, в которой предварительные оценки (альфа- и бета-величины), полученные первыми, аппроксимируют окончательные минимаксные оценки: чем ближе эти оценки, тем больше отсечений и меньше перебор. Это положение иллюстрирует пример на рис. 20, в котором ветвь дерева с минимаксно-оптимальной игрой строится практически в самом начале поиска.

Таким образом, эффективность альфа-бета процедуры зависит от порядка построения и раскрытия вершин в дереве игры. Возможен и неудачный порядок просмотра, при котором придется пройти без отсечений через все вершины дерева, и в этом, худшем, случае альфа-бета процедура не будет иметь никаких преимуществ по сравнению с минимаксной процедурой.

Наибольшее число отсечений достигается, когда при переборе в глубину первой обнаруживается конечная вершина, статическая оценка которой станет в последствии минимаксной оценкой начальной вершины. При максимальном числе отсечений строится и оценивается минимальное число концевых вершин. Показано [Нильсон73, раздел 5.14], что в случае, когда самые сильные ходы рассматриваются первыми, количество концевых вершин глубины N, которые будут построены и оценены альфа-бета процедурой, примерно равно числу концевых вершин, которые были бы построены и оценены на глубине N/2 обычной минимаксной процедурой. Таким образом, при фиксированном времени и памяти альфа-бета процедура сможет пройти при поиске вдвое глубже по сравнению с обычной минимаксной процедурой.

В заключение отметим, что статическая оценочная функция и альфа-бета процедура – две непременные составляющие почти всех компьютерных игровых программ (в том числе коммерческих). Часто используются также дополнительные эвристические приемы в самой альфа-бета процедуре [Слейгл, глава 2; Нильсон 73, раздел 5.14]:

- направленное (эвристическое) отсечение неперспективных ветвей: например, построение дерева игры обрывается на «пассивных» позициях и к ним применяется статическая оценочная функция, для «активных» же позиций поиск продолжается дальше до нужной глубины или даже глубже, поскольку на это можно использовать время, сэкономленное вследствие отсечения ветвей;
- последовательное углубление, при котором альфа-бета процедура применяется неоднократно: сначала до некоторой небольшой глубины (например, 2), а затем глубина увеличивается с каждой итерацией, причем после каждой итерации выполняется переупорядочение всех дочерних вершин – с тем, чтобы увеличить число отсечений в последующих итерациях;
- фиксированное упорядочение вершин при спуске-построении дерева вглубь, при котором в первую очередь строится и раскрывается дочерняя вершина, оцениваемая как более перспективная (эта оценка может быть проведена как с помощью статической оценочной функции, так и более простой, хотя и менее надежной эвристической функции);
- динамическое упорядочение вершин, при котором каждый раз после уточнения минимаксных оценок и проведения отсечений производится переупорядочивание вершин во всем построенном к текущему моменту дереве (с помощью некоторой эвристической функции) и для дальнейшего раскрытия выбирается наиболее перспективная вершина (заметим, что по существу это означает переход от классического перебора вглубь к алгоритму упорядоченного перебора на И/ИЛИ-графах).

Для усиления игры могут быть также использованы библиотеки типовых игровых ситуаций [Братко, глава 15] и другие идеи [Лорьер, глава 6].

Плэнер-алгоритмы поиска на игровых деревьях

Рассмотрим сначала описание на языке Плэнер функции MIN_MAX, реализующей минимаксную процедуру. Аргументы этой функции: Instate – исходная позиция игры, для которой ищется наилучший ход; N – глубина поиска, т.е. количество ходов вперед. Вырабатываемое функцией значение – это дочерняя для Instate позиция, соответствующая искомому наилучшему ходу.

```
[define MIN_MAX (lambda (Instate N)
   [SELECT_MAX [MM_EVALP .Instate 0 T]] )]
```

Функция MIN_MAX использует две вспомогательные функции: SELECT_MAX и MM_EVALP. Первая функция, SELECT_MAX выбирает из своего единственного аргумента-списка, состоящего из позиций и их оценок, позицию с наибольшей оценкой:

```
[define SELECT_MAX (lambda (List)
  [prog (Elem  Max_elem)
      [fin Max_elem List]
   SM [cond ([fin Elem List] [return [2 .Max_elem]])
            ([gt [1 .Elem] [1 .Max_elem]]
                     [set Max_elem .Elem])]
      [go SM]] )]
```

Функция MM_EVALP с тремя аргументами является главной рекурсивной функцией, оценивающей вершины дерева игры по минимаксному принципу. На каждом шаге рекурсии она оценивает вершину-позицию Position, находящуюся на глубине Depth и имеющую тип Deptype: "ИЛИ" при Deptype=T и "И" при Deptype=(). Исходная позиция (корневая вершина дерева) имеет тип T и находится на глубине 0. Значением функции MM_EVALP является либо вычисленная оценка вершины Position (при Depth>0) либо список дочерних для Position позиций с их числовыми оценками (при Depth=0).

При работе MM_EVALP используются вспомогательные функции, определение которых зависит от конкретной игры: OPENING, вычисляющая для заданной позиции игры список дочерних вершин-позиций, и STAT_EST – статическая оценочная функция.

```
[define MM_EVALP (lambda (Position Depth Deptype)
  [prog (D  %дочерняя позиция;
         (Movelist ())   %список ходов-позиций);
         Pvalue Dvalue)  %оценки текущей и дочерней
                          позиций;
% 1: установка развилки, включающей все дочерние вершины текущей позиции (список () добавлен в развилку, чтобы "поймать" момент ее закрытия);
   [set D [among (<OPENING .Position> ())]]
% 2: если развилка закрыта - возврат функцией подсчитанной оценки;
   [cond ([empty .D]
            [return [cond ([eq .Depth 0] .Movelist)
                          (t .Pvalue) ]] )]
% 3: вычисление оценки очередной дочерней позиции: либо применение статической функции, либо рекурсивный спуск;
      [cond ([eq .Depth [- .N 1]]
               [set Dvalue [STAT_EST .D]])
            (t [set Dvalue [MM_EVALP .D [+ 1 .Depth]
                                    [not .Deptype]]] )]
% 4: пересчет оценки текущей позиции по минимаксному принципу;
   [cond ([hasval Pvalue] [pset Pvalue
              [cond (.Deptype [max .Pvalue .Dvalue])
                    (t [min .Pvalue .Dvalue]) ]])
            (t [pset Pvalue .Dvalue]) ]
% 5: при необходимости пересчет для исходной позиции списка ходов(дочерних позиций) с их оценками;
      [cond ([eq .Depth 0]
              [pset Movelist (!.Movelist (.Dvalue .D))] )]
% 6: возврат к другой альтернативе развилки;
   [fail]  ])]
```

Теперь опишем на Плэнере функцию ALPHA_BETA, реализующую альфа-бета процедуру. Она имеет сходную с MIN_MAX структуру:

```
[define ALPHA_BETA (lambda (Instate N)
   [SELECT_MAX [AB_EVALP .Instate 0 T () () ]] )]
```

Аргументы этой функции: Instate – заданная позиция игры, для которой ищется лучший ход, N – глубина поиска. Значением функции ALPHA_BETA является дочерняя для Instate вершина-позиция, соответствующая наилучшему ходу.

Как и MIN_MAX, функция ALPHA_BETA использует две вспомогательные функции: SELECT_MAX и AB_EVALP. Первая из этих функций была определена выше, она выбирает из своего аргумента-списка позицию с наибольшей оценкой.

Вторая функция, AB_EVALP – главная рекурсивная функция, которая эффективно оценивает вершины дерева игры по минимаксному принципу с выполнением отсечений. Она имеет пять аргументов: Pos, Depth, Deptype, Amax, Bmin, и на каждом шаге рекурсии вычисляет оценку вершины-позиции Pos, находящейся на глубине Depth и имеющей тип Deptype ("ИЛИ" при Deptype=T и "И" при Deptype=()). Для проведения отсечений функция использует вспомогательные величины Amax и Bmin, являющиеся соответственно максимальной из альфа-величин и минимальной из бета-величин вершин, предшествующих оцениваемой вершине Pos. Значением функции AB_EVALP (как и MM_EVALP) является либо вычисленная оценка позиции Pos (при Depth>0), либо список дочерних для Pos позиций с их числовыми оценками (при Depth=0).

При первом обращении к функции AB_EVALP (внутри функции ALPHA_BETA) смысл ее аргументов такой: исходная позиция Pos (т.е. корневая вершина) имеет тип T, находится на глубине 0, величины Amax и Bmin неизвестны. Эти величины необходимы для проверки условий правил отсечений, поэтому отсечения не могут быть выполнены, пока величины не определены (альфа-отсечение – пока неизвестна величина Bmin, а бета-отсечение – пока неизвестна Amax). Эти значения станут известны только лишь когда часть дерева игры будет просмотрена до заданной глубины.

Функция AB_EVALP (как и MM_EVALP) использует вспомогательные функции, определение которых зависит от конкретной игры: OPENING, вычисляющую для заданной позиции игры список дочерних вершин-позиций, и STAT_EST – статическую оценочную функцию.

```
[define AB_EVALP (lambda (Pos Depth Deptype Amax Bmin)
  [prog (D (Movelist ()) Pvalue Dvalue)
% 1: установка развилки из всех дочерних для Pos вершин (список () добавлен в развилку, чтобы "поймать" момент ее закрытия);
   [set D [among (<OPENING .Pos> ())]]
% 2: если развилка закрыта, то возврат функцией подсчитанной оценки;
   [cond ([empty .D]
            [return [cond ([eq .Depth 0] .Movelist)
                          (t .Pvalue)] ] )]
% 3: проверка возможности отсечения и отсечение - возврат на предыдущий уровень рекурсии (встроенная функция permex реализует досрочный выход с заданным значением из заданной функции, при этом уничтожает все развилки, возникшие после входа в заданную функцию);
   [cond ([and [neq .Depth 0] [hasval Pvalue]]
           [cond (.Deptype
             [cond ([and [num .Bmin] [ge .Pvalue .Bmin]]
                    [permex .Pvalue AB_EVALP]) ])
                 (t
             [cond ([and [num .Amax] [le .Pvalue .Amax]]
                    [permex .Pvalue AB_EVALP])] )])]
% 4: вычисление оценки очередной дочерней вершины: либо применение статической оценочной функции, либо рекурсивное обращение к рассматриваемой функции;
   [cond ([eq .Depth [- .N 1]]
            [set Dvalue [STAT_EST .D]])
          (t [set Dvalue [AB_EVALP .D  [+ 1 .Depth]
                                [not .Deptype]
          [cond ([and [hasval Pvalue] .Deptype]
                  [cond ([num .Amax] [max .Amax .Pvalue])
                        (t .Pvalue)])
                (t .Amax)]
          [cond ([and [hasval Pvalue] [not .Deptype]]
                  [cond ([num .Bmin] [min .Bmin .Pvalue])
                        (t .Pvalue) ])
                (t .Bmin)] ]] )]
% 5: пересчет предварительной оценки текущей позиции по минимаксному принципу;
   [cond ([hasval Pvalue] [Pset  Pvalue
                  [cond (.Deptype [max .Pvalue .Dvalue])
                        (t [min .Pvalue .Dvalue])] ])
          (t [pset Pvalue .Dvalue] )]
% 6: в случае исходной позиции пересчитываем список ходов (дочерних позиций) с их оценками;
   [cond ([eq .Depth 0]
            [pset Movelist (!.Movelist (.Dvalue .D))] )]
% 7: возврат к другой альтернативе развилки ;
   [fail] ])]
```

Обе рассмотренные процедуры, MIN_MAX и ALPHA_BETA, должны быть соответствующим образом модифицированы, если при просмотре позиций на заданную глубину N возможно появление выигрышной или проигрышной позиции, обрывающих некоторые ветви игрового дерева. В этом случае к таким позициям необходимо сразу применить статическую оценочную функцию.

Заметим в заключении, что в случае выбора языка Лисп для реализации процедур поиска на игровых деревьях минимаксная процедура может быть запрограммировано весьма кратко и понятно, особенно при использовании функционалов. Что же касается альфа-бета процедуры, то программа на Лиспе [Уинстон, глава 13] существенно более громоздка и трудна для понимания, чем приведенная выше плэнер-функция, поскольку в последней применяется встроенный механизм возвратов.

Редукция задач

Основные понятия

Кроме уже рассмотренного подхода – представления задач в пространстве состояний – для решения ряда задач возможен и другой, более сложный подход. При этом подходе производится анализ исходной задачи с целью выделения такого набора подзадач, решив которые, мы решим исходную задачу. Каждая из выделенных подзадач в общем случае является более простой, чем исходная, и может быть решена каким-либо методом, в том числе – с использованием пространства состояний. Но можно продолжить процесс, последовательно выделяя из возникающих задач свои подзадачи – до тех пор, пока не получим ***элементарные задачи***, решение которых уже известно. Такой путь называется подходом, основанным на ***сведении задач к подзадачам***, или на ***редукции задач.***

Для иллюстрации этого подхода рассмотрим один из вариантов известной головоломки – задачи о ханойской башне, или пирамидке. В ней используются 3 колышка (обозначим их буквами A, B, C) и 3 диска разного диаметра, которые можно нанизывать на колышки через отверстия в центре. В начале все диски расположены на колышке A, причем диски меньшего диаметра лежат на дисках большего диаметра – см. рис. 6 (диски перенумерованы в порядке возрастания диаметра). Требуется переместить все диски на колышек C, соблюдая следующие правила. Перемещать можно только самый верхний диск, и нельзя никакой диск класть на диск меньшего размера. На рис.6 показаны начальное и целевое состояния задачи о ханойской башне.

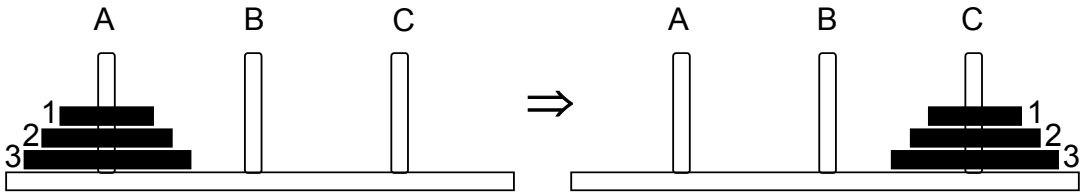


Рис. 6. Задача о ханойской башне

Эта задача легко может быть формализована в пространстве состояний: состояние задачи задается списком из трех элементов, каждый из которых указывает местоположение соответствующего диска (первый элемент – первого диска, второй – второго, третий – третьего). Начальное состояние описывается списком (AAA), а целевое – (CCC). При этом предполагается, что если на одном колышке находится более одного диска, то любой больший диск расположен ниже любого меньшего.

Полное пространство состояний задачи о пирамидке представлено графом на рис.7, включающем 27 вершин. Путь в этом графе, выделенный жирными дугами со стрелками, представляет собой решение задачи, которое включает 7 перемещений дисков.

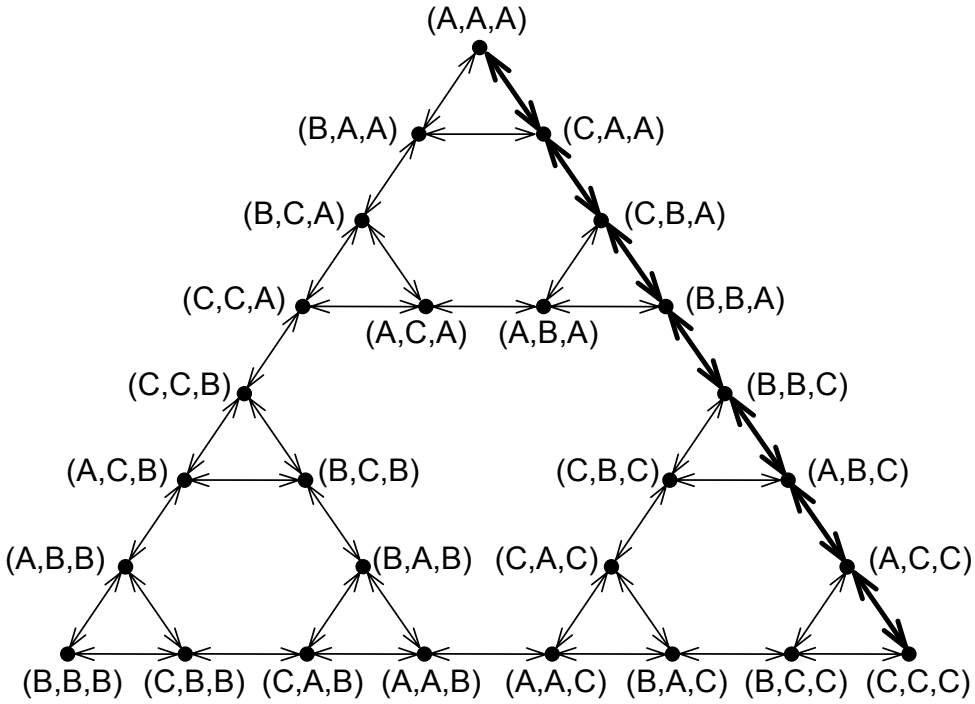


Рис. 7. Пространство состояний в задаче о ханойской башне

Это решение можно обнаружить перебором всех возможных перемещений дисков, но метод редукции задач позволяет решить рассматриваемую задачу быстрее. Ключевая идея редукции состоит в том, что для перемещения всей пирамидки необходимо переложить самый нижний диск 3, а это возможно, только если располагающаяся над ним пирамидка из двух меньших дисков 1 и 2 перенесена на колышек B – см. рис. 8(а).

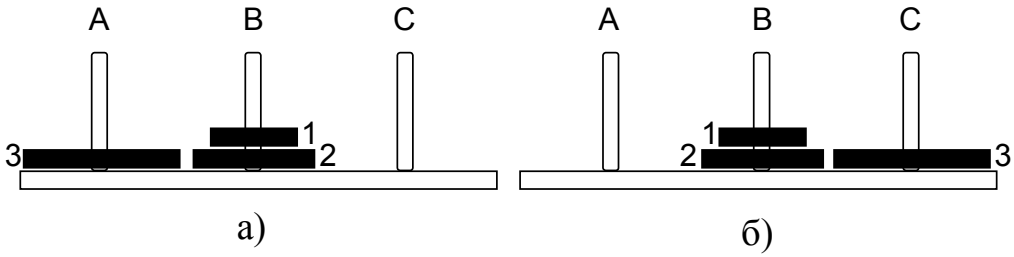


Рис. 8. Задача о ханойской башне: два решающих состояния

Таким образом, исходную задачу можно свести к трем следующим подзадачам:

1) переместить диски 1 и 2 с колышка A на колышек B (1, 2 : A→B);
2) переместить диск 3 с колышка A на колышек C (3 : A→C);
3) переместить диски 1 и 2 с колышка B на колышек C (1, 2 : B→C).

На рис. 8(б) показано исходное состояние для третьей задачи.

Каждая из трех указанных задач проще исходной. Действительно, в первой и в третьей задачах требуется переместить всего два диска, вторая же задача может рассматриваться как элементарная, так как ее решение состоит ровно из одного хода – перемещения диска 3 на колышек C. В первой и третьей задачах можно вновь применить метод редукции задач, и свести их к элементарным задачам. Весь процесс редукции можно схематически представить в виде дерева на рис. 9. Вершины дерева соответствуют решаемым задачам/подзадачам, причем листья дерева соответствуют элементарным задачам перемещения дисков, а дуги связывают редуцируемую задачу с ее подзадачами.

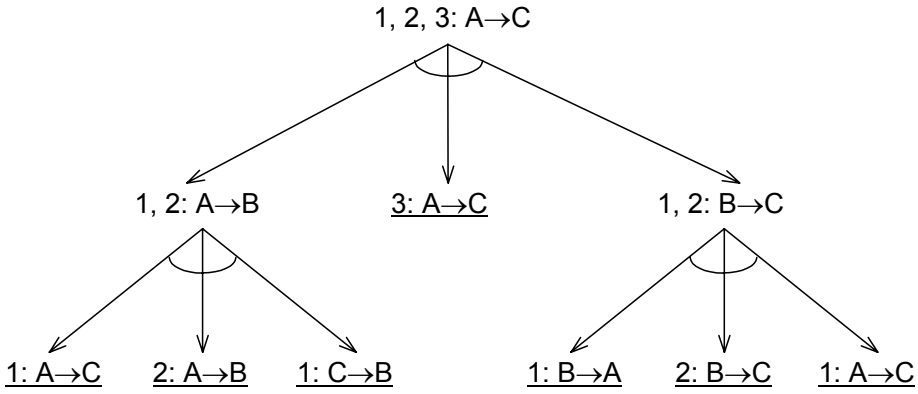


Рис. 9. Редукция задачи о ханойской башне

Заметим, что рассмотренная идея сведения задачи к совокупности подзадач может быть применена и в случае, когда начальная конфигурация задачи о пирамидке содержит не три, а большее число дисков. Таким образом, в случае подхода, основанного на редукции задач, мы получаем также пространство, но состоящее не из состояний, а из задач/подзадач (точнее, их описаний). При этом роль, аналогичную операторам в пространстве состояний, играют операторы, сводящие задачи в подзадачи. Точнее, каждый ***оператор редукции*** преобразует описание задачи в описание множества подзадач, причем это множество таково, что решение всех подзадач обеспечивает решение редуцированной задачи.

При решении задач методом редукции, как и при решении в пространстве состояний, может возникнуть необходимость перебора. Действительно, на каждом этапе редукции может оказаться несколько применимых операторов (т.е. способов сведения задачи к подзадачам) и, соответственно, несколько альтернативных множеств подзадач. Некоторые способы, возможно, не приведут к решению исходной задачи, поскольку могут обнаружиться неразрешимые подзадачи, другие же способы могут дать окончательное решение. В общем случае для полной редукции исходной задачи необходимо перепробовать несколько операторов. Процесс редукции продолжается, пока исходная задача не будет сведена к набору элементарных задач, решение которых известно.

Аналогично представлению в пространстве состояний, формализация задачи в рамках подхода, основанного на редукции задач, включает определение следующих составляющих:

- формы описания задач/подзадач и описание исходной задачи;
- множества операторов и их воздействий на описания задач;
- множества элементарных задач.

Эти составляющие задают неявно ***пространство задач***, в котором требуется провести поиск решения задачи.

Что касается формы описания задач/подзадач, то часто их удобно описывать в терминах пространства состояний, т.е. задавая начальное состояние и множество операторов, а также целевое состояние или его свойства. В этом случае элементарными задачами могут быть, к примеру, задачи, решаемые за один шаг в пространстве состояний.

Если выбрать такую форму описания в задаче о пирамидке, то элементарная задача перекладывания самого большого диска с колышка A на C записывалась бы как (BBA)⇨(BBC), а исходная задача – как (AAA)⇨(CCC). Причем, поскольку множество операторов предполагается одним и тем же для всех задач/подзадач, то в их описаниях оно в явном виде может не фигурировать. При выбранной форме описания задач результирующие подзадачи естественно интерпретируются как задачи нахождения пути между определенными состояниями-вехами в пространстве состояний. К примеру, такими вехами являются состояния (BBA) и (BBC) задачи о пирамидке, изображенные на рис.8. Они обладают тем свойством, что через них пройдет и искомый решающий путь.

В дополнение отметим, что подход с использованием пространства состояний можно рассматривать как вырожденный случай подхода, основанного на редукции задач, так как применение оператора в пространстве состояний сводит обычно исходную задачу к несколько более простой задаче, т.е. редуцирует ее. При этом результирующее множество подзадач состоит только из одного элемента, т.е. мы имеем простейший случай замены редуцируемой задачи на ей эквивалентную.

И/ИЛИ графы. Решающий граф

Для изображения процесса редукции задач и получающихся при этом альтернативных множеств подзадач используются обычно графоподобные структуры, вершины которых представляют описания задач и подзадач, а каждая дуга связывает пару вершин, соответствующих редуцируемой задаче и одной из результирующих подзадач, причем стрелки на дугах указывают направление редукции. Пример такой структуры приведен на рис.10(а): задача G может быть решена путем решения либо задач D_1 и D_2, либо E_1, E_2

и E_3 либо задачи F. При этом ребра, относящиеся к одному и тому же множеству подзадач, связываются специальной дугой. Чтобы сделать такую структуру более наглядной, вводятся дополнительные промежуточные вершины, и каждое множество результирующих задач группируется под своей родительской вершиной. При этом структура на рис.10(а) преобразуется в структуру, изображенную на рис.10(б): для двух из трех альтернативных множеств подзадач добавлены соответственно вершины D и E.

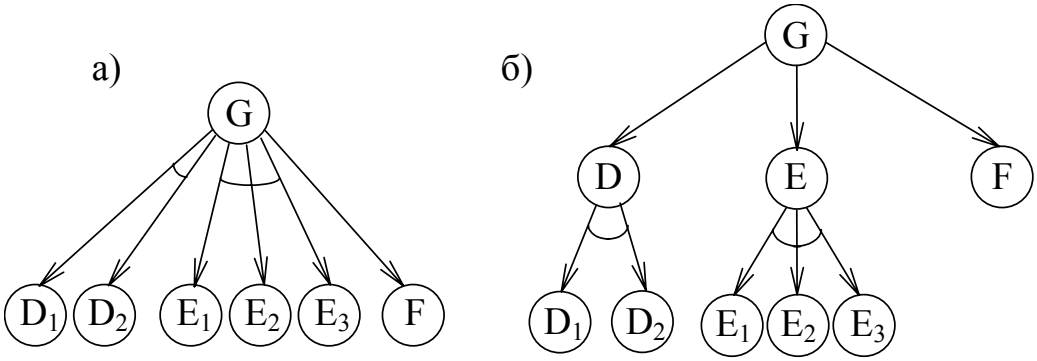


Рис. 10. Схематическое изображение редукции задач

Если считать, что вершины D и E соответствуют описаниям альтернативных путей решения исходной задачи, то вершину G можно назвать ***ИЛИ-вершиной***, так как задача G разрешима или способом D, или способом E, или способом F. Аналогично вершины D и E можно назвать ***И-вершинами***, поскольку каждый из соответствующих им способов требует решения всех подчиненных задач, что и обозначается специальной дугой. По этой причине структуры, подобные структурам, изображенным на рис.10(б) и рис.11, называются ***И/ИЛИ-графами*** (или ***графами целей***).

Если некоторая вершина такого графа имеет непосредственно следующие за ней (дочерние) вершины, то либо все они являются И-вершинами, либо все они – ИЛИ-вершины. Заметим, что если у некоторой вершины И/ИЛИ-графа имеется ровно одна дочерняя вершина, то последнюю можно считать как И-вершиной, так и ИЛИ-вершиной – такова, например, вершина F на рис.10(б).

На языке И/ИЛИ-графов применение нескольких операторов редукции задачи будет означать, что сначала будет построена промежуточная И-вершина, а затем непосредственно следующие за ней ИЛИ-вершины подзадач. Исключение составляет случай, когда множество задач состоит только из одного элемента, в этом случае будет образована ровно одна вершина, будем для определенности считать ее ИЛИ-вершиной.

Вершину И/ИЛИ-графа, соответствующую описанию исходной задачи, будем называть ***начальной*** вершиной. Вершины же, которые соответствуют описаниям элементарных задач, будем называть ***заключительными*** вершинами. В графе, показанном на рис.11, начальной является вершина P_0, а заключительными – вершины P_1, P_4, P_5, P_7 и P_8 (они изображены жирными кружками).

Поиск решения задачи, осуществляемый путем перебора вершин графа, применим и в подходе, основанном на редукции задач. Цель поиска на И/ИЛИ-графе – показать, что ***разрешима*** исходная задача, т.е. начальная вершина. Разрешимость этой вершины зависит от разрешимости других вершин графа.

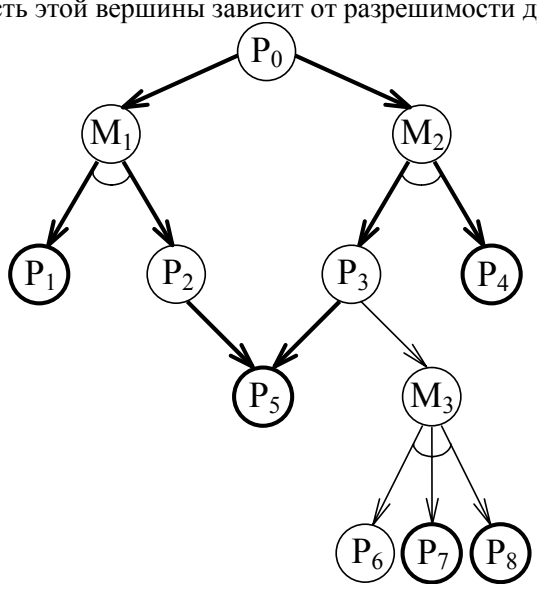


Рис. 11. Пример И/ИЛИ-графа

Сформулируем общее рекурсивное определение разрешимости вершины в И/ИЛИ-графе:

- заключительные вершины разрешимы, так как они соответствуют элементарным задачам;
- ИЛИ-вершина, не являющаяся заключительной, разрешима тогда и только тогда, когда разрешима по крайней мере одна из ее дочерних вершин;
- И-вершина, не являющаяся заключительной, разрешима тогда и только тогда, когда разрешима каждая из ее дочерних вершин.

Если в процессе поиска удалось показать, что начальная вершина разрешима, то это значит, что обнаружено решение исходной задачи, которое заключено в так называемом решающем графе. ***Решающий граф*** – это подграф И/ИЛИ-графа, состоящий только из разрешимых вершин и доказывающий разрешимость начальной вершины.

Для И/ИЛИ-графа, изображенного на рис.11, разрешимыми являются (кроме заключительных) вершины M_1, M_2, P_2, P_3. Этот граф содержит два решающих графа: первый состоит из вершин P_0, M_1, P_1, P_2 и P_5 ; а второй – из вершин P_0, M_2, P_3, P_4 и P_5. Заметим, что вершины M_3 и P_6 не являются разрешимыми (M_3 неразрешима в силу неразрешимости вершины P_6).

Пример: задача символьного интегрирования

В качестве примера применения метода редукции рассмотрим решение задачи символьного интегрирования, т.е. нахождения неопределенного интеграла $\int F(x)dx$. Обычно эта задача решается путем последовательного преобразования интеграла к выражению, содержащему известные табличные интегралы. Для этого используется несколько правил интегрирования, в том числе: правило интегрирования суммы функций, правило интегрирования по частям, правило вынесения постоянного множителя за знак интегрирования, а также применение алгебраических и тригонометрических подстановок и использование различных алгебраических и тригонометрических тождеств.

Для формализации этой задачи в рамках подхода, основанного на редукции задач, необходимо определить форму описания задач/подзадач, операторы редукции и элементарные задачи. В качестве возможной формы описания задач может быть взята символьная строка, содержащая запись подынтегральной функции и переменной интегрирования (если последняя не фиксирована заранее). Операторы редукции будут основаны, очевидно, на упомянутых правилах интегрирования. Например, правило интегрирования по частям

$$\int u dx = u\int dv - \int v du$$

сводит исходную задачу (неопределенный интеграл в левой части равенства) к двум подзадачам интегрирования (два соответствующих интеграла в правой части равенства).

Заметим, что часть получаемых таким образом операторов редукции (как, например, операторы, соответствующие правилу интегрирования по частям или правилу интегрирования суммы функций), действительно редуцируют исходную задачу и порождают И-вершину в И/ИЛИ-графе, в то время как алгебраические и тригонометрические подстановки и тождества (как, например, деление числителя на знаменатель или дополнение до полного квадрата) лишь заменяют одно подынтегральное выражение на другое, порождая, таким образом, ИЛИ-вершины.

Элементарные задачи интегрирования соответствуют табличным интегралам, например:

$$\int \sin x \, dx = -\cos x + C$$

Отметим, что поскольку каждая из таких табличных формул содержит переменные, на самом деле она является схемой, задающей бесконечное множество элементарных задач.

Особенностью задачи интегрирования является то, что, например, при интегрировании по частям может оказаться несколько способов разбиения исходного подынтегрального выражения на части и соответственно несколько способов применения этого правила интегрирования. Это означает, что в общем случае для одного правила возможно несколько вариантов редукции задачи, т.е. несколько способов применения одного и того же оператора редукции.

Другая особенность рассматриваемой задачи состоит в том, что на каждом шаге редукции применимо обычно большое количество операторов (включая несколько применений одного и того же оператора), и получающийся И/ИЛИ-граф задачи слишком велик даже для несложных задач интегрирования. Поэтому, чтобы сделать поиск на таком графе достаточно эффективным, необходимо как-то ограничивать и/или упорядочивать множество порождаемых при поиске вершин. К примеру, можно упорядочить операторы редукции по степени их полезности, и приписать больший приоритет операторам, соответствующим правилам интегрирования суммы и интегрирования по частям.

На рис.12 показан решающий граф для одной задачи интегрирования. В вершинах графа указаны соответствующие задачи/подзадачи, заключительные вершины заключены в двойные рамки. Решение задачи может быть собрано из содержащейся в графе информации. Оно состоит из следующих шагов:

1. применения подстановки z = tg(x);
2. эквивалентного преобразования подынтегрального выражения;

3. применения правила интегрирования суммы функций, причем одна из трех результирующих подзадач оказывается элементарной, а две остальные решаются за один шаг (первая – путем вынесения постоянного множителя за знак интеграла, вторая – применением подстановки z =tg (w)).

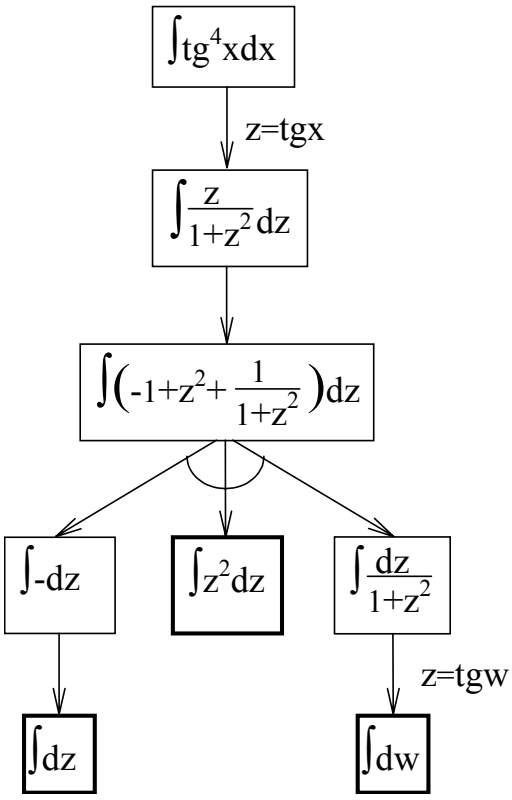


Рис. 12. Решающий граф для одной задачи интегрирования

Подход, основанный на редукции задач, применим и имеет преимущества по сравнению с подходом, использующим представление в пространстве состояний, когда получающиеся при редукции подзадачи можно решать независимо друг от друга, как в примере с интегрированием. Впрочем, это условие взаимной независимости результирующих задач можно несколько ослабить. Метод редукции применим также, если решения получающих подзадач зависят друг от друга, но при этом существует такой порядок их редукции, при котором найденные решения одних, более ранних подзадач не разрушаются при решении других, более поздних – как в головоломке о ханойской башне, в которой важен лишь порядок решения выделяемых подзадач.

Интеллектуальные роботы

Интеллектуальный робот (его следует отличать от также иногда называемых роботами технических систем, типа систем точечной сварки и т.п.) – программно-аппаратный комплекс, оснащенный *акцепторами* (датчиками о состоянии проблемной среды) и *эффекторами* (средствами воздействия на эту среду, в частности, средствами передвижения), в состав которого входит система ИИ, способная к планированию действий робота в среде.

Часто требуется возможность автономного функционирования робота в проблемной среде (например, в среде агрессивной, в которой человек находиться не может).

Иногда предполагается возможность передачи роботу (человеком-оператором) управляющих команд.

В свое время (70-е гг. XX века) задача создания интеллектуальных роботов рассматривалась как универсальная задача-рамка для исследований в области ИИ. Действительно, помимо таких проблем как представление знаний, планирование решения при создании роботов приходится ставить и решать задачи обработки изображений, управления эффекторами, не возникающие в случае более традиционных систем ИИ (решателей интеллектуальных задач).

В последние годы понятие *интеллектуальный робот* в известной степени вытеснено близким понятием *интеллектуальный агент*.

Интеллектуальные Агенты

Термин ***Интеллектуальный Агент***, ставший популярным лет 5 назад, понимается различными исследователями по-разному.

Общим для различных подходов (точек зрения) является следующая трактовка:

Интеллектуальный Агент – некоторая программно-аппаратная сущность, способная действовать в интересах достижения целей, поставленных перед ним владельцем и/или пользователем (или даже от лица

владельца и/или пользователя).

Обычно предполагается, что *Интеллектуальный Агент*:

- действует автономно или совместно с другими компьютерными/интеллектуальными системами;
- выполняет шаблонные предписанные действия и/или действия, требующие активности и учета состояния окружающей среды (reactivity);
- может генерировать цели и действовать рациональным образом для их достижения (activity / pro-activity);
- в той или иной степени способен к обучению, корпоративным действиям; мобилен.

В работах, посвященных *Интеллектуальным Агентам*, указываются, в частности, следующие сферы их применения (и соответствующие технологии):

агенты, поддерживающие интеллектуальный пользовательский интерфейс;

мульти-агентные системы (и технологии «распределенного искусственного интеллекта»);

мобильные агенты (в том числе реализованные программно), которые могут общаться между собой и перемещаться в своем специфическом окружении (в частности, в компьютерных сетях; они могут покидать клиентский компьютер и перемещаться на удаленный сервер для выполнения некоторых действий, после чего возвращаются обратно).

Проблема знаний - центральная проблема ИИ

Метод представления знаний – совокупность взаимосвязанных средств формального описания знаний и оперирования (манипулирования) этими описаниями.
(аналог ***модели данных*** в теории Баз Данных – понятие концептуального уровня)

Логические методы (язык предикатов)

Знания, необходимые для решения задач и организации взаимодействия с пользователем, – факты (утверждения).

Факт – формула в некоторой логике.

Система знаний – совокупность формул.

База знаний – система знаний в компьютерном представлении.

Основные операции: логический вывод (доказательство теорем)

Примеры:

иметь (Саша, книга)	«Саша имеет книгу»
иметь (Саша, книги) → иметь (Саша, книга)	«Если Саша имеет книги, то он имеет книгу»
(∀x) [человек (x) → иметь (x, книга)]	«Каждый человек имеет книгу»
(∀x) [свободен (x) → ¬(∃y) (на (y,x))]	«Если кубик x свободен, то нет такого кубика y, который находится на кубике x»

Достоинства:

- формальный аппарат вывода (новых фактов/знаний из известных фактов/знаний),
- возможность контроля целостности,
- простая и ясная нотация.

Недостатки:

- знания трудно структурировать,
- при большом количестве формул вывод идет очень долго,
- при большом количестве формул их совокупность трудно обозрима.

Семантические сети

Знания, необходимые для решения задач и организации взаимодействия с пользователем, – объекты/события и связи между ними.

Статические семантические сети - сети с объектами.

Динамические семантические сети (***сценарии***) - сети с событиями.

Система знаний – совокупность сетей (или одна общая сеть).

База знаний – система знаний в компьютерном представлении.

Для представления семантических сетей используются графы:

вершина - атомарный объект (событие),
подграф- структурно сложный объект (событие),
дуга - отношение или действие.

Примеры отношений:

род-вид («компьютер» – «персональный_компьютер»)
целое-часть («компьютер» – «память»)
понятие-пример («компьютер» – «конкретный компьютер . . . »)

Основные операции: сопоставление с образцом, поиск, замена, взятие копии

Пример сети:
<описание компьютера>

Достоинства:
- знания хорошо структурированы, структура понятна человеку.

Недостатки:
- при большом объеме сети очень долго выполняются все операции,
- при большом объеме сети она трудно обозрима.

Фреймы

Знания, необходимые для решения задач и организации взаимодействия с пользователем, – фреймы.

Фрейм-понятие – отношение/действие + связанные этим отношением/участвующие в этом действии объекты.

Фрейм-пример – конкретный экземпляр отношения/действия + конкретные объекты (связанные этим отношением/участвующие в этом действии).

Система знаний – совокупность фреймов-понятий и фреймов-примеров.

База знаний – система знаний в компьютерном представлении.

Фрейм: ИМЯ - отношение/действие
СЛОТЫ- объекты или другие фреймы

С каждым слотом может быть связана такая информация:
УСЛОВИЕ НА ЗАПОЛНЕНИЕ (тип, «по умолчанию», связь с другими слотами)
АССОЦИИРОВАННЫЕ ПРОЦЕДУРЫ (действия, выполняемые, например, при заполнении этого слота)

Основные операции: поиск фрейма/слота, замена значения слота, взятие копии фрейма-понятия

Примеры:
Фрейм-понятие «Перемещать»
ПЕРЕМЕЩАТЬ (кто?, что?, откуда?, куда?, когда?, . . .)
Условия: кто? – человек, робот, . . .
откуда? – место
. . .
Фрейм-пример
ПЕРЕМЕЩАТЬ (Саша, Саша, Главное_Здание_МГУ, Факультет_ВМК, вчера в 15-30, . . .)

Фрейм-понятие «Персональный_компьютер»
ПЕРСОНАЛЬНЫЙ_КОМПЬЮТЕР (процессор?, тактовая_частота?, память?, монитор?, . . .)
Фрейм-пример
ПЕРСОНАЛЬНЫЙ_КОМПЬЮТЕР (Pentium-IV, 5 ГГц, 512Мб, SONY, . . .)

Достоинства:
- знания хорошо структурированы, структура понятна человеку.

Недостатки:
- при большом количестве фреймов долго выполняются все операции,
- при большом количестве фреймов знания трудно обозримы.

Продукции

Знания, необходимые для решения задач и организации взаимодействия с пользователем, – продукции (продукционные правила).

Продукция – правило вида: $p: \alpha \rightarrow \beta$ (где: p – предусловие, α - антецедент, β - консеквент).

Система знаний – система продукционных правил + стратегия выбора правил.

База знаний – система знаний в компьютерном представлении.

Основные операции: вывод (применение правила, определение правила-преемника и т.д.)

Примеры:
True: T > 200°C & P > 5 кПа $\rightarrow$ открыть клапан № 3
True: X - башня $\rightarrow$ X имеет_часть У1 & У1 есть КРЫША & . . .

Достоинства:
- простая и ясная нотация.

Недостатки:
- при большом количестве правил вывод идет очень долго,
- при большом количестве правил их совокупность трудно обозрима.

Терминологические замечания:
1. Из психологии и педагогики нам известна триада: ***знания – умения – навыки***.
Знания – усвоенные Понятия.
Умения – способность выполнять новые действия в новых условиях.
Навыки – действия, автоматизировавшиеся в процессе их усвоения и выполнения.
В работах по ИИ знаниями обычно называют и собственно знания, и умения, и навыки.
Поэтому говорят о: *базах понятий, базах фактов, базах правил* и т.п.
Чтобы не вступать в противоречие с литературными источниками, мы согласимся с такой трактовкой (расширенной) термина **знания**.

2.**Базы знаний (БЗ)** в работах по ИИ часто не совсем корректно противопоставляются ***базам данных*** (утверждается, например, что базы знаний в отличие от баз данных имеют встроенный дедуктивный механизм вывода следствий из известных фактов и т.п.).
Для нас это феномены разноплановых уровней:
База знаний – (у нас) – совокупность «знаний» системы ИИ в компьютерном представлении. Средством представления «знаний» может быть, в частности, та или иная штатная база данных (в обычном смысле).

Остановимся на некоторых острых аспектах проблемы знаний:

Проблема извлечения знаний

(рассмотрим в связи с экспертными системами – ЭС)

Проблема приобретения знаний

(рассмотрим в связи с экспертными системами – ЭС)

Проблема открытости знаний

Совокупность «знаний» системы ИИ неизбежно должна быть открыта для включения в нее новой информации, отражающей динамику проблемной среды и динамику поручаемых системе ИИ заданий.

Открытость может быть реализована по-разному:
- пополнение БЗ «хирургическим путем» (программист/администратор вносят изменения в тексты БЗ),
- обучение системы пользователем в рабочем режиме,
- самообучение системы (приспособление ее к новым условиям/задачам).

В дальнейшем (в этом разделе) мы будем рассматривать задачу создания универсальных ***Адаптивных диалоговых систем ИИ*** широкой ориентации (АДИС).
Это означает, что система:
- может работать с различными пользователями (группами пользователей) – уровень подготовки профессия, и др.;
- может работать в различных *Проблемных областях* – в каждой из которых свои объекты, задачи и др.;
- может настраиваться при этом на *Сеанс* (работа i-го пользователя в j-й Проблемной области);
- может самостоятельно уточнять (если это требуется) условия задач;
- демонстрирует способность к:
  С-адаптации – сиюминутному изменению совокупности «знаний» в новых условиях – и
  С-обучению – запоминанию результатов адаптации для использования в дальнейшем.

Здесь и далее мы будем добавлять префикс «С-» к терминам, характеризующим структурные элементы АДИС и ее функциональные возможности.

Предметная область – «срез» действительности, со своими объектами, отношениями.
Проблемная область – Предметная область + характерные задачи.

Примеры:
Предметная область – Лисп как язык для обработки списков
Проблемные области: автоматический синтез программ на Лиспе,
автоматизированное обучение приемам программирования на Лиспе.

Структуризация С-знаний (по нескольким независимым критериям):
1.
базовые - встроенные в АДИС ее разработчиками
открытые - пополняемые на различных этапах жизненного цикла АДИС
2.
общие - используемые при работе в разных проблемных областях и с разными пользователями,

поблемно-ориентированные - специфичные для конкретных проблемных областей,
личностно-ориентированные - специфичные для различных пользователей
3.
лингвистические - описывающие язык/языки общения с АДИС
предметные - описывающие особенности конкретных предметных/проблемных областей
коммуникативные - описывающие особенности общения в различных сеансах
4.
знания АДИС о ее окружении - описывающие «внешний мир» АДИС
метазнания - знания о С-знаниях

Принцип полноты базовых знаний. Возможность/невозможность «обучения с нуля».

Проблемы полноты и репрезентативности обучающей выборки (при пополнении С-знаний).

Система С-знаний – динамически меняющаяся модель АДИС и ее окружения

Метазнания – средства разрешения конфликта между наличными С-знаниями и входной информацией.
Примеры конфликтов:
- не удается завершить анализ текста условия задачи, т.к. в нем встретилось незнакомое АДИС слово;
- не удается продолжить планирование решения, т.к. ни один оператор к очередной вершине дерева поиска неприменим;
- новый факт формально противоречит одному из ранее известных.

Разрешение конфликта:
- поиск возможных причин (незнакомое слово – это либо действительно новое слово, либо слово с орфографической ошибкой);
- их динамическое (в текущем С-сеансе) упорядочение;
- выбор наилучшего способа устранения конфликта;
- необходимая коррекция С-знаний (С-адаптация) или изменение входных данных (исправление орфографической ошибки);
- С-обучение (факультативно), например, запись в словарь системы нового слова.

Пример обучаемой программы (М.Вайнцвайг)
Внешняя постановка задачи:
Программа получает на вход цепочку символов в некотором алфавите А (вопрос),
Генерирует ответ (цепочка символов над этим алфавитом),
Получает оценку Обучающего (+ - ответ верен, - - ответ неверен).
В процессе обучения программа должна научиться всегда строить правильные ответы (всегда получать оценку +).

Базовые знания программы:
Несколько универсальных эвристик, используемых человеком в случае столь же неопределенных (не имеющих смысловой интерпретации) задач. В числе таких эвристик:
Не выходить за пределы А.
Давать на очередной вопрос ответ, совпадающий с ответом на предыдущий вопрос.
Давать на очередной вопрос ответ, совпадающий с ответом на предыдущий вопрос, если этот ответ был оценен +.
Правильность ответа на некоторый вопрос не зависит от контекста (хода диалога).

В последние годы определенную популярность в работах по ИИ получил подход к моделированию процессов обучения/развития на основе так называемых ***генетических алгоритмов***.

Понятие о генетических алгоритмах

Генетические алгоритмы (ГА) - это стохастические, эвристические оптимизационные методы, впервые предложенные Холландом (1975). Они основываются на идее эволюции с помощью естественного отбора, выдвинутой Дарвином.

ГА работают с совокупностью "**особей**" - популяцией, каждая из которых представляет возможное решение данной проблемы. Каждая особь оценивается мерой ее "**приспособленности**" согласно тому, насколько "хорошо" соответствующее ей решение задачи. В природе это эквивалентно оценке того, насколько эффективен организм при конкуренции за ресурсы. Наиболее приспособленные особи получают возможность "воспроизводить" потомство с помощью "перекрестного скрещивания" с другими особями популяции. Это приводит к появлению новых особей, которые сочетают в себе некоторые характеристики, наследуемые ими от родителей. Наименее приспособленные особи с меньшей вероятностью смогут воспроизвести потомков, так что

те свойства, которыми они обладали, будут постепенно исчезать из популяции в процессе эволюции. Иногда происходят мутации, или спонтанные изменения в генах.

Таким образом, из поколения в поколение, хорошие характеристики распространяются по всей популяции. Скрещивание наиболее приспособленных особей приводит к тому, что исследуются наиболее перспективные участки пространства поиска. В конечном итоге популяция будет сходиться к оптимальному решению задачи. Преимущество ГА состоит в том, что он находит приблизительные оптимальные решения за относительно короткое время.

ГА состоит из следующих компонентов: 1) **Хромосома** (Решение рассматриваемой проблемы. Состоит из генов); 2) **Начальная популяция** хромосом; 3) **Набор операторов** для генерации новых решений из предыдущей популяции; 4) **Целевая функция** для оценки приспособленности (fitness) решений.

Чтобы применять ГА к задаче, сначала выбирается метод кодирования решений в виде строки. Фиксированная длина (l-бит) двоичной кодировки означает, что любая из 2^l возможных бинарных строк представляет возможное решение задачи.

Стандартные операторы для всех типов генетических алгоритмов это: *селекция, скрещивание* и *мутация*.

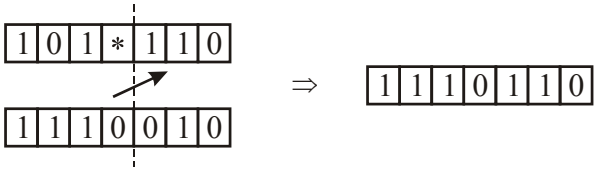
Селекция

Оператор селекции (reproduction, selection) осуществляет отбор хромосом в соответствии со значениями их функции приспособленности. Существуют как минимум два популярных типа оператора селекции: рулетка и турнир.

- **Метод рулетки** (roulette-wheel selection) - отбирает особей с помощью n "запусков" рулетки. Колесо рулетки содержит по одному сектору для каждого члена популяции. Размер i-ого сектора пропорционален некоторой величине вычисляемой по формуле.
  При таком отборе члены популяции с более высокой приспособленностью с большей вероятностью будут чаще выбираться, чем особи с низкой приспособленностью.
- **Турнирный отбор** (tournament selection) реализует n турниров, чтобы выбрать n особей. Каждый турнир построен на выборке k элементов из популяции, и выбора лучшей особи среди них. Наиболее распространен турнирный отбор с $k=2$.

Скрещивание

Оператор скрещивания (crossover) осуществляет обмен частями хромосом между двумя (может быть и больше) хромосомами в популяции. Может быть одноточечным или многоточечным. Одноточечный кроссовер работает следующим образом. Сначала, случайным образом выбирается одна из l - 1 точек разрыва. Точка разрыва - участок между соседними битами в строке. Обе родительские структуры разрываются на два сегмента по этой точке. Затем, соответствующие сегменты различных родителей склеиваются и получаются два генотипа потомков.



Одноточечный оператор скрещивания (точка разрыва равна трем)

Мутация

Мутация (mutation) - стохастическое изменение части хромосом. Каждый ген строки, которая подвергается мутации, с вероятностью P_{mut} (обычно очень маленькой) меняется на другой ген.

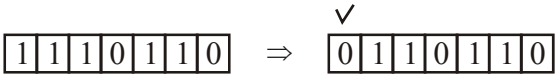
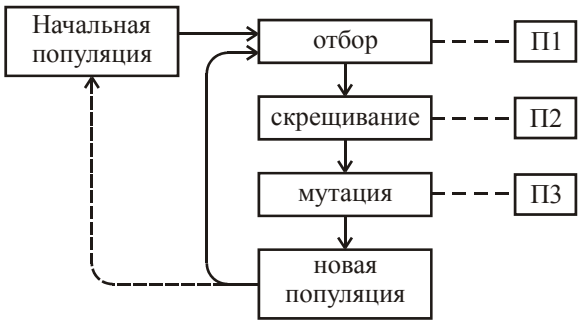


Схема работы ГА

Работа ГА представляет собой итерационный процесс, который продолжается до тех пор, пока не выполнятся заданное число поколений или какой-либо иной критерий останова. На каждом поколении ГА реализуется отбор пропорционально приспособленности, кроссовер и мутация.

Схема работы простого ГА выглядит следующим образом:



Экспертные системы

Работы по созданию **Экспертных систем (ЭС)** - первая попытка практического применения результатов в области **Искусственного интеллекта (ИИ)**.

История: "романтический период" ИИ -> серьезные научные исследования -> практические задачи.

Экспертная система (ЭС) – вычислительная система, в которой представлены знания специалистов в некоторой конкретной узко-специализированной предметной области и которая в рамках этой области способна принимать решения (решать задачи) на уровне эксперта-профессионала.

Основные особенности ЭС:
- ориентированы на решение практических задач в трудноформализуемых узких предметных областях,
- результаты работы сравнимы с результатами человека-эксперта,
- «прозрачность» решения,
- открытая совокупность знаний.

Одна из первых задач - задача ***Диагностики*** (выявление причин неправильного функционирования системы).

Одна из первых ЭС для задачи медицинской диагностики - **MYCIN** (диагностика бактериальных инфекций).

Одна из первых ЭС **в Геологии**: **PROSPECTOR** - оценка потенциальной рудоносности района

Что может делать СИСТЕМА **PROSPECTOR**:
- Беседует с пользователем (экспертом-геологом), употребляя знакомые ему термины.
- Позволяет пользователю вводить любую информацию о районе в любой момент диалога; учитывает при этом степень уверенности пользователя в достоверности сообщаемых им фактов.
- Выбирает гипотезу о рудных залежах в районе и затем задает пользователю вопросы с целью либо доказать, либо опровергнуть ее.
- Перефразирует вопросы, когда пользователь указывает, что они ему непонятны.
- Может сообщить пользователю, почему ей нужен ответ на конкретный вопрос.
- Отбрасывает свои гипотезы и выбирает новые на основании ответов пользователя на свои вопросы.
- Ищет противоречия в ответах пользователя, указывает их и позволяет пользователю изменять и старые, и новые ответы с целью добиться непротиворечивости.
- Предлагает свои заключения вместе со сводкой соображений, на основании которых она пришла к этим заключениям.

ФРАГМЕНТ ПРОТОКОЛА ДИАЛОГА ПОЛЬЗОВАТЕЛЯ С СИСТЕМОЙ **PROSPECTOR**.
01.П: *присутствуют дайки* (запоминает ЭС: дайки, достоверность = 5)
02.П: *присутствуют меловые диориты* (запоминает ЭС: меловые диориты, достоверность = 5)
03.П: *возможно, есть сиенодиорит* (запоминает ЭС: сиенодиорит/монцонит, достоверность = 3)
04.П: *может быть некоторое количество кварцевого монцонита* (запоминает ЭС: кварцевый монцонит, достоверность = 3)

. . .
77:ЭС: на шкале от -5 до +5 уверенность, что имеется месторождение медно-порфировых руд типа А (МПРА), равна 1.683.
Основные заключения в пользу гипотезы:
- Соответствующее геологическое строение региона (достоверность = 3.576).
- Соответствующее петро-тектоническое окружение (достоверность = 3.333).
- Соответствующая интрузивная структура для МПРА (достоверность = 1.683).
Основные наблюдения, поддерживающие гипотезу:
- В исследуемом регионе есть система гранитных интрузий (достоверность = 5.0).
- Наблюдаются измененные породы (достоверность = 5.0)
. . .
Существует еще много других наблюдений в пользу гипотезы. Хотите рассмотреть еще какие-нибудь?
П: *нет*

Типичные задачи, решаемые с помощью ЭС:

Интерпретация - описание ситуации по информации, поступающей от датчиков.
SPE - определение концентрации гамма-глобулина в крови.

Прогноз - определение вероятных последствий заданных ситуаций.
PLANT/cd - определения потерь урожая от черной совки.

Планирование - определение последовательности действий.
TATR - планирование авиаударов по аэродромам противника.

Диагностика - выявление причин неправильного функционирования системы.
MYCIN - диагностика бактериальных инфекций.

Отладка - составление рецептов исправления неправильного функционирования системы.
ONCOCIN - планирования химиотерапевтического лечения.

Ремонт - выполнение последовательности предписанных исправлений.
TQMSTUNE - настройка масс-спектрометра.

Проектирование - построение конфигурации объектов при заданных ограничениях.
XCON (R1) - выбор оптимальной конфигурации аппаратных средств (VAX).

Наблюдение - сравнение результатов наблюдения с ожидаемыми результатами.
VM - наблюдение за состоянием больного в палате интенсивной терапии.

Обучение - диагностика, отладка и ремонт поведения обучаемого.
GUIDON - обучение студентов-медиков (антибактериальная терапия).

Управление - управление поведением системы как целого.
VM

Сферы применения ЭС:

ХИМИЯ: DENDRAL (интерпр.) - определение структурной формулы хим.в-ва

МЕДИЦИНА: VM, MYCIN (см.выше)

ВОЕННОЕ ДЕЛО: TATR (см.выше), I&W (прогнозир.) - прогнозирование вооруженных конфликтов

ЭЛЕКТРОНИКА: EURISKO (проектир.) - проектирование СБИС

КОМПЬЮТЕРНЫЕ СИСТЕМЫ: XCON (см.выше), PTRANS (планир.&прогнозир.) - маркетинг в DEC

ТЕХНИКА: REACTOR (наблюден.) - в составе системы управления ядерным реактором

ГЕОЛОГИЯ: PROSPECTOR (интерпр.) - оценка потенциальной рудоносности района

Основные компоненты ЭС (архитектура ЭС):

- **решатель** / машина вывода (решение задач пользователя),
- **база знаний** (хранение знаний, необходимых для решения задач),
- **подсистема объяснений** (объяснение того, как получено решение),
- **пользовательский интерфейс**,
- **подсистема приобретения знаний**,
- **интерфейс администратора** / инженера знаний.

Решатель ЭС:

Вызов процедур (модулей / правил) по образцу → гибкая схема взаимодействия (управления)
Продукция – правило вида: $p: \alpha \rightarrow \beta$ (где: p – предусловие, α - антецедент, β - консеквент).

Основной цикл работы:

- выборка (правил-кандидатов)
- сопоставление / означивание
- разрешение конфликтов
- выполнение / действия

Метазнания в ЭС:

1. ВЫБОР ПРАВИЛ:

П1: утечка серной кислоты → использовать анион-обменник
(стоимость: дорого, источник информации: доктор Грин, степень опасности: невелика)

П2: утечка серной кислоты → использовать уксусную кислоту
(стоимость: дешево, источник информации: практикант Грун, степень опасности: велика)

П3: прежде всего использовать правило, требующее минимальных затрат

П4: прежде всего использовать правило, внесенное в БЗ специалистом

П5: прежде всего использовать правило с минимальной степенью опасности

2. ОПРАВДАНИЕ ПРАВИЛ:

П6: утечка серной кислоты → использовать известь
(оправдание: нейтрализация, образование нерастворимого и химически неактивного вещества)

П7: утечка уксусной кислоты → использовать известь
(оправдание: нейтрализация)

П8: утечка соляной кислоты → использовать известь
(оправдание: нейтрализация)

3. ОБНАРУЖЕНИЕ ОШИБОК В ПРАВИЛАХ:

ПР01: использовать известь - ***нет антецедента***

ПР02: утечка: соляная кислота → использовать известь

ПР03: соляная кислота → использовать известь - ***проверить: не совпадает ли предусловие с предусловием предыдущего правила***

П9: ***если некоторое правило никогда не срабатывает, проверить его предусловие***

4. СТРАТЕГИЧЕСКИЕ ПРАВИЛА:

П10: пространство поиска относительно мало → оправдан полный перебор

П11: один из конъюнктов часто ложен → перенести его в начало

П12: фрагмент часто выполняется → оптимизировать его

П13: фрагмент часто выполняется & редко меняется → скомпилировать его

П14: утечка вещества, которое не описано в БЗ → база знаний по утечкам неадекватна

Пример вывода в ЭС, основанной на правилах продукций

Правила: R1: разлита горючая жидкость → звонить по телефону 01
R2: разлита уксусная кислота → использовать известь
R3: pH жидкости < 6 → кислота
R4: кислота & имеет запах уксуса → уксусная кислота

Факты: F1: разлита жидкость
F2: pH жидкости < 6
F3: жидкость имеет запах уксуса

Цепочка вывода:

F1 & F2 → F4 (разлита кислота) & F3 → F5 (разлита уксусная кислота) → F6 (нейтрализация)
R3 R4 R2

Цепочка вывода с учетом достоверности / вероятности:

F1	&	F2	→	F4	&	F3	→	F5	→	F6
80%		60%	R3	70%		100%	R4	85%	R2	

Распространение вероятности - изменение вероятности в узлах сети вывода с целью учета влияния новой информации о вероятности в некотором конкретном узле.

Объяснение в ЭС.

Цель - обосновать, аргументировать ответ в максимально естественной форме.

Что объяснять?

- как получено решение
- как использована некоторая информация (факты, правила)
- почему не использована некоторая информация (факты, правила)
- что использовано в целом при решении задачи (факты, правила)

Для кого нужны объяснения?
- эксперты
- инженеры знаний
- пользователи
- изучающие (новички)

Этапы построения ЭС:
- **идентификация ПО** (цели и характеристики ЭС, ресурсы, участники разработки)
- **концептуализация** (основные понятия и связи между ними, основные задачи)
- **формализация** (запись на выбранном языке представления знаний, формирование БЗ)
- **реализация**
- **проверка правил, тестирование**

Стадии разработки ЭС:

прототип: демонстрационный
исследовательский
действующий

система: промышленная
коммерческая

(прототип → система - происходит отчуждение ЭС от разработчика)

Создавать ли ЭС?
ДА - ЕСЛИ: Разработка возможна & Разработка оправдана & Разработка разумна

Разработка возможна:

&
- задача не слишком трудна
- задача вполне понятна
- задача требует только интеллектуальных навыков
- существуют хорошие эксперты
- эксперты единодушны
- эксперты могут описать свои знания

Разработка оправдана:

v
- полученное решение высокорентабельно
- человеческий опыт утрачивается
- экспертов мало
- опыт нужен во многих местах
- опыт нужен в неблагоприятной среде (автономная ЭС)

Разработка разумна:

&
- задача требует оперирования символами
- задача требует эвристических решений
- задача не слишком проста
- задача имеет практический интерес
- задача решаема (ЭС реализуема)

Трудности, Ловушки, Советы

ВЫБОР ЗАДАЧИ: задача слишком трудна, построенная ЭС не решает нужные задачи.

РЕСУРСЫ: количество разработчиков, ЭС - не обычная программа.

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА: неэффективны, не начинать с языков низкого уровня.

ЭКСПЕРТЫ: выбор Э, у Э нет времени, Э далек от компьютеров, Э незнаком с терминологией ЭС, правила слишком коротки и просты, правила слишком громоздки, Э теряет интерес к работе, слишком много Э (они не единодушны во мнениях).

РЕАЛИЗАЦИЯ: разделять знания Э и знания для решения задач, в готовой ЭС нет многих понятий, очень много специфических частных правил, нет хороших объяснений, нет удобств для пользователя, разочарование на этапе тестирования, трудно вносить изменения (большая система).

Извлечение экспертных знаний и формирование БЗ:

Эмпирические правила:

"Чем более компетентен эксперт, тем менее способен он описать те знания, которые использует при решении задач".
"Не будьте своим собственным экспертом".
"Не принимайте на веру все, что говорят эксперты".

Методы извлечения экспертных знаний:

Наблюдение на рабочем месте

Э решает реальные задачи, ИЗ - пассивно наблюдает
цель: ИЗ получает представление о характерных задачах.

Обсуждение задач

ИЗ обсуждает с Э отобранные им (ИЗ) характерные задачи
цель: ИЗ узнает, как организованы знания Э (понятия, гипотезы), как Э работает с неполной, неточной, противоречивой информацией, какие процедуры необходимы для решения задач.

Описание задач

ИЗ просит Э описать типичные задачи для каждого класса задач
цель: ИЗ узнает, как связаны между собой задачи одного класса, классы задач.

Анализ задач

ИЗ предлагает Э задачи и расспрашивает о ходе решения
цель: ИЗ пытается найти и сформулировать стратегии решения задач.

Доводка системы

Э предлагает ИЗ/прототипу_ЭС характерные задачи
цель: ИЗ проверяет сформированную совокупность знаний (БЗ).

Оценивание системы

Э анализирует и оценивает правила, стратегии, систему понятий ПО
цель: Э оценивает точность работы ИЗ и правильность сформированной БЗ.

Проверка системы

ИЗ предлагает независимым экспертам протоколы решения задач Э и прототипом_ЭС
цель: объективная оценка результатов работы ИЗ и Э (и сформированной БЗ).

Пример: Оценка размера страховых выплат. Ситуация.
Беседа Инженера_знаний с Экспертом.
Некоторые правила:
Если *повреждение истца действительно требует, чтобы он носил очки*
и *и истец до повреждения не носил очков*
увеличить фактор неудобства на 2500$
Если *истец действительно имеет вероятность заболеть глаукомой*
и *эта вероятность была вызвана повреждением*
и *величина этой вероятности равна 10%*
увеличить фактор осложнений в будущем на 30000$

Общение человека с системой ИИ

Искусственный интеллект и естественный язык

Как мы знаем, интеллектуальная деятельность предполагает создание и использование абстрактных объектов (понятий), освобожденных от второстепенных, привходящих характеристик и отражающих наиболее существенные стороны действительности. Она реализует высшую форму регуляции деятельности человека в предметной среде.

В свою очередь существенность, значимость отражаемого определяется деятельным контекстом: в процессе управляемой мышлением деятельности и в соотнесении с ее целями и средствами. Очевидно, что деятельность человека нельзя рассматривать в отрыве от его общественных отношений. В социальной среде, в условиях общественного разделения труда структура человеческой деятельности усложняется. Человек может выполнять лишь отдельные этапы решения задачи, а цель деятельности может быть связана с его непосредственными потребностями косвенным путем. При этом функции регуляции совместной и дифференцированной деятельности выполняет ***коммуникативная деятельность*** (***общение***), заключающаяся в обмене информацией о деятельности индивидуальной.

В качестве средства такого обмена (средства общения) используется знаковая система социального уровня - ***язык***. Связи языковых знаков (элементов языка) с психическими явлениями и объектами психического взаимодействия объективированы в социальной группе носителей данного языка и относительно константны для нее. Каждый член группы воспринимает эти связи как данные извне, как существующие объективно, а процесс усвоения этих связей (и знаков языка) - одна из необходимых предпосылок формирования личности, включения человека в систему общественных отношений. Овладевая языком, человек приобретает детально

разработанную систему объективации и реализации собственной психической деятельности, активно используемую им в ходе социальных взаимодействий.

Основными элементами языка как знаковой системы являются ***языковые знаки***, образующие субстанцию языка. Они связаны внутрисистемными отношениями, определяющими как структуру языка в целом, так и структуру конкретных знаковых конструкций, и участвуют в отношениях с внеязыковыми объектами.

Языковой знак - материальный объект, поставленный в соответствие некоторому другому объекту и заменяющий последний в ходе деятельности (свойство ***знаковости***). Другими словами, если некоторый объект является знаком, то он поставлен в соответствие некоторому другому объекту и способен его заменять. Отметим также, что знаки следует отличать от единиц языка: фонем, морфем и слов (в естественном языке). Фонемы свойством знаковости не обладают и служат лишь исходным материалом для построения знаков языка, а морфемы и слова являются знаками.

Второй атрибут языкового знака - его ***конвенциальность***, или немотивированность. Это свойство означает, что устанавливается указанное соответствие соглашением людей, использующих язык (всюду в данном разделе, если специально не оговаривается иное, речь идет о языке человека). Знак может не иметь никакого сходства с объектом, в соответствие которому он поставлен, и не быть связанным с ним причинно-следственными отношениями.

Однако выбор материального объекта на роль знака не может быть произволен абсолютно. Объект должен обладать так называемыми системообразующими свойствами - необходимыми для включения его в систему языка.

К числу системообразующих свойств языка относятся ***дискретность*** ("членораздельность") и ***неоднородность*** ("различаемость") его элементов. Неоднородность языка проявляется и в его ***иерар- хичности*** - в языке может быть выделено несколько иерархических неоднородных уровней, единицы каждого из которых относительно однородны (морфемы - как единицы морфологического уровня, например). С особенностями речевого аппарата человека связан принцип ***линейности*** в языке – в конструкциях, построенных по правилам языка, знаки могут располагаться лишь в линейной последовательности, то есть цепочкой.

Как элементы системы (языка) знаки обладают еще рядом свойств. Связь знака с внеязыковыми объектами задается отношениями: ***сигматическими*** - связь знака с реальными объектами и явлениями действительности или отдельными аспектами их; ***семантическими*** - связь с психическими моделями соответствующих сторон реальности или с моделями реально не существующих объектов;
прагматическими - связь с людьми, использующими знаки языка в своей деятельности. **Синтаксические** отношения (при описании естественных языков обычно рассматриваются два уровня синтаксических отношений: морфологический и синтаксический в узком смысле) характеризуют связи между знаками как элементами языка: иерархические - отношения вхождения знака в сложный знак; синтагматические - отношения взаимодействия знаков или их классов; парадигматические - отношения между элементами одного класса, например, формами одного слова.

Характеристика языка как знаковой системы, элементы которой связаны отношениями с различными сторонами действительности, полезна и при рассмотрении процессов его использования (традиционная начиная с работ Ф. де Соссюра проблема лингвистики "язык - речь"). Язык является знаковой системой социального уровня. Выбор знаков, правил их комбинирования и соотнесения с явлениями объективного и субъективного планов закреплен общественным соглашением, которое разделяют все владеющие данным языком. Тем самым язык задает нормы интерпретации и употребления знаков. А в реальных процессах его использования языка на основе этих общих норм строятся и анализируются конкретные знаковые конструкции, описывающие конкретные ситуации.

Таким образом, процесс использования языка - ***речевая деятельность*** (РД) - может быть охарактеризован как актуализация и конкретизация существующих в языке в потенциальной форме синтаксических отношений и отношений знаков с внеязыковыми объектами: выбор нужной формы слова при синтезе, установление актуального сигматического отношения с реальным объектом - "денотатом" и т.п. Результатом РД при синтезе является **сообщение**, несущее определенную информацию и о самом языке (по правилам которого оно построено), и о ситуации, в которой оно было синтезировано, и об авторе.

В структурном плане сообщение представляет собой наделенную структурой линейную последовательность знаков. Каждый знак характеризуется его позицией в цепочке и актуальными синтаксическими отношениями с другими знаками. "Нечленимые" в языке знаки - материальные оболочки "элементарных" языковых единиц (морфем, неизменяемых слов, фразеологизмов) - "воспроизводятся" в речевом произведении в готовом виде. Структурно сложные знаки строятся из составляющих по правилам языка.

Общение, или коммуникативная деятельность, предполагает наличие не менее двух носителей языка - ***автора*** сообщения и субъекта, которому это сообщение адресовано, - ***реципиента*** (в случае диалога они естественным образом поочередно "меняются местами"), а также наличие некоторой совместной деятельности, которую общение должно регулировать.

Определенное сходство в структурном и функциональном плане с естественными языками имеют и представители класса искусственных языков.

Хотя проблема соотношения естественных и искусственных языков представляется на первый взгляд (и является на самом деле) тривиальной, при ее рассмотрении часто допускаются ошибки. Одна из наиболее распространенных - абсолютизация временных и несущественных различий (существенные при этом как правило упускаются из виду), учет особенностей искусственных языков, существующих в данный момент, и

неоправданный перенос выявляемых при этом различий на весь класс искусственных языков вообще. Очевидно, что в этом случае и сама проблема подменяется другой.

Единственное непреходящее отличие естественного языка от всех прочих - которые и следует называть искусственными - связано с историей его возникновения. ***Естественный язык*** - продукт, естественно-исторически возникший из объективных общественных потребностей (в первую очередь из потребности в общении, регулирующем совместную и дифференцированную деятельность) на ранних этапах общественного развития, когда человеческое познание было практически нерефлексивным и ни о каком активном сознательном регулировании процесса создания языка не могло быть и речи.

Объективность существования ЕЯ объясняет, почему многие языковые реликты, исключения и нерегулярности "живут" в языке до сих пор, а попытки избавиться от них, модернизировать, улучшить язык часто заканчиваются неудачей. Для того, чтобы некоторые новшества попали в язык, "закрепились" в нем, их должно усвоить большинство носителей языка (и при спонтанном развитии языка новшества, возникшие в речи, попадают в язык лишь в этом случае). Это и есть объективация. В противном же случае (без объективации) модернизатор, пусть даже руководствующийся самыми благими намерениями, окажется в положении Шалтая-Болтая из "Алисы в Зазеркалье".

Примечательно, что хотя существенные отклонения от общих языковых правил и норм недопустимы, индивидуальные модели ЕЯ (***индивидуальные языковые модели*** - ИЯМ), усвоенные его носителями и определяющие особенности речи последних, могут иметь некоторое своеобразие. Они описывают своего рода "подмножество-расширение" эталонного языка, не содержащее неизвестных конкретному носителю языка знаков и правил и включающее ряд конструкций и форм, не входящих в общелитературный язык (заведомо неграмматичных) или используемых в рамках узких социальных групп (формы просторечные и разговорные, жаргонизмы, профессионализмы). Несмотря на формальную недопустимость или ограничения на употребление подобных форм, они (эти формы) не только встречаются в повседневной речи (соответствующие запреты либо вообще не входят в ИЯМ носителя языка, либо случайно или намеренно им игнорируются), но и понимаются людьми.

Понятие ИЯМ подчеркивает факт субъективного преломления естественного языка как знаковой системы социального уровня в ходе его усвоения и использования. Формирование ИЯМ - сложный и длительный процесс, включенный в процесс развития человеческой психики и детерминируемый как внешними (социальная среда), так и внутренними (индивидуальные психофизиологические особенности) факторами. Примечательно, что с уменьшением социальной группы (производственный коллектив, семья) своеобразие используемого в ней подмножества-расширения ЕЯ возрастает. И это естественно: усвоить отклонения от языка-эталона должно в этом случае небольшое число лиц.

Характерной, но не обязательно отличительной чертой любого естественного языка является его ***универсальность***. Возникнув как средство регуляции самых разнообразных видов человеческой деятельности (протекающей в различных контекстах, предполагающей использование различных средств, направленной на различные объекты), ЕЯ может быть использован для выражения качественно различных видов содержания, для описания реальной действительности и процессов психической деятельности с разной степенью строгости, полноты, эксплицитности.

Автор сообщения, например, может описать некоторую ситуацию в весьма общих чертах и, не потеряв при этом объективно существенные (но не сочтенные им таковыми) детали, возложить задачу выявления этих деталей и дальнейшей конкретизации совместной деятельности на реципиента.

И хотя в ходе последующих языковых взаимодействий могут потребоваться новые языковые средства (необходимые для выражения новых видов информации), соответствующее расширение ЕЯ, представляющего собой открытую систему, всегда возможно. При этом способность ЕЯ служить метаязыком для себя позволяет описать такое расширение средствами самого языка.

Поскольку ЕЯ является средством объективации психической деятельности человека (построения моделей отражаемого) и средством передачи информации об отражаемом, сигматические, семантические и прагматические аспекты знаков чрезвычайно существенны в процессах использования языка. Важную роль играют и синтаксические отношения, существующие в потенциальной форме в языке или в ИЯМ и актуализируемые в РД. Возможность передачи информации с необходимостью предполагает структурированность языковых выражений, репрезентующих эту информацию, а многие тонкие оттенки содержания передаются исключительно синтаксическими средствами. Так, порядок слов, например, находится в соответствии с "актуальным членением" соотнесенного с текстом содержания.

Необходимо учитывать и различие значения в языке и речевой деятельности, а также динамическую природу значения - с течением времени со знаком начинают ассоциироваться новые объекты (расширение значения), а часть прежнего значения может теряться (сужение), например, за счет появления уточняющих терминологических словосочетаний.

Синтаксическое значение существует в языке как система ассоциированных с данным сообщением эталонных парадигматических, синтагматических и иерархических связей с другими знаками языка - аналог "значимости" в смысле одного из лингвистов-классиков Ф. де Соссюра. Являясь соотносительной характеристикой знаков языка, оно определяет выбор некоторого знака как средства выражения других видов значения. ***Сигматическое значение*** - класс реальных объектов ("денотатов", или "обозначаемых"), в соответствие которым может быть поставлено сообщение, в то время как ***семантическое значение*** отсылает к классу эталонных психических моделей денотатов (к "десигнатам", "означаемым", или "концептам"). ***Прагматическое значение*** представляет собой класс нормативно соотнесенных с сообщением действий

потенциальных реципиентов или же класс действий и целей потенциального автора сообщения, побуждающих его к речевой деятельности.

Полное ***значение сообщения*** является комплексом четырех указанных видов значения. Возможны и "вырожденные" случаи. Так, например, сигматическое значение некоторых знаков может быть пустым (всегда - *кентавр*, в определенном временном интервале - *Великий инквизитор*); пустая морфа не изображается никаким знаком (хотя, например, в словоформе *слов* отсутствие флексии передает грамматическое значение родительный падеж, множественное число); служебные морфемы обладают лишь грамматическим и крайне абстрактным семантическим (грамматический род, число) значением.

Кроме того, для естественного языка чрезвычайно характерно отсутствие взаимно-однозначного соответствия между знаками и связанными с ними обозначемыми, означаемыми и действиями.

Знак или сообщение называется ***омонимичным***, если связанные с ним в языке или в речевом произведении классы обозначаемых, означаемых или деятельных актов содержат более одного элемента. Частным случаем омонимии является ***полисемия*** - наличие у слова нескольких обозначаемых (означаемых): *оросительный канал - канал ствола орудия - канал связи с ЭВМ - дипломатические каналы*. Возможна и грамматическая омонимия: *второй параграф - о второй главе - для второй главы*, *цинковые белила - белила потолок*, *прием посла = посол принял* <кого-то> - <кто-то> *принял посла*.

В письменной речи (если не проставлены ударения) неразличимы формы: *профессора - профессора, колет - колет, это все твои вещи*? - *это все твои вещи*?, а в устной (***омофония***) - конструкции *и скота - из кота.* Определенный интерес представляют сходные по строению, но имеющие несовпадающие значения слова (***паронимы***): *языковый - языковой, представить - предоставить* (*языковая колбаса - языковой знак*, *представить справку - предоставить возможность*).

Если же различные знаки имеют "общее значение" и могут, в частности, заменять друг друга в сообщении (общее в синтаксическом значении, "значимости"), то их обычно называют ***синонимами***. "Абсолютная" синонимия в ЕЯ встречается редко: *языкознание = лингвистика = языыковедение*; флексии *-ее* и *-ей* с грамматическим значением сравнительная степень прилагательного, находящиеся в отношении свободного чередования: *весел-ее = весел-ей*.

Как правило, возможность замены знака другим, то есть употребления синонима, ограничена нетождественностью "общего значения", контекстными условиями. Так, например, морфологические синонимы - флексии с "общим значением" мужской род, множественное число, именительный падеж: *-ы, -и, -е, -а, -я* - употребляются с существительными различных типов склонения (*лингвист-ы, филолог-и, южан-е, профессор-а, кра-я*). К контекстным условиям относятся также лексическая сочетаемость и стилистическая окраска. Например, замена *хочу* на *желаю* во фразе *Я не хочу его видеть* допустима, а во фразе *Кроме нас хотят переселиться в колхоз еще несколько семей* – нет.

Очевидно, что для экспликации понятия синонимии необходимо, вообще говоря, построить некоторую метрику в пространстве значений (разумеется, и описать это пространство), а также выбрать ту или иную трактовку синонимии. Например, можно считать, что два объекта являются "абсолютными синонимами", если расстояние между ними в построенной метрике равно нулю (сильная трактовка), "синонимами" (или "полусинонимами"), если оно меньше единицы и т.д.

Описывая одну и ту же ситуацию, одно обозначаемое с помощью различных синонимичных конструкций (синонимичными, естественно, могут быть и предложения, тексты), носитель языка имеет возможность переструктурировать не только сообщение, но и репрезентуемую им модель ситуации, выделять то одни, то другие компоненты и аспекты.

Согласно данному выше определению значение сообщения является комплексом эталонных для некоторого ЕЯ внутриязыковых связей знаков и ассоциированных со знаками внеязыковых объектов. А поскольку при усвоении естественного языка конкретным человеком, при формировании ИЯМ язык претерпевает субъективное преломление, отклонения от эталона имеют место и в сфере значений - субъективное сужение или расширение значения. Возможно, что отдельные виды значения при этом не меняются или меняются в разной степени. Своеобразием может отличаться и значение, ассоциированное со знаком в той или иной социальной группе (научные термины, профессионализмы, диалекты языка, жаргонизмы).

Учет субъективной и социальной соотнесенности значения (как следствие преломления языка в ИЯМ или в модели языка, общей для членов социальной группы) - необходимая предпосылка эффективного использования ЕЯ для общения между людьми, между людьми и искусственными информационными системами.

В конкретных процессах РД происходит дальнейшая модификация значения - с сообщением связываются лишь отдельные компоненты преломленного в ИЯМ значения. Подобная актуализация предполагает выбор уместных в текущем контексте аспектов семантического и прагматического (учет конкретной цели автора сообщения, особенностей собеседника) значений; установление, если это возможно, связи с обозначаемым; выбор (раскрытие) синтаксических средств выражения значения.

Соотнесенная с сообщением в реальном процессе речевой деятельности подсистема значения (виртуально ассоциированного с данным сообщением в ИЯМ носителя языка) может быть названа ***смыслом сообщения***.

Понимание выражений естественного языка

С привлечением понятий значения и смысла сообщения можно конкретизировать описание информационного аспекта общения, регулирующего совместную и дифференцированную деятельность. Автор

очередного сообщения строит его таким образом и с использованием таких (представленных в его ИЯМ) языковых средств, чтобы смысл сообщения максимально точно отображал важнейшие в текущей контекстной ситуации аспекты деятельности, преследуемые им цели. Задача же реципиента - выявить этот смысл, то есть установить те стороны значения (допустимого с позиций его ИЯМ), которые наиболее существенны в текущей ситуации с его точки зрения и которые, как он предполагает, имел в виду автор сообщения.

Подобный процесс раскрытия смысла и назван ***пониманием сообщения***.

Понимание сообщения можно трактовать как вид интеллектуальной деятельности, обладающий всеми атрибутами ее и, естественно, всеми атрибутами психического отражения: информационностью (выявляется информация об актуальных связях знаков) и субъективностью (учитывается информация, существенная с позиций реципиента). Процесс понимания может быть прерывистым: выдвижение подцелей, возврат на предыдущие этапы и коррекция. Кроме того, он может сопровождаться адаптацией как к языку в целом, так и к особенностям ИЯМ собеседника.

В силу того, что понимание языковых выражений - один из наиболее привычных для человека видов интеллектуальной деятельности, многие этапы понимания реализуются с помощью неосознаваемых человеком средств - вторичных автоматизмов, сформированных в ходе усвоения языка.

Однако хотя построение интерпретации сообщения часто осуществляется "автоматически", и не вызывает осознаваемых субъективных трудностей, смысл, соотнесенный с сообщением реципиентом
- Ср, может отличаться от смысла, ассоциированного с тем же сообщением его автором - Са. Главные причины этого: несовпадение ИЯМ, используемых конкретными носителями языка; неадекватное отражение в них языка как социального феномена; многоплановость сущности, репрезентуемой языковым выражением (негомогенность значения, потенциальная неисчерпаемость этого значения).

После того как Са объективирован в построенном автором сообщении и сообщение передано реципиенту, последний рассматривает как единственно доступные реальности только это сообщение (линейную последовательность знаков языка) и текущий деятельный контекст. Он может воспринимать этот контекст иначе, чем автор сообщения - из-за иных представлений о членении ситуации и существенности отдельных компонентов ее - и сопоставить сообщению иную интерпретацию, отображающую те аспекты, которые восприняты им. Даже и в том случае, когда основные компоненты ситуации отображены реципиентом верно (относительно Са), многогранность свойств и характеристик реальных объектов может привести к расхождению смыслов, ассоциируемых с соответствующим сообщением.

Кроме того "ключевыми" в текущем коммуникативном акте могут быть различные виды значения сообщения. Одно и то же сообщение может быть синтезировано с разными целями: установить актуальные семантические и сигматические связи и выполнить действия над соответствующими объектами, запомнить смысл сообщения, запомнить лишь знаковую сторону сообщения.

Замечательный пример, иллюстриующий возможность неправильного относительно Са выделения ключевого вида значения описан Дж.Литлвудом. В печатном тексте одной из своих работ Литлвуд не обнаружил последней фразы - "Таким образом, σ следует сделать сколь возможно малым". Однако вместо нее на пустом месте в конце статьи стояла миниатюрнейшая σ. Наборщик вместо того, чтобы воспроизвести данную фразу (что он и должен был сделать по роду своей профессии), выполнил потенциально предусматриваемые ее значением действия.

Примечательно, что своеобразие восприятия смысла часто определяется устойчивыми психическими факторами, в первую очередь социальной ролью реципиента (поэтому ошибку наборщика следует признать экстраординарной). Так, лингвиста могут заинтересовать несущественные для прочих носителей языка проявления в речи языковых закономерностей, а математика - можно вспомнить дона Веласкеса из романа Я. Потоцкого "Рукопись, найденная в Сарагоссе" (и одноименного фильма) - количественные отношения между описываемыми в сообщении объектами.

В силу действия подобных осложняющих факторов понимание должно предполагать ориентацию на ИЯМ автора полученного сообщения, учет целей его обращения к реципиенту, а также адаптацию и обучение, направленные на сближение используемых собеседниками ИЯМ и повышение точности отражения в них языка. Основная предпосылка близости ИЯМ - активные коммуникативные взаимодействия, протекающие в общем деятельном контексте (так, в небольшой социальной группе ИЯМ достаточно близки, хотя они, как отмечалось, могут отражать язык с искажениями). А для сближения смыслов можно использовать перефразирование сообщений, упоминавшееся в связи с синонимией, и уточняющий диалог.

Разумеется, такой диалог будет возможен и плодотворен лишь в том случае, когда реципиент осознает, что он понял сообщение "как-то не так", что в сообщении не указаны важные с его точки зрения детали - "*Этого просто не вынести*! *А что вам нужно вынести*? - *спросила Алиса*" (Л.Кэрролл, "Алиса в Стране Чудес"), что в его распоряжении вообще нет средств приписать сообщению какую-либо интерпретацию.

В тех же случаях, когда реципиент считает, что смысл сообщения раскрыт им правильно, возможно объективное несовпадение этого смысла с Са и/или с наиболее вероятной в языке интерпретацией сообщения - нормативно выделенной подсистемой значения ЯВ, которую можно назвать смыслом относительно языка (Ся). В языке критерии выделения Ся должны быть объективными, например синтаксическими. Так, наиболее вероятная (нормативная) в русском языке трактовка фразы *Мать любит дочь* определяется порядком слов: *мать* - субъект, *дочь* - объект. Можно считать, что значение данного выражения включает и другую, менее вероятную интерпретацию (которую следовало бы выразить фразой *Дочь любит мать*) и предположить, что эта интерпретация соответствует Са. Ср может в данном случае совпасть либо с Ся, либо с Са.

Таким образом, Са - "то, что хотел сказать автор сообщения", Ся - "то, что сказано", и Ср - "то, что понял реципиент", вообще говоря могут не совпадать. Если Ср совпадает с Са, реципиент понял сообщение ***правильно относительно автора***, если же Ср совпадает с Ся - ***объективно правильно***. ***Субъективно правильное понимание*** имеет место в том случае, когда Ср релевантен текущей деятельности реципиента, когда реципиент сумел извлечь из полученного сообщения ценную для себя информацию.

Введенные критерии правильности независимы - понимание правильное в одном аспекте, может быть правильным и в других, а может и не быть таковым. Причем правильность трактуется как соотносительная характеристика процесса понимания. Понимание может быть правильным (или неправильным) лишь по отношению к некоторому "судие": автору сообщения, языку, деятельности реципиента. В то же время при совпадении (близости) Са, Ся и Ср можно говорить и об абсолютно правильном понимании. Здесь, правда, возникает уже упоминавшаяся в связи с синонимией проблема описания "пространства смыслов" - причем, общего для собеседников (!) - и задания его "метрики". В пространстве смыслов, наряду с правильностью понимания, характеризующей корреляцию наиболее существенных аспектов значения, можно было бы рассматривать и ***полноту понимания*** - меру близости объемов смыслов.

Для того, чтобы добиться правильного понимания адресуемых ему сообщений, каждый из участников процесса общения должен располагать информацией об определяющей предмет общения проблемной среде, о языке (эта информация представлена в его ИЯМ), о собеседнике, в том числе и об используемой им ИЯМ, и о себе. Эта информация соответствует ***глобальному контексту*** общения.

Естественно, что при обработке очередного сообщения (отдельной фразы, абзаца и т.п.) важную роль играет и информация, почерпнутая из предшествующих сообщений (из ***локального контекста***). Именно учет глобального и локального контекстов: предмета обсуждения, собственных целей и целей собеседника, языковых и внеязыковых связей между отдельными сообщениями - и помогает реципиенту приписать очередному сообщению наиболее уместную интерпретацию, то есть правильно понять его.

Установив, о чем идет речь в сообщении, как должна быть использована содержащаяся в нем информация, реципиент может относительно легко разрешать проблемы, возникающие при анализе чисто знаковых (синтаксических) отношений, определяющих структуру сообщения.

Иллюстрирует эти возможности способность человека:

1) выбирать "наиболее разумную" интерпретацию сообщения, отсеивая интерпретации неестественные (но формально допустимые): *За безбилетный проезд и провоз одного места багажа взимается штраф 1 рубль, Сведения о войсках противника, которые помогали нашим партизанам, В черных костюмах выступают наши фигуристы, которые отделаны красными и зелеными цветами;*
2) понимать неграмматичные (ошибочные) конструкции: *Ошибка вслове лектор, В аудиторию вошли лектора* [следует: *лекторы*], *Предоставить* [следует: *представить*] *справку в бухгалтерию* - и грамматически неоформленные квазифразы типа: *ребен- спа- комнат- шир- распах- окн-*;
3) определять по контексту достаточные с точки зрения текущего этапа общения аспекты значений и функциональные роли в тексте незнакомых слов и конструкций. Читатель "Алисы в Зазеркалье", например, достаточно ясно представляет себе, что произошло с головой Бармаглота (... *Взы-взы - стрижает меч, Ува! Ува! И голова Барабардает с плеч*!), хотя и не знает семантическое и сигматическое значения незнакомого глагола *барабардать*.

Примечательно, что ориентация на "высшие" аспекты значения (сигматический, семантический и прагматический), то есть на внеязыковые связи знака характерна и для более частных видов речевой деятельности. Так, согласно данным психолингвистики и при выборе слов из долговременной памяти человек ориентируется в первую очередь на их семантические значения и связи. Использование других критериев, звукового сходства, например, свидетельствует либо о невозможности обращения к семантическому уровню (незнание семантического значения слова), либо о нарушении психической деятельности (шизофрения).

В этой связи можно вспомнить знаменитую фразу Л.В.Щербы *Глокая куздра штеко будланула бокра и кудрячит бокренка.* Невозможность установить сигматические и семантические отношения квазислов этой фразы заставляет человека при ее анализе обратиться к чисто знаковым (синаксическим) отношениям. Предполагая грамматическую корректность фразы, можно исследовать ее синтаксические свойства: порядок слов, словоизменение, словообразование (*бокр - бокр-енок*). Определенные ассоциации могут возникнуть и при анализе знаковой (звуковой) структуры корневых морфем. Так, *глокость* может показаться кому-то очень нехорошим качеством, а глагол *кудрячить* может ассоциироваться либо с существительным *кудри*, либо с глаголами *корчить* или *корячить*, либо с названием встроенной функции CDR (рекомендуется произносить "кудр") языка Лисп.

Несомненно, "высшие" аспекты значения передаются с помощью знаковых (синтаксических) средств, а проникнуть на эти "высшие" уровни не удается, не начав анализа структуры сообщения. Однако можно предположить, что по мере раскрытия внеязыковых связей знаков - даже до завершения анализа синтаксической структуры сообщения в целом - происходит переход на уровень информационной модели описываемой ситуации. Причем выявляемая информация (семантическая, сигматическая, прагматическая) не только пополняет эту модель, но и управляет дальнейшим анализом текста.

Проблема речевых ошибок

Использование естественного языка в качестве средства общения (***речевая деятельность*** человека) неизбежно сопровождается теми или иными нарушениями языковых правил. Такие нарушения - вне зависимости

от того, обусловлены они неполнотой знаний человека о языке или же случайными сенсомоторными "сбоями" (описки, опечатки, оговорки) - мы будем называть ***речевыми ошибками***.

Обнаружить речевую ошибку не всегда просто. Действительно, для получателя сообщения (реципиента) внешним признаком речевой ошибки служит появление в тексте какой-либо незнакомой ему речевой единицы. Однако такая "подозреваемая" речевая единица может оказаться и правильной конструкцией или формой (например, просторечным вариантом или термином), не знакомой реципиенту.

С другой стороны, абсолютно правильная на первый взгляд единица может быть ошибкой, обнаружить которую удается лишь на "высших" этапах анализа. Так, в предложении "*Пуск ракеты осуществляется нажатием краской кнопки"* все слова известны, синтаксические связи правильны; опечатка обнаруживается только на семантическом/смысловом уровне.

Если одним из участников общения является компьютерная система (система автоматической обработки текста – ***АОТ-система***), положение становится еще более сложным. И лингвистические знания, и интеллектуальные способности (в том числе - в плане работы с языком) такого "собеседника" пока весьма скромны.

Отметим еще одно обстоятельство. Как бы ни разнились характер использования и назначение АОТ-систем (системы машинного перевода, работающие в пакетном режиме; системы обеспечения диалога с машиной на естественном языке), оснащение их средствами обнаружения и исправления речевых ошибок повышает устойчивость и эффективность функционирования таких систем, облегчает (в случае диалоговых систем) процесс общения человека с ЭВМ.

Классификация речевых ошибок

Первый критерий классификации речевых ошибок, в соответствии с которым ошибки подразделяются на мотивированные и случайные, связан с понятием индивидуальной языковой модели. ***Индивидуальная языковая модель*** (ИЯМ) - это то подмножество языковых единиц и правил, которое усвоил и использует в своей речевой практике конкретный носитель некоторого естественного языка. Субъективное преломление языка (как знаковой системы социального уровня) в процессе его усвоения приводит к тому, что в ИЯМ не попадают (или попадают в искаженном варианте) некоторые языковые единицы и правила языка.

Поэтому в речи конкретных носителей языка начинают проявляться некоторые индивидуальные особенности, либо вступающие в противоречие с языковыми нормами, либо нет.

В первом случае мы имеем дело с мотивированными речевыми ошибками - точнее, с ошибками, мотивированными особенностями ИЯМ конкретного носителя языка (автора анализируемого АОТ-системой текста). К ошибкам такого рода относятся, например, ошибки в словоизменении (*контейнер**а́*** - в форме именительного падежа множественного числа), орфографические ошибки в основах (*ед**е**ница*), некоторые пунктуационные ошибки, смешение слов-паронимов (*представить* - *предоставить*), нарушение лексической сочетаемости (*делать горе*), искажение фразеологизмов (*не так страшен черт, как его малютки*).

Ошибки, обусловленные внешними по отношению к ИЯМ факторами: сбой речевого аппарата человека, несвоевременное переключение регистра клавиатуры, нажатие соседней клавиши, сбой на линии связи с ЭВМ - мы будем называть случайными. Как правило, мотивированные речевые ошибки регулярно повторяются в речи носителя языка, а случайные ошибки могут как повторяться (например, при западании клавиши), так и не повторяться. Отметим, что иногда отличить случайную ошибку от мотивированной сложно. Так, употребление слова *представить* вместо *предоставить* в контексте *представлено право* может быть или результатом случайной ошибки (пропуск буквы), или результатом мотивированной ошибки (смешения паронимов).

Мотивированные речевые ошибки могут различаться степенью серьезности (грамматичности). Помимо серьезных, абсолютно недопустимых грамматических ошибок - типа орфографических ошибок в основах или смешения слов - рассматриваются и ошибки, в результате которых появляются "полуграмматичные" формы (*контейнера́*, *сидевши*), которые имеют в словарях стилистические пометы: просторечное, устарелое, разговорное, областное и др.

Следующий критерий классификации ошибок (мотивированных и случайных) связан с языковыми уровнями, нормы (правила) которых оказываются нарушенными в результате речевых ошибок. В соответствии с этим критерием речевые ошибки можно классифицировать следующим образом:

1) орфографические ошибки: пропуск одной буквы, замена одной буквы, перестановка двух рядом стоящих букв, одна лишняя буква (отдельно может рассматриваться случай удвоения буквы), замена буквы русского алфавита буквой латиницы и др.;
2) морфологические (словоизменительный уровень) ошибки: ошибки в окончаниях (флексиях) при склонении и спряжении слов (рассматриваются различные подклассы таких ошибок), употребление отсутствующих в языке форм слов, несоблюдение правил чередования в основе, употребление незнакомых АОТ-системе вариантов слов, испытывающих колебания в роде, одушевленности;
3) синтаксические ошибки: ошибки в моделях управления слов-предикатов, пунктуационные ошибки, нарушение нормативного порядка слов (в том числе - в устойчивых словосочетаниях), вставка пробела внутрь слова, пропуск пробела (отдельно могут рассматриваются случаи слитного и раздельного написания частиц *не* и *ни*);
4) лексико-семантические ошибки: употребление слов в ненормативном значении, нарушение лексической сочетаемости, семантические противоречия.

Диагностика речевых ошибок

Методы обнаружения и исправления орфографических и морфологических ошибок в текстах широкой тематики базируются на представлении о тексте как о цепочке независимо появляющихся словоформ. Известно три основных метода обнаружения орфографических ошибок - статистический, полиграммный и словарный.

При статистическом методе словоформы, обнаруживаемые в тексте, упорядочиваются согласно частоте их встречаемости. Искаженные слова оказываются среди малоупотребительных слов в конце списка. При полиграммном методе все встречающиеся в тексте двух- или трёхбуквенные сочетания (полиграммы) проверяются по таблицам, содержащим информацию об их допустимости в русском языке. Если в словоформе имеются недопустимые полиграммы, то она считается неправильной. При словарном методе все входящие в текст словоформы проверяются по компьютерному словарю. Если словарь такую форму допускает, она считается правильной, а иначе либо сразу признаётся ошибочной, либо предъявляется человеку.

В настоящее время первые два метода практически не используются, т.к. уже есть хорошие компьютерные словари, достаточно большие по объёму и с эффективным доступом.

Диагностика же и исправление синтаксических, пунктуационных и лексико-семантических ошибок предполагает взгляд на текст как на последовательность связанных единиц, комбинирование которых имеет свои закономерности. Подходы к автоматизации выявления и коррекции этих ошибок можно разбить на две группы: синтаксически-ориентированные подходы и подходы, основанные на концептуальных фреймах. Последние больше пригодны для систем, работающих в строго ограниченных предметных областях. Для текстов широкой тематики предназначены синтаксически ориентированные подходы. Сначала поступившее на вход предложение обрабатывается средствами грамматики, рассчитанной на синтаксически правильный текст. Если такая проверка обнаруживает дефекты синтаксической структуры, некоторые условия ослабляются. Какие грамматические правила смягчаются, зависит от учитываемых системой ошибок. Например, в русских текстах иногда оказывается пропущенной запятая, обособляющая причастный оборот в постпозиции. Для того, чтобы такое предложение могло быть обработано, требуется временная отмена условия (присутствующего в каноническом правиле) обязательного наличия запятой. Однако ослабление канонических правил неизбежно влечёт за собой возрастание числа возможных интерпретаций. При этом нельзя опознать ошибочный текст прежде, чем будет закончен анализ средствами канонической грамматики. Другой подход предлагает сначала использовать слабую грамматику, а затем подвергнуть обрабатываемое предложение фильтрации на основе строгих требований правильности. Но при этом наличие ошибки предполагается более вероятным, чем соблюдение норм грамматики.

Также отметим, что описанные методы позволяют автоматически обнаружить ошибку только тогда, когда не удаётся построить связный синтаксический граф для рассматриваемого предложения. Однако ошибки, при которых возможно получение формально приемлемой, но по сути неверной интерпретации, остаются невыявленными. При этом никаких сообщений об ошибках не поступает.

Система комплексного контроля качества текста ЛИНАР

Построение автокорректоров сталкивается с рядом принципиальных и не решенных пока в полном объеме проблем: компактное хранение словарей, эффективные методы морфологического и синтаксического анализа и т.д. Тем не менее на очереди - создание систем, способных производить более сложное по сравнению с автокорректорами автоматическое или автоматизированное редактирование текстов на естественном языке. В идеале же необходима система, выполняющая функции научного редактора - человека, осуществляющего литературную и научную правку научно-технических текстов. Такое направление развития представляет разрабатывавшаяся в 1986-1990 гг. на кафедре алгоритмических языков факультета ВМК МГУ система ЛИНАР (ЛИтературно-НАучный Редактор) - интеллектуальная система комплексного контроля качества и редактирования русскоязычных текстов.

Суть подхода заключалась в существенном расширении возможностей имевшихся в то время автокорректоров за счет:

- ограничения предметной области, к которой относились обрабатываемые тексты (методы, алгоритмы и программы обработки данных телеметрии на многопроцессорных вычислительных комплексах);
- ограничения видов текстов (научно-технические отчеты, деловая переписка);
- использования средств синтаксического и семантического анализа текста;
- привлечения более полных моделей русского языка.

Пользователем ЛИНАР является человек, оценивающий с помощью системы качество некоторого текста с позиций лица, которому адресован этот текст (адресата), и вносящий в текст необходимые исправления. В качестве адресата могут выступать литературный или научный редактор, корректор, потенциальные читатели (конструкторы, программисты, руководители). Пользователем ЛИНАР может быть, например, автор обрабатываемого текста, желающий взглянуть на него "со стороны", или научный руководитель работы, обеспокоенный терминологическими и стилистическими неувязками в текстах разделов, подготовленных различными участниками проекта.

Обработка текста с помощью системы ЛИНАР включает в себя в общем случае несколько циклов (как и при подготовке текста "вручную"), каждый из которых оформляется как самостоятельный сеанс работы с системой. В начале сеанса пользователь формирует задание на обработку текста, для выполнения которого система загружает необходимые информационные модули и вызывает программы контроля текста. Каждая программа проверяет

некоторое определенное свойство текста, т.е. реализует одноаспектный контроль текста. Таким образом, в структурном плане систему ЛИНАР можно считать пакетом прикладных программ; сеанс работы с ней состоит из серии одноаспектных проверок текста или его фрагментов.

Основная технологическая схема использования системы ЛИНАР предусматривает, что текст хранится на машинных носителях и обрабатывается программами контроля, формирующими протокол замечаний по тексту (иногда система предлагает свой вариант исправления). Далее пользователь просматривает эти замечания и, если он с ними соглашается, вносит необходимые изменения в текст с помощью текстового редактора. Измененная версия текста может быть объектом обработки в следующем сеансе. В зависимости от объема текста пользователь может выбрать диалоговый или пакетный режим работы с системой. В последнем случае протокол замечаний формируется на внешнем носителе.

Часто написании отдельных фрагментов текста разными авторами для обозначения одной и той же сущности могут быть использованы различные термины, что усложняет понимание текста. Автоматическое обнаружение подобных конфликтов требует привлечения глубоких знаний о понятийном и терминологическом аппарате предметной области, и в ЛИНАР не реализуется. Однако в составе системы имеется программа контроля, которая может сформировать по фрагментам текста списки используемых терминологических словосочетаний. На основе этой информации решить терминологические проблемы человеку будет значительно проще, чем при обработке текста "вручную".

ЛИНАР не только обнаруживает неточности, ошибки, но и может "объяснить" пользователю суть своих замечаний, а также предложить способы устранения ошибок. Так, например, в случае орфографической ошибки система предлагает свой вариант исправления слова, в случае нарушения естественного порядка слов - правильный порядок слов и т.д. Рекомендации системы призваны помочь пользователю в улучшении текста, направляют его деятельность.

База знаний системы ЛИНАР

Контроль текста, осуществляемый системой ЛИНАР, основывается на использовании знаний о том, что такое правильный, хороший текст. Совокупность этих знаний называется контролирующими знаниями, или К-знаниями. При формировании К-знаний учитывались результаты лингвистических, психологических работ, исследований по эргономике; принят во внимание опыт редакторов, корректоров, нормоконтролеров.

К-знания должны обеспечить возможность оценки текста с различных сторон:

- соответствие общеязыковым нормам;
- соответствие "внешним" нормам, например, требованиям ГОСТов, регламентирующих форму изложения материала в научно-технических документах;
- сложность восприятия текста потенциальным читателем;
- семантическая корректность текста (соответствие выявляемых в тексте семантических отношений и понятийной модели предметной области).

Часть К-знаний (процедурная составляющая) представлена программами одноаспектного контроля. Каждая программа фиксирует строго определенное свойство текста или строго определенный дефект текста (конфликтную ситуацию). Затем формируется соответствующее диагностическое сообщение, которое, в зависимости от выбранного режима работы, либо сразу предъявляется пользователю, либо включается в протокол замечаний.

Важным компонентом информационного обеспечения системы ЛИНАР является и лингвистическая база знаний, содержащая базовые общие знания о русском языке. Кроме того, ЛИНАР использует тематический словарь и тезаурус предметной области, к которой относятся обрабатываемые тексты, и описания нормативных требований, предъявляемых к текстам. Соответствующие информационные массивы создавались разработчиками системы на основе общеязыковых и предметно-ориентированных словарей и справочников, Государственных стандартов и отраслевых инструкций по оформлению текстовых документов.

База знаний ЛИНАР содержит также заранее формируемый - и пополняемый в ходе эксплуатации системы - ***банк адресатов***: конкретных читателей или определенных однородных групп читателей (конкретный руководитель научно-исследовательского проекта; конкретный представитель руководства организации-заказчика; инженеры, которые будут создавать описываемый программно-аппаратный комплекс и др.). Настройка на адресата производится в начале очередного сеанса работы с ЛИНАР и позволяет моделировать процесс восприятия текста разными людьми и, следовательно, оценивать качество текста с разных точек зрения. При такой настройке могут меняться базовые и тематические лингвистические знания (состав словаря, совокупность грамматических правил), степень жесткости требований по соблюдению тех или иных норм и условий.

Таким образом, К-знания ЛИНАР (которые служат критерием корректности текста и используются для обнаружения "дефектов" текста - отклонений от требований, предъявляемых К-знаниями) формируются динамически в каждом конкретном сеансе работы с системой и являются комплексными по своей природе. Они включают как процедурные знания об исследуемом аспекте текста (воплощенные в соответствующих программах контроля), так и декларативные знания, фильтруемые и конкретизируемые в начале каждого сеанса.

Обнаруженные программой контроля несоответствия текста и К-знаний могут быть устранены двумя способами:

- путем внесения изменений в текст (это наиболее частый случай: несоответствие - суть ошибка, допущенная в тексте, которую необходимо исправить);
- путем изменения К-знаний системы.

Заметим, что изменениям подвергается лишь один компонент К-знаний - лингвистические знания, причем не все, а лишь те, которые соответствуют наиболее подвижной части естественного языка - лексикону. Как правило,

такие изменения заключаются в пополнении базы знаний, например, в создании новой словарной статьи для слова, впервые встретившегося в тексте и не знакомого системе.

Знания, отображающие требования семантической корректности и простоты интерпретации, общеязыковые и внешние нормы, может изменять только администратор системы.

Для внесения изменений в базу лингвистических знаний используются сервисные программы; для изменения текста - подсистема редактирования ЛИНАРа.

Отметим, что (даже при работе с ЛИНАР в диалоговом режиме) редактирование текста обычно производится по завершении работы программ контроля. Это связано с тем, что исправление фиксируемых системой ошибок и неточностей зачастую требует переделки относительно больших фрагментов текста (разбиение длинной фразы на несколько более простых, устранение неоднозначности трактовки и т.п.). Однако некоторые - локальные - изменения можно внести в текст сразу же в момент обнаружения ошибки. Поэтому в ряде программ контроля, например, в программах орфографического уровня, предусмотрена возможность исправления фиксируемых ошибок в момент их обнаружения.

Программы контроля

Программы контроля текста могут быть классифицированы по нескольким критериям.

Первый критерий связан с анализируемым программой аспектом текста. В соответствии с этим критерием выделяются следующие группы программ одноаспектного контроля:

- контроль орфографии (включая поиск ошибок в склонении и спряжении слов);
- анализ лексического состава текста;
- стилистический контроль;
- проверка выполнения правил структуризации текста;
- контроль синтаксической структуры;
- пунктуационный контроль;
- семантический контроль.

Рассмотрим несколько примеров работы программ синтаксического и семантического контроля:

```
1)
Рассмотрим структуру памяти вычислительной машины, в которой хранятся команды.
СЛОВО которой ИМЕЕТ БОЛЕЕ ОДНОГО СЛОВА-ХОЗЯИНА В ГЛАВНОМ ПРЕДЛОЖЕНИИ: машины, памяти, структуру
Каждому каналу соответствует свое устройство, которые в свою очередь связаны с главной ЭВМ.
СЛОВО которые НЕ ИМЕЕТ СЛОВА-ХОЗЯИНА В ГЛАВНОМ ПРЕДЛОЖЕНИИ
Мощь языка Си - результат выявления его авторами потребностей программистов, которые возникают при программировании на языке ассемблера.
СЛОВО которые ИМЕЕТ БОЛЕЕ ОДНОГО СЛОВА-ХОЗЯИНА В ГЛАВНОМ ПРЕДЛОЖЕНИИ: программистов, потребностей, авторами
2)
Все рассматриваемые программы написаны на ассемблере.
НЕСОВПАДЕНИЕ СЕМАНТИЧЕСКИХ КЛАССОВ!
В ОПИСАНИИ ГЛАГОЛА "написать" СЕМ.-КЛАСС АКТАНТА:
=язык_программирования=
РЕАЛЬНЫЙ АКТАНТ ассемблере ИМЕЕТ СЕМ.-КЛАСС: =транслятор=
Схема прерываний подключается к магистрали.
НЕСОВПАДЕНИЕ СЕМАНТИЧЕСКИХ КЛАССОВ!
В ОПИСАНИИ ГЛАГОЛА "подключаться" СЕМ.-КЛАСС АКТАНТА:
=устройство=
РЕАЛЬНЫЙ АКТАНТ схема прерываний ИМЕЕТ СЕМ.-КЛАСС:
=структура2=
3)
Снижение напряжения вызвало отключение принтера.
НЕОДНОЗНАЧНАЯ ИНТЕРПРЕТАЦИЯ!
  1 трактовка:
=причина=    : снижение напряжения
=следствие=  : отключение принтера
  2 трактовка:
=причина=    : отключение принтера
=следствие=  : снижение напряжения
4) Каждому каналу сопоставлено определенное устройство. Они, в свою очередь, связаны с главной ЭВМ.
ДЛЯ МЕСТОИМЕНИЯ они В ПРЕДШЕСТВУЮЩЕЙ ФРАЗЕ НЕ НАЙДЕНО СЛОВ,
НА КОТОРЫЕ ЭТО МЕСТОИМЕНИЕ ССЫЛАЕТСЯ
Рассмотрим структуру памяти ЭВМ. Она состоит из двух основных частей.
ДЛЯ МЕСТОИМЕНИЯ она В ПРЕДШЕСТВУЮЩЕЙ ФРАЗЕ НАЙДЕНО БОЛЕЕ ОДНОГО СЛОВА,
НА КОТОРОЕ ССЫЛАЕТСЯ ЭТО МЕСТОИМЕНИЕ:  ЭВМ, памяти, структуру
5)
Информация передается в сопроцессор АК-34 по 16 каналу.
ОБЪЕКТ: сопроцессор АК-34
ГРУППА: 16 каналу
ВЫХОД ЗНАЧЕНИЯ ЗА ВЕРХНЮЮ ГРАНИЦУ ДИАПАЗОНА
(СОПРОЦЕССОР АК-34 ИМЕЕТ КАНАЛЫ: 0,1,2, ... 15)
```

Общение человека с системой ИИ (естественный язык и естественность общения)

Наиболее существенными и привлекательными (в контексте задачи общения с системой ИИ) свойствами ЕЯ являются:

- максимально широкое использование его человеком в своей повседневной деятельности (это избавляет от необходимости специального изучения формализованного языка общения с ЭВМ и от трудностей, связанных с формулировкой заданий и запросов на таком языке);
- возможность использования естественного языка для выражения качественно различного содержания с любой доступной или желательной человеку степенью строгости и полноты (что гарантирует чрезвычайную широту сферы общения - как в плане охвата самых разнообразных предметных областей, так и в плане варьирования формулировок);
- его открытость и способность служить метаязыком для самого себя (что обеспечивает расширяемость используемых языковых средств).

Эти обстоятельства (обычно упоминается первое - не только потому, что оно действительно важно, но и потому, что оно абсолютно очевидно, лежит на поверхности) служат очень серьезными доводами в пользу общения с системами ИИ именно на естественном языке. Пока исследования носили чисто экспериментальный характер, эти доводы были достаточны. Однако на нынешнем этапе, для которого как уже отмечалось, характерна практическая переориентация работ, возникают новые проблемы, ранее остававшиеся в тени.

Часть из них: необходимость отчуждения системы от разработчика, надежность и устойчивость ее функционирования, эффективность реализации, наличие средств сопровождения - возникает и при создании традиционного программного обеспечения. Новые моменты связаны с использованием для общения с машиной именно естественного языка.

Среди проблем, особо актуальных на нынешнем этапе исследований и разработок, укажем:

- тщательный анализ вопроса целесообразности использования естественного языка в человеко-машинном общении;
- поиск ситуаций, в которых общение с машиной на естественном языке оправдано технологически и эргономически;
- выявление обстоятельств, учет которых обеспечивает человеку комфортные, естественные условия общения с компьютером;
- анализ пригодности использовавшихся ранее подходов и методов в изменившихся (практическая переориентация) условиях.

Перед автором некоторого искусственного языка общения с машиной (например, языка программирования), конечно же, не стоит вопрос о целесообразности использования созданного языка по его прямому назначению. При оценке такого языка речь может идти о выразительных средствах, эффективной реализуемости, легкости усвоения и т.п. Отдельные неудачные решения могут быть изменены в ходе доработки и отражены в разного рода пересмотренных сообщениях и др. Объективация языка заключается в создании стандартов, трансляторов, формировании круга пользователей.

Естественный же язык изначально дан разработчикам систем ИИ извне, он объективирован (и активно используется в речевой практике) в большой социальной группе носителей данного языка, которые привыкли к вполне определенным, человеческим условиям общения (в том числе, рассмотренным в начале данной главы). Если эти условия (человеческий фактор) будут игнорироваться, язык общения, возможно, сохранив внешнее сходство с тем или иным ЕЯ, потеряет главное - естественность. А учет этих условий требует от разработчиков систем ИИ очень серьезных дополнительных усилий, поскольку предполагает воссоздание (моделирование) нетривиальных человеческих механизмов работы с языком, наделение системы ИИ - как "собеседника" пользователя - основными чертами (на уровне информационных процессов) собеседника-человека.

Поэтому при создании систем ИИ практической ориентации следует тщательно проанализировать, оправданы ли интеллектуальные и материальные затраты (весьма значительные, в нынешних условиях отсутствия в нашей стране рынка готового программно-информационного обеспечения) на их разработку, экономична ли (с учетом ресурсоемкости) их эксплуатация.

Серьезная практическая задача обеспечения общения с ЭВМ на естественном языке требует серьезного и практичного подхода. В каждой конкретной ситуации необходимо учитывать основательность доводов в пользу общения с системой именно на естественном языке, помнить о реально предоставляемых пользователю удобствах (в частности, об утомительности клавиатурного ввода, о возможностях - пока весьма скромных - технических средств обеспечения общения: устройства распознавания и синтеза звучащей речи, читающие автоматы).

Стремление разработчика или заказчика не отстать от моды, создать "высокоинтеллектуальную" информационную систему, оснащенную средствами естественноязычного интерфейса, не является достаточно веским основанием, а дилетантский подход (в этой новой и чрезвычайно сложной области особенно) не только не приводит к успеху, но и дискредитирует саму идею общения с ЭВМ на естественном языке.

Задачу обеспечения **естественного общения** человека с машиной можно принять без каких бы то ни было оговорок. Однако ниоткуда не следует, что наиболее удобным и естественным для пользователя (и целесообразным, с точки зрения разработчика) средством такого общения будет естественный язык. Пререкания с "непонятливым" компьютером, ориентированным на ведение диалога с человеком на так называемом ограниченном естественном языке, могут потребовать более значительных усилий, чем изучение искусственного формального языка общения.

Нас интересуют ситуации, в которых необходимость использования естественного языка диктуется глубинными внутренними причинами:

- характер поручаемых системе ИИ заданий, а следовательно, и адресуемых ей сообщений таков (носит предварительный, неформальный характер), что описать их на каком-либо формализованном языке крайне трудно;
- общение пользователя с машиной происходит эпизодически и/или в очень широкой сфере (изучение специального языка общения нецелесообразно, или же он становится необъятным). Часто в таких случаях альтернативой использованию естественного языка может служить хорошо разработанная схема "выспрашивания" нужной системе информации с помощью традиционных средств организации дружелюбного интерфейса (например, меню).

Весьма интересен феномен появления "гибридных" знаковых систем - искусственных языков, обладающих рядом свойств ЕЯ. Однако любые формальные средства общения требуют предварительной экспликации, формализации той ситуации, о которой идет речь, привлечения знаний об используемых в системе способах структуризации знаний. Формулируя же адресуемые машине сообщения на естественном языке, человек может не знать: каким набором семантических единиц располагает система ИИ; в какие формальные структуры будет отображаться содержание сообщения; какие компоненты описываемой ситуации являются в настоящий момент значимыми, а какие - второстепенными.

Более того, оказавшись в абсолютно новых условиях, человек, как правило, не сможет воспользоваться ограниченным формальным языком. Эти обстоятельства могут оказаться решающими при выборе языка общения с машиной как для профессионала, который на очередном этапе решения задачи не располагает пока адекватной формальной моделью и вынужден по этой причине отказаться от привычного для него искусственного языка, так и для конечного пользователя, для которого содержательный уровень общения является единственно доступным.

Эксперименты с лингвистическим процессором АДАМАНТ

* * * РАБОТАЕТ ЛП "АДАМАНТ" * * *
КТО?
БОГАЧЕВ
РЕЖИМ РАБОТЫ?
ТЕСТОВЫЙ
ВЫ РАБОТАЕТЕ В ТЕСТОВОМ РЕЖИМЕ!
ЛЮБОЕ ВХОДНОЕ СООБЩЕНИЕ РАССМАТРИВАЕТСЯ ЛИБО КАК КОНТРОЛЬНАЯ ИЛИ ОБУЧАЮЩАЯ ФРАЗА
(НА ЭКРАНЕ РАСПЕЧАТЫВАЕТСЯ НАЙДЕННАЯ ИНТЕРПРЕТАЦИЯ),
ЛИБО КАК ОЦЕНКА ИЛИ УКАЗАНИЕ.
СТЕПЕНЬ САМОСТОЯТЕЛЬНОСТИ СТАНДАРТНАЯ?
ДА
СТЕПЕНЬ ПОДРОБНОСТИ СТАНДАРТНАЯ?
НЕТ
ЗАДАЙТЕ, ПОЖАЛУЙСТА, СТЕПЕНЬ ПОДРОБНОСТИ: ЧИСЛО ОТ 0 - МИН., ДО 5 - МАКС.
4
МОГУ ЛИ Я РАССЧИТЫВАТЬ НА ВАШУ ПОМОЩЬ НА ЭТАПЕ АНАЛИЗА?
ДА

По фамилии пользователя и выбранному им режиму работы ЛП настраивается на очередной сеанс, считывая в свою оперативную память необходимые процедуры и описания уместных в данном сеансе языковых единиц (в частности, словарь пользователя по фамилии Богачев). Адресуемые данному ("рядовому") пользователю вопросы будут формулироваться в таком виде, что для ответа на них не потребуются знания, выходящие за рамки школьного курса русского языка (при диалоге с администратором в вопросах могут появиться лингвистические термины, понятия языковой модели системы). В начале сеанса устанавливаются "степень самостоятельности" ЛП при обработке незнакомых языковых объектов и "степень подробности" выдаваемых на экран терминала сообщений о работе процедур языковой адаптации и обучения (этот параметр можно менять в ходе сеанса); ЛП также спрашивает пользователя, согласен ли он отвечать на его вопросы, возникающие в случаях, когда установленная степень самостоятельности не позволяет процессору изменять лингвистические знания автоматически. При заданной степени подробности (4) процессор будет информировать пользователя о всех обращениях к процедурам языковой адаптации.

ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ
МАЛЬЧИК ИГРАЛ С КОШКОЙ НА БАЛКОНЕ.
ОБРАБАТЫВАЕТСЯ НЕЗНАКОМОЕ СЛОВА "БАЛКОНЕ"
ПРЕДСКАЗАНИЯ: НЕОДУШЕВЛ. СУЩ. В ФОРМЕ ПРЕДЛОЖН. ПАДЕЖА. СЕМ. КЛАСС: =МЕСТО=
УКАЖИТЕ, ПОЖАЛУЙСТА, РОД СУЩЕСТВИТЕЛЬНОГО С ОСНОВОЙ "БАЛКОН" (ВОЗМОЖНЫЕ ОТВЕТЫ: М/Ж/С)
М
КАКОЕ ОКОНЧАНИЕ ИМЕЕТ СЛОВО "БАЛКОН" В ФОРМЕ ТВОРИТ. ПАДЕЖА ЕД. ЧИСЛА (ОМ ИЛИ ЕМ)?
ОМ
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БАЛКОНЕ"
ОНО ВОСПРИНЯТО КАК НЕОДУШЕВЛ. СУЩ. МУЖ. РОДА В ФОРМЕ ПРЕДЛ. ПАДЕЖА ЕД. ЧИСЛА
ОСНОВЕ ПРИПИСАНЫ ГРАММАТИЧЕСКИЕ ПРИЗНАКИ: НЕОДУШ. СУЩ. МУЖ. РОДА, 1 ТИП СКЛОНЕНИЯ.
ОСНОВЕ ПРИПИСАНЫ СЕМАНТИЧЕСКИЕ ПРИЗНАКИ: =МЕСТО=

ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 3.68750 СЕК.)
=ИГРАТЬ2= (ИГРАТЬ)
=СУБЪЕКТ= : МАЛЬЧИК
=ОБЪЕКТ= : КОШКА
=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЛОСЬ=
=МЕСТО= : БАЛКОН (НА)
ПРАВИЛЬНО
НОВАЯ ИНФОРМАЦИЯ ЗАПОМИНАЕТСЯ В ДАННОМ СЕАНСЕ

Пользователь помог ЛП определить грамматические признаки незнакомого ему слова *балкон*; род и тип склонения (ср. *на кузне*, *на окне*, *на ясене*). Семантический класс (=МЕСТО=) установлен автоматически, поскольку группа *на балконе* анализировалась процессором - после заполнения валентностей выбранной трактовки предиката *играть* (2-местный предикат с актантами: "кто (человек)?", "с кем/чем (животное или предмет)?") - как обстоятельство места. Так как пользователь согласился с найденной интерпретацией, ЛП запоминает новую словарную статью слова *балкон*.

ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ
МАЛЬЧИК ИГРАЛ С ДРУГОМ В СРЕДУ.
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "СРЕДУ"
ОНО ВОСПРИНЯТО КАК НЕОДУШ. СУЩ. ЖЕНСКОГО РОДА В ФОРМЕ ВИНИТ. ПАДЕЖА ЕД. ЧИСЛА
ОСНОВЕ ПРИПИСАНЫ ГРАММАТИЧЕСКИЕ ПРИЗНАКИ: НЕОДУШ. СУЩ. ЖЕНСКОГО РОДА, 2 ТИП СКЛОНЕНИЯ
ОСНОВЕ ПРИПИСАНЫ СЕМАНТИЧЕСКИЕ ПРИЗНАКИ: =ИГРА=
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 3.10875 СЕК.)
=ИГРАТЬ1= (ИГРАТЬ)
=СУБЪЕКТ= : МАЛЬЧИК
=ПАРТНЕР= : ДРУГ
=ИГРА= : СРЕДА
=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЛОСЬ=

НЕВЕРНО. СРЕДА - ДЕНЬ НЕДЕЛИ
УКАЗАНИЕ УЧТЕНО. НОВАЯ ИНТЕРПРЕТАЦИЯ:
=ИГРАТЬ1= (ИГРАТЬ)
=СУБЪЕКТ= : МАЛЬЧИК
=ПАРТНЕР= : ДРУГ
=ИГРА= : ?
=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА= : СРЕДА
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЛОСЬ=
ВЕРНО. РЕЖИМ: СП = 2.
НОВАЯ ИНФОРМАЦИЯ ЗАПОМИНАЕТСЯ В ДАННОМ СЕАНСЕ
ИЗМЕНЕН ПАРАМЕТР: СТЕПЕНЬ ПОДРОБНОСТИ (СП)

Попытка процессора воспринять группу *в среду* как один из актантов 3-х местного предиката *играть1* ("кто (человек)?", "с кем (человек)?", "во что (игра)?") оказалась неудачной. Пользователь отверг найденную интерпретацию и определил семантику незнакомого слова *среда* через знакомое ЛП понятие *день недели*. После этого процессор: проанализировал словоформу *среда* с предсказанием "существительное в форме именительного падежа, семантический класс =врем1="; отменил все действия, выполненные с момента начала обработки группы *в среду* (группы *мальчик* и *с другом*, выбранные на роли 1-го и 2-го актантов, повторно не обрабатывались); продолжил анализ фразы. Слово *среда* теперь - по семантическим признакам - не может быть воспринято как название игры. Третья валентность предиката *играть1* остается незаполненной, а на этапе поиска обстоятельств группа *в среду* воспринимается как обстоятельство времени.

Согласившись с новой интерпретацией пользователь (с помощью директивы "РЕЖИМ") уменьшил значение параметра "степень подробности". Поэтому в дальнейшем о незнакомых словах выдается менее подробная информация.

ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ
МАЛЬЧИК СТРИГЕТ КУЗЯВУЮ БУТЯВКУ НА БАЛКНЕ.
"СТРИГЕТ" - ОШИБКА В СПРЯЖЕНИИ! ПРИ ФЛЕКСИИ "ЕТ" В ОСНОВЕ ПРОИСХОДИТ ЧЕРЕДОВАНИЕ:
Г - Ж. ОШИБКА ИСПРАВЛЕНА
КАКАЯ ИЗ ОСНОВ: 1. БАЛКОН, 2. БАЛКН - ВВЕДЕНА ВЕРНО (1 ИЛИ 2, 0 - ЕСЛИ ОБЕ ВВЕДЕНЫ ВЕРНО)?
1
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БУТЯВКУ"
- ОДУШ. СУЩ. ЖЕН. РОДА В ФОРМЕ ВИНИТ. ПАДЕЖА ЕД. ЧИСЛА (13 ТИП СКЛОНЕНИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "КУЗЯВУЮ"
- ПРИЛ. В ФОРМЕ ВИН. ПАДЕЖА ЕД. ЧИСЛА (ЖЕНСКИЙ РОД) (1 ТИП СКЛОНЕНИЯ)
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 4.80000 СЕК.)
=СТРИЧЬ= (СТРИЧЬ)
=СУБЪЕКТ= : МАЛЬЧИК

=ОБЪЕКТ= : БУТЯВКА --АТР -> КУЗЯВАЯ

=ВРЕМЯ= : =СЕЙЧАС=/=ВСЕГДА=/=ИНОГДА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЕТСЯ=/=ДЕЙСТВИЕ ХАРАКТЕРНО=

ВЕРНО

НОВАЯ ИНФОРМАЦИЯ ЗАПОМИНАЕТСЯ В ДАННОМ СЕАНСЕ

При установленных значениях параметров процессор исправляет грамматические и случайные ошибки в знакомых словах самостоятельно (информируя об этом пользователя). Исправление случайной ошибки в слове *балкон*, впервые появившемся только в данном сеансе, возможно только с согласия пользователя.

ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ

ГЛОКАЯ КУЗДРА ШТЕКО БУДЛАНУЛА БОКРА И КУДРЯЧИТ БОКРЕНКА.

ОБРАБАТЫВАЕТСЯ НЕЗНАКОМОЕ СЛОВО "БУДЛАНУЛА"
ЭТО НЕЗНАКОМОЕ СЛОВО ТРАКТУЕТСЯ КАК ГЛАГОЛ-ПРЕДИКАТ. ВЫ СОГЛАСНЫ (ДА, НЕТ)?
ДА
ОБРАБАТЫВАЕТСЯ НЕЗНАКОМОЕ СЛОВО "КУДРЯЧИТ"
ЭТО НЕЗНАКОМОЕ СЛОВО ТРАКТУЕТСЯ КАК ГЛАГОЛ-ПРЕДИКАТ. ВЫ СОГЛАСНЫ (ДА, НЕТ)?
ДА

При анализе этой фразы процессор по синтаксическим шаблонам выделяет в ее составе два ядерных предложения, связанных союзом *и*. В каждом из них на роли слов-предикатов выбираются (с согласия пользователя) формы незнакомых глаголов: *будлануть* и *кудрячить*. Этим глаголам сопоставляется одна из простейших стандартных моделей управления ("кто?", "кого/что?"), которая используется при поиске актантов. На роль подлежащего второго предложения выбирается именная группа *глокая куздра* - подлежащее первого предложения. После уточнения грамматических признаков нескольких незнакомых слов распечатывается полученная интерпретация.

СУЩЕСТВИТЕЛЬНОЕ "БОКР" - ОДУШЕВЛЕННОЕ (ДА, НЕТ)?

ДА
КАК ПРАВИЛЬНО: 1. ГЛОКИЙ ИЛИ 2. ГЛОКОЙ?
1
СУЩЕСТВИТЕЛЬНОЕ "БОКРЕНОК" - ОДУШЕВЛЕННОЕ (ДА, НЕТ)?

ДА
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "ШТЕКО" - НАРЕЧИЕ (ОБР. ДЕЙСТВИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БОКРА" - ОДУШ. СУЩ. МУЖ. РОДА В ФОРМЕ ВИНИТ. ПАДЕЖА ЕД. ЧИСЛА (1 ТИП СКЛОНЕНИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "КУЗДРА"
- ОДУШ. СУЩ. ЖЕН. РОДА В ФОРМЕ ИМ. ПАДЕЖА ЕД. ЧИСЛА (2 ТИП СКЛОНЕНИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "ГЛОКАЯ"
- ПРИЛ. В ФОРМЕ ИМ. ПАДЕЖА ЕД. ЧИСЛА (ЖЕНСКИЙ РОД) (3 ТИП СКЛОНЕНИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БУДЛАНУЛА"
- ГЛАГОЛ (НЕВОЗВР., СОВЕРШ. ВИД) В ФОРМЕ: ПРОШ. ВРЕМЯ, ЖЕН. РОД, ЕД. ЧИСЛО (2 ТИП СПРЯЖЕНИЯ)
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БОКРЕНКА"
- ОДУШ. СУЩ. МУЖ. РОДА В ФОРМЕ ВИН. ПАДЕЖА ЕД. ЧИСЛА (3 ТИП СКЛОНЕНИЯ);
ОСНОВА МНОЖ. ЧИСЛА: "БОКРЯТ"
ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "КУДРЯЧИТ"
- ГЛАГОЛ (НЕВОЗВР., НЕСОВ. ВИД) В ФОРМЕ: НАСТ. ВРЕМЯ, 3 ЛИЦО, ЕД. ЧИСЛО (4 ТИП СПРЯЖЕНИЯ)
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 13.78025 СЕК.)
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БУДЛАНУТЬ)
=СУБЪЕКТ= : КУЗДРА --АТР -> ГЛОКАЯ
=ОБЪЕКТ= : БОКР

=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЕНО=
=ОБР. Д.= : ШТЕКО
======ЗАТЕМ==> =ВЫПОЛНЯТЬ ДЕЙСТВИЕ=
(КУДРЯЧИТЬ)
=СУБЪЕКТ= : КУЗДРА --АТР -> ГЛОКАЯ
=ОБЪЕКТ= : БОКРЕНОК

=ВРЕМЯ= : =СЕЙЧАС=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЕТСЯ=

ВЕРНО

НОВАЯ ИНФОРМАЦИЯ ЗАПОМИНАЕТСЯ В ДАННОМ СЕАНСЕ
ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ

ВЧЕРА КУЗДРА БУДЛАНУЛА БОКРА И ШТЕКО БАРАБАРДАЕТСЯ

ОБРАБАТЫВАЕТСЯ НЕЗНАКОМОЕ СЛОВО "БАРАБАРДАЕТСЯ"
ЭТО НЕЗНАКОМОЕ СЛОВО ТРАКТУЕТСЯ КАК ГЛАГОЛ-ПРЕДИКАТ. ВЫ СОГЛАСНЫ (ДА, НЕТ)?

ДА

ВСТРЕТИЛОСЬ НЕЗНАКОМОЕ СЛОВО "БАРАБАРДАЕТСЯ"
ОНО ВОСПРИНЯТО КАК ГЛАГОЛ (ВОЗВР., НЕСОВ. ВИД): НАСТ. ВРЕМЯ, 3 ЛИЦО, ЕД. ЧИСЛО (1 ТИП СПР.)
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 4.53125 СЕК.)
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БУДЛАНУТЬ)
=СУБЪЕКТ= : КУЗДРА
=ОБЪЕКТ= : БОКР

=ВРЕМЯ= : ИЮНЬ --ДЕНЬ -> 5
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЕНО=
======ЗАТЕМ==>
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БАРАБАРДАТЬСЯ)
=СУБЪЕКТ= : КУЗДРА
=ОБЪЕКТ= : КУЗДРА

=ВРЕМЯ= : =СЕЙЧАС=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЕТСЯ=
=ОБР. Д.= : ШТЕКО

ВЧЕРА КУЗДРА БУДЛАНУЛА БОКРА, И БУТЯВКА БАРАБАРДАЕТСЯ.

ПОСКОЛЬКУ ОЦЕНКИ НЕ БЫЛО, СЧИТАЕТСЯ, ЧТО ПРЕДЫДУЩАЯ ФРАЗА ПОНЯТА ПРАВИЛЬНО
АНАЛИЗ ОЧЕРЕДНОГО СООБЩЕНИЯ:
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 4.01250 СЕК.)
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БУДЛАНУТЬ)
=СУБЪЕКТ= : КУЗДРА
=ОБЪЕКТ= : БОКР

=ВРЕМЯ= : ИЮНЬ --ДЕНЬ -> 5
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЕНО=
========И1 ==>
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БАРАБАРДАТЬСЯ)
=СУБЪЕКТ= : БУТЯВКА
=ОБЪЕКТ= : БУТЯВКА

=ВРЕМЯ= : =СЕЙЧАС=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЕТСЯ=

ВЕРНО

ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ
ГЛОКАЯ КУЗДРА, ШТЕКО БУДЛАНУЛА БОКРА.
ФРАЗА НЕПОНЯТНА (ВРЕМЯ АНАЛИЗА = 3.73025 СЕК.)
НЕСВЯЗНЫЕ ФРАГМЕНТЫ:
1) КУЗДРА --АТР -> ГЛОКАЯ

2) 2) =ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БУДЛАНУТЬ)

=СУБЪЕКТ= : ?
=ОБЪЕКТ= : БОКР

=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЕНО=
=ОБР. Д.= : ШТЕКО

ЗАБЫТЬ: ГЛОКАЯ, КУЗДРА, ШТЕКО, БУДЛАНУТЬ, БОКР

СДЕЛАНО

РЕЖИМ: СП = 0

ИЗМЕНЕН ПАРАМЕТР: СТЕПЕНЬ ПОДРОБНОСТИ (СП)
ЖДУ ОЧЕРЕДНОЕ СООБЩЕНИЕ
ГЛОКАЯ КУЗДРА, ШТЕКО БУДЛАНУЛА БОКРА.
ПОНЯТНО (ВРЕМЯ АНАЛИЗА = 8.01250 СЕК.)
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (БУДЛАНУТЬ)
=СУБЪЕКТ= : ШТЕКО
=ОБЪЕКТ= : БОКР

=ВРЕМЯ= : =РАНЕЕ НАСТОЯЩЕГО МОМЕНТА=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЕНО=
======ПРИ ЭТОМ==>
=ВЫПОЛНЯТЬ ДЕЙСТВИЕ= (ГЛОКАТЬ)

=СУБЪЕКТ= : ШТЕКО
=ОБЪЕКТ= : КУЗДР

=ВРЕМЯ= : =В ТЕКУЩИЙ МОМЕНТ=
=ВИД= : =ДЕЙСТВИЕ ВЫПОЛНЯЛОСЬ=

ВЕРНО. КОНЕЦ.
НОВАЯ ИНФОРМАЦИЯ ЗАПОМИНАЕТСЯ В ДАННОМ СЕАНСЕ
РАБОТАЕТ БЛОК ОБУЧЕНИЯ
СЕАНС ОКОНЧЕН

Последние два примера показывают, что ЛП АДАМАНТ может забывать (директива ЗАБЫТЬ) информацию, полученную в ходе сеанса, и что результат анализа фразы зависит от наличных (и меняющихся во времени) лингвистических знаний.

Завершая сеанс, ЛП запоминает новую информацию в своей долговременной памяти. В данном случае (в словарь пользователя Богачева) будут записаны словарные статьи слов: *балкон, куздр*, *глокать* (глагол, имеющий форму деепричастия *глокая*) и др.

Сферы применения систем автоматической обработки текстов

Системы ***автоматической обработки текста*** (т.е. переработки одного вида текста в памяти ЭВМ в другой) по выполняемым функциям (входной и выходной информации) можно классифицировать следующим образом:

	Язык входного текста	Язык выходного текста
1	Естественный-1	Естественный-2
2	Искусственный	Естественный
3	Естественный	Искусственный / Естественный
4	Естественный	Естественный + { Искусственный}

К системам первого типа относятся программы машинного перевода, получающие текст на некотором естественном языке и перерабатывающие его в текст на другом естественном языке. Второй тип - системы генерации (синтеза) текстов по некоторому формальному описанию. Системы третьего типа, наоборот, перерабатывают текст на естественном языке в текст на искусственном (индексирование, извлечение смыслового содержания) или в другой текст на естественном языке (реферирование). К последнему классу отнесем программы, занимающиеся проверкой текста, написанного на естественном языке. Они в результате своей работы либо исправляют входной текст автоматически, либо формируют некоторый протокол замечаний.

Естественный язык - сложная, многоплановая система, с множеством правил, внутренних связей, имеющая отношение ко всем аспектам деятельности человека. Точность и правильность работы программ определяется глубиной анализа. Достаточно глубокий анализ пока достигается только для определенных узких предметных областей (из-за специфичности подъязыка такой области: в каждой области свои термины, специфические семантические отношения и т.п.).

Для создания систем, работающих со всем естественным языком без потери глубины анализа, в настоящий момент не хватает либо технических возможностей (быстродействия, памяти), либо теоретической базы (например, пока нет даже единой схемы достаточно полного, глубокого и непротиворечивого описания семантики естественного языка). Однако в коммерческих системах, ввиду того, что предназначаются они для большого количества пользователей, разных предметных областей, принята концепция поверхностного анализа, к тому же и производится такой анализ значительно быстрее. Дальнейшее продвижение вперед, использование естественного языка в практических областях невозможно без оснащения этих систем обширными и глубокими (с точки зрения охвата различных явлений языка) описаниями и моделями, созданными лингвистами-профессионалами.

Эта тенденция прогнозируется многими исследователями и прослеживается на примере развития АОТ-систем, уже в наши дни представляющих коммерческий интерес и использующихся при решении следующих прикладных задач:

1. Machine Translation and Translation Aids - машинный перевод;
2. Text Generation - генерация текста;
3. Localization and Internationalization - локализация и интернационализация;
4. Controlled Language - работа на ограниченном языке;
5. Word Processing and Spelling Correction - создание текстовых документов (ввод, редактирование, исправление ошибок)
6. Information Retrieval - информационный поиск и связанные с ним задачи.

Отметим, что это деление несколько условное, и в реальных системах часто встречается объединение функций. Так, для машинного перевода требуется генерация текста, а при исправлении ошибок приходится заниматься поиском вариантов словоформы и т.д.

Машинный перевод

Исторически ***машинный перевод*** является первой попыткой использования компьютеров для решения невычислительных задач (знаменитый Джорджтаунский эксперимент в США в 1954 г.; работы по машинному переводу в СССР, начавшиеся в 1954 г.). Развитие электронной техники, рост объема памяти и производительности компьютеров создавали иллюзию быстрого решения этой задачи. Идея захватила воображение ученых и администраторов. Практическая цель была простой: загрузить в память компьютера максимально возможный словарь и с его помощью из иноязычных текстов получать текст на родном языке в удобочитаемом виде. Однако первоначальная эйфория по поводу того, что столь трудоемкую работу можно поручить ЭВМ, сменилась разочарованием в связи с абсолютной непригодностью получаемых текстов. Приведем в качестве примера результаты работы одной из современных коммерческих систем перевода. Предложим ей перевести народное английское стихотворение, известное нам в переводе "Робин-Бобин" (текст этот очень простой, московские дети изучают его в начальной школе):

Robin, Robin, what a man!	*Малиновка, Малиновка, какой человек!*
He eats as much as no one can.	*Он ест насколько никто не может.*
He ate a lot of fish, he ate a lot of meat.	*Он съел много рыб, он съел много мяс.*
He ate a lot of ice-cream and a sweet.	*Он съел много ледяных-сливки и сладкий.*
He ate a lot of porridge and ten eggs	*Он съел много каша и десять яйцо*
And all the cookies Mother had.	*И вся Мать повары имела.*
He drank a lot of juice, he ate a cake	*Он пил много соков, он съел торт*
Then said: "I have a stomach-ache"	*Затем сказал: "У меня есть желудок- боль"*

Сравним с художественным переводом К.Чуковского:

Робин Бобин Барабек
Скушал сорок человек.
И корову, и быка,
И кривого мясника,
И телегу, и дугу,
И метлу, и кочергу.
Скушал церковь, скушал дом,
И кузницу с кузнецом,
А потом и говорит:
- У меня живот болит!

Следующий пример показывает неустойчивость системы машинного перевода при обработке неоднозначностей. Два предложения по отдельности "*Flyer flies*." и "*Flyers fly.*" переводятся "*Летчик летает*." и "*Летчики летают.*", если же из тех же словосочетаний составить одно предложение "*Flyer flies and flyers fly*" получаем "*Летчик летает и муха летчиков.*".

Конечно, системы, настроенные на определенную предметную область, дают гораздо более приемлемые результаты. Однако в этом случае системы перевода получаются очень узко ориентированными, и попытка использовать их даже в смежных предметных областях дает совершенно непредсказуемые результаты. Подобные эксперименты даже распространены среди любителей пошутить: инструкция по эксплуатации манипулятора-мыши, переведенная с английского языка на русский системой автоматического перевода, использующей специализированный медицинский словарь, превращается в описание всевозможных издевательств над несчастным маленьким грызуном.

Возникают эти проблемы из-за принципиально разных подходов к переводу человека и машины. Квалифицированный переводчик понимает смысл текста и **пересказывает** его на другом языке словами и стилем, максимально близкими к оригиналу. Для компьютера этот путь выливается в решение двух задач: 1) перевод текста в некоторое внутреннее семантическое представление и 2) генерация по этому представлению текста на другом языке. Поскольку не только не решена сама по себе ни одна из этих задач, а нет даже общепринятой концепции семантического представления текстов, при автоматическом переводе приходится фактически делать "подстрочник", заменяя по отдельности слова одного языка на слова другого и пытаясь после этого придать получившемуся предложению некоторую синтаксическую согласованность. Смысл при этом может быть искажен или безвозвратно утерян.

Более реалистичными являются попытки создать системы ***автоматизированного перевода*** - программы, которые не берут на себя полностью весь перевод, а лишь помогают человеку-переводчику справиться с некоторыми трудностями (Computer Aided Translation). Одним из примеров таких систем является Eurolang Optimizer. Его можно рассматривать как нечто переходное между компьютерным словарем и программой-переводчиком, как некий набор предметно-ориентированных глоссариев, снабженный интерфейсом для удобства переводчика: предлагается несколько вариантов перевода, выделенные разными цветами в зависимости от условий применимости; переводчик может с помощью меню определенным образом настраивать словари для более быстрого и правильного выбора нужного эквивалента.

Подобные программные средства могут помочь в решении проблем, связанных с терминологией и вообще со знаниями переводчика о предметной области: одни и те же слова могут по-разному переводиться в зависимости от того, о каком предмете идет речь.

Автоматически может быть решена проблема согласованности. Понятно, что согласованность важна в рамках одного документа: один и тот же термин, даже если его без потери смысла можно перевести несколькими словосочетаниями, должен переводиться одинаково на протяжении всего документа. Однако еще более важной

является согласованность в широком смысле - разработка и применение единой концепции интерпретации одного и того же термина на разных языках (скажем, американский разработчик программного обеспечения может быть недоволен, что термин *dialog box* переводится на итальянский как *finestra* (окно) и как *boite* (коробка, ящик) на французский). Ошибки, возникающие вследствие нарушения согласованности, являются серьезной проблемой, так как, имея только текст-результат перевода, уже невозможно установить, какие термины в оригинале были одинаковыми, а теперь переведены по-разному (в отличие от орфографических ошибок, которые исправить никогда не поздно).

В последнее время также появляются автоматизированные системы "доперевода" или "перевода изменений". Их возникновение связано с тем, что большинство технических текстов (описания, инструкции) не являются целиком новыми (как и явления, продукты, механизмы и т.п., ими описываемые), а содержат в себе лишь некоторые изменения, связанные, например, с усовершенствованием конструкции. Система "доперевода" извлекает из памяти знакомые предложения, а новые куски предлагает переводчику. Заметим, что такой человеко-машинный способ генерации новых текстов также помогает согласованности в стиле и терминологии при переходе от одной версии к другой.

Развитием систем подобного вида можно считать канадскую (Канада - двуязычная страна, постоянно сталкивающаяся с проблемой перевода на государственном уровне) систему генерации прогнозов погоды Forecast Generator (FOG). Можно считать, что в ней перевод полностью заменен генерацией текстов. В памяти системы хранится 20 миллионов слов и словосочетаний, связанных с прогнозами погоды, что позволяет генерировать как английский, так и французский вариант непосредственно из базы данных. Конечно, успешная работа этой системы в значительной мере объясняется ограниченной природой текстов: сообщения о погоде являются классическим примером подъязыка. Ограниченность словаря, грамматики и семантики дает возможность достичь отличных результатов сравнительно простыми методами.

Генерация текста

С необходимостью генерации хотя бы простейших фраз разработчики практических систем столкнулись еще на заре их создания. Даже в столь примитивно организованной (в плане дружественности пользовательского интерфейса) среде, как DOS, при попытке сгенерировать стандартное сообщение о количестве скопированных файлов мы сталкиваемся с проблемой построения фразы: в зависимости от этого количества необходимо использовать разные слова (в английской версии *file* в случае одного файла и *files*, если больше; в русской - и того хуже: могут встретиться варианты *файл*, *файла* и *файлов*, причем правила, в каком случае какой из них использовать, достаточно сложны).

По степени сложности и выразительности существующие методы генерации сообщений принято подразделять на 4 класса (часто используются комбинации методов). Рассмотрим их на примере генерации сообщений о копировании файлов.

1) ***Canned-based methods***

Неизменяющийся шаблон - просто печать строки символов без каких-либо изменений.

Для генерации сообщений создаются таблицы шаблонов, которые будут выдаваться в зависимости от ситуации. В нашем варианте при копировании одного файла будет напечатана первая строка таблицы:

```
1 file copied,
```

а в случае, например, трех - третья:

```
3 files copied
```

2) ***Template-based methods***

Изменяющийся шаблон - бесконтекстная вставка слов в образец-строку (именно этот метод используется в MS-DOS):

Шаблон: `<Число> file(s) copied`

может быть использован для генерации сообщений:

```
0 file(s) copied,
1 file(s) copied,
2 file(s) copied
```

3) *Phrase-based methods*

Контекстная вставка.

В зависимости от вида сообщения (контекста) шаблон может быть несколько изменен. Скажем, система может распознавать, с каким окончанием писать слово *file* в зависимости от их количества.

Шаблон: `<Число> <Определение> <file/files при =1, >1><Глагол: время - прош.>`

может использоваться для генерации сообщений:

```
1 file copied,
2 marked files copied,
2 marked files deleted
```

4) *Feature-based methods*

Синтез сообщения на основе набора свойств (грамматических признаков).

Это наиболее сложный метод, он требует привлечения обширных лингвистических знаний, но, в то же время, он и наиболее привлекателен. Предложение определяется набором характеристик составляющих его слов (например, наличие/отсутствие отрицания, настоящее/прошедшее время) и правилами их сочетаемости.

Шаблон:<Число><Определение><file/files при =1, >1><Глагол: время - любое>
позволяет генерировать сообщения:

```
1 file should be copied,
1 file was copied,
2 marked files were copied
```

Понятно, что генерация логически связных, целостных текстов является гораздо более сложной задачей: к правилам построения предложений добавляются правила их сочетаемости, правила развития сюжета, соблюдения стиля и т.п. Ввиду невозможности их полной формализации задачу генерации полноценных художественных текстов можно считать на настоящий момент неразрешимой. Однако для некоторых специализированных технических текстов эти правила строго оговорены некоторыми стандартами, немногочисленны и поэтому поддаются формализации. Примером таких текстов могут служить различные инструкции, техническая документация, тем более задача ее автоматической генерации давно назрела.

На Западе уже давно разработка документации превратилась в особую подотрасль разработки любых достаточно сложных систем (в том числе программного обеспечения). Сопроводительная техническая документация весьма разнообразна: руководство пользователя, руководство для менеджера (администратора) системы, руководство по монтажу (инсталляции) и первичному запуску, руководство по эксплуатации, руководство по интегрированию системы с другими устройствами (программами), проектные материалы и т.д. Однако часто пользователь не получает своевременно и в полном объеме необходимый ему материал, соответствующий используемой им версии системы. Это можно объяснить двумя причинами. Во-первых (субъективная причина), подготовка документации - это дополнительная работа, требующая дополнительного времени и дополнительных навыков (разработчику трудно изложить требуемое на понятном рядовому пользователю языке, остальным же надо сначала детально изучить систему). Во-вторых (объективная причина), документация устаревает по ходу модернизации системы.

Поиски решения этих проблем привели в свое время к появлению новой профессии "технического писателя". Однако понятно, что привлечение дополнительных работников ведет к удорожанию продукта. Поэтому в последние годы появились практические системы, осуществляющие помощь в разработке документации, вплоть до ее автоматической генерации. Форма и содержание документации часто выбирается не столько из соображений удобства и полезности для пользователя, сколько из соображений простоты ее создания.

Документация, как правило, содержит графическую и текстовую части. Графическую часть проще сформировать, однако без текстовой не обойтись: в ней описывается семантика продукта (назначение, технические данные, ограничения, детализация работы в разных режимах). Очевидно, что качественная система должна генерировать текст, правильный с точки зрения грамматики и синтаксиса естественного языка. Поскольку предметная область точно определена, а техническая документация составляется по определенным строго заданным правилам, степень формализации в постановке данной задачи существенно выше, чем в задаче машинного перевода, что позволяет надеяться на более высокие результаты.

Локализация и интернационализация

Для того чтобы иметь успех на международном рынке, программные продукты должны быть локализованы, т.е. приспособлены к культурным и языковым нормам потенциальных покупателей.

Для многих программных приложений локализация может быть сравнительно простой, когда основная программа (алгоритм) изменяется незначительно. Конечно, опции меню, сообщения об ошибках, экранные подсказки и другие текстовые строки, вставленные в программу, должны переводиться, но это не создает особых проблем, если при разработке приложения была предусмотрена возможность локализации. Для решения этой задачи программный код и текст должны быть разделены. По установленному стандарту текстовые строки оформляются в отдельном файле, вызываемом из программы. Таким способом текстовые строки можно переводить, не затрагивая исходный код.

Подобные принципы облегчения локализации возможны не для всех приложений. Системы, в которых естественный язык используется не только для формирования сообщений на экране, но и является предметом деятельности самой системы (например, программы-автокорректоры), поддаются локализации с большим трудом. Здесь могут потребоваться большие специализированные словари и полная переработка алгоритмов. Часто эта задача настолько сложна, что разработчик ею заниматься не может, и проблема локализации приложений является заботой пользователя-носителя языка.

В идеале для нашего многоязычного мира программные средства должны быть интернациональными; пользователь, купив версию программы для некоторого языка, не должен покупать другую версию для другого. Назрела необходимость иметь программные средства, позволяющие автоматически настраивать приложение на заданный язык. Пока мы довольно далеки от этой цели, но работы в этой области ведутся с большой интенсивностью, особенно в Европе, где в связи с образованием Европейского Союза возникает необходимость вести дела и документацию на всех официальных и некотором количестве неофициальных языков.

Работа на ограниченном языке

Одним из способов разрешения проблем, связанных с обработкой естественного языка, является упрощение и некоторая формализация самих текстов: использование ограниченного языка (подмножества языка). Под ограниченным понимается упрощенный язык, использующий ограниченный словарь, грамматику, строго определенные несложные синтаксические конструкции. Обычно в нем запрещаются длинные предложения, длинные цепочки существительных (типа "*решение проблемы разработки систем перевода на базе представления текста в виде последовательности предложений...*"), не используются пассивные и негативные конструкции, вводятся строгие правила использования терминов. Тексты должны соответствовать одному из стандартных стилей или даже быть составлены по определенному шаблону, принятому в данной предметной области для документов подобного рода.

Эти правила не являются современным изобретением: именно их обычно применяют при написании технической документации. Достаточно "древним" примером ограниченного языка является "Бэйсик Инглиш", введенный англичанами для общения с туземным населением в колониях. Неожиданно он оказался полезен и для общения самих туземцев друг с другом: колонизация ввела в их быт множество предметов и понятий, просто не имеющих названий в их родных языках. Забавно, что через много лет при "колонизации" Европы и всего мира англоязычными техническими средствами используются практически те же методы. Например, все специалисты в области компьютерной техники пользуются английскими терминами (*файл*, *принтер* и т.д.), не пытаясь подыскать эквивалент на родном языке, и мы по-русски говорим *word для windows*, а не *слово для окон*.

Применение ограниченного языка делает документ более понятным, удобным для восприятия, он становится легче для переводчиков, поскольку дает меньше возможностей для неоднозначного толкования: такой документ легче составить автору, не являющемуся носителем языка документа. Правительства, особенно в Европе, начинают вводить стандарты на подготовку документации, нормы, по которым требуется использование ограниченных языков, особенно в международной торговле. В связи с этим возникает потребность автоматизации проверки соответствия текста правилам ограниченного языка; появляется задача создания систем, осуществляющих перевод с естественного языка на ограниченный.

Boeing, Caterpillar и несколько других компаний призвали вести всю документацию только на ограниченном языке. Ими разработана система Boeing Simplified English Checker для проверки соответствия текстов различным промышленным стандартам и государственным нормам. На ее базе создается программа Clearcheck, не только контролирующая правильность текста на ограниченном языке, но и исправляющая ошибки.

Некоторые разработчики прогнозируют создание систем с использованием ограниченных языков, в которых полный и корректный перевод документации будет производиться без вмешательства человека.

Создание текстовых документов (ввод, редактирование, исправление ошибок)

Нет необходимости говорить о многообразии систем для подготовки текстовых документов: текстовых редакторов, издательских систем и т.п. Они прочно вошли в нашу жизнь, без них не может обойтись ни один пользователь и ни одна область деятельности. Более того, создание текстовых документов - одна из основных сфер применения персональных компьютеров. Использование текстовых редакторов обусловлено не только тем, что они облегчают работу, но и тем, что в последнее время во многих сферах деятельности введены стандарты на подготовку текстов, основанные на применении определенных редакторов.

В отличие от машинного перевода разработка систем редактирования текстов еще на заре своего развития, в 60-е годы, считалась коммерчески перспективной прикладной областью. В настоящее время рынок перенасыщен подобными системами; среди их создателей существует жесткая конкуренция, поэтому при введении одним из поставщиков каких-либо новых возможностей (например, проверка стиля) остальные вынуждены вводить в свои системы нечто подобное. Одним из первых массовых нововведений стало включение в состав текстового редактора программ проверки правописания и внесения необходимых исправлений - ***автокорректоров***. Чтобы придать своему продукту новые коммерчески перспективные свойства, создатели вынуждены все больше использовать лингвистические знания, применять методы морфологического и синтаксического анализа. На очереди - создание систем, выполняющих функции научного редактора, т.е. осуществляющих литературную и научную правку текстов, способных производить сложное автоматизированное редактирование текстов на естественном языке.

Проверка текста в таких системах может вестись в режиме "off-line" - когда формируется протокол замечаний по тексту, либо в режиме "on-line" - когда исправление ошибок ведется по мере их обнаружения (возможно, после получения соответствующего подтверждения от пользователя). При обнаружении ошибки система может предложить вариант ее исправления (при наличии нескольких вариантов - их упорядоченный список). Замечания по тексту также могут носить различный характер. Они могут быть локальными (указывается фрагмент текста с ошибкой) и глобальными (выдается диагностическое сообщение, касающееся всего текста, например: "данный текст труден для восприятия"). В третьей главе мы рассмотрим подробнее проблемы создания систем подобного рода.

Поиск информации

Не вызывает сомнений необходимость автоматизации поиска заданных текстовых фрагментов в текстах на естественном языке.

Однако часто даже при поиске информации другого рода (например, аудио- и видео-) работа на самом деле ведется с описаниями на естественном языке (например, для организации поиска фотографий необходимо снабдить каждую из них набором словесных характеристик типа "портрет, профиль, полный рост, женщина", "пейзаж, лес, осень" и т.п.).

В последних разработках классических систем поиска текста основное внимание уделяется дополнению их разнообразными средствами текстовой обработки, что приводит к расширению возможностей и облегчению работы для пользователя-непрофессионала.

Применение компьютеров не только ускоряет создание и обработку документов, но и чрезвычайно стимулирует рост их количества и объема. Очень многие пользователи регулярно сталкиваются с необходимостью быстро просматривать большой объем документов и выбирать из них действительно нужные. Эта задача возникает при работе с текстовыми базами данных, с электронной почтой, при поиске в Интернете. Сократить количество просматриваемых документов могут помочь системы ***категоризации***. Большой поток входных документов эти системы распределяют по небольшому количеству классов. При категоризации могут учитываться как чисто внешние показатели документов (объем, расширение имени соответствующего файла и т.п.), так и их содержательные характеристики (название, фамилия автора, ключевые слова), которые могут позволить отнести текст к той или иной тематической рубрике. В последнем случае мы имеем дело с ***рубрицированием*** текстов.

Часто бывает, что в крупных организациях, особенно государственных, правила делопроизводства предписывают сопровождать каждый документ кратким описанием или набором ключевых слов. Во всех указанных случаях была бы весьма полезна возможность автоматически составлять сжатые описания содержания документов - рефераты.

К сожалению, автоматические методы не настолько совершенны, чтобы создать полноценный реферат путем генерации предложений текста. Однако уже сейчас возможно ***автоматическое реферирование*** - составление более или менее информативных и связных рефератов заданного объема (квазирефератов**)** - путем выбора информативных предложений из исходного текста, а также выделение достаточно представительного списка ключевых слов.

В качестве ключевых слов система может выбирать слова, наиболее часто встречающиеся в тексте (и являющиеся при этом информативными, т.е. не предлоги, союзы и проч.), либо использовать для отбора какие-либо синтактико-семантические признаки (из фрагмента: *"Определение. Интегралом ... называется ..."* можно заключить, что *интеграл* - ключевое слово).

При реферировании из текста отбираются предложения, в наибольшей степени характеризующие его содержание. Таковыми могут считаться, например, предложения, содержащие ключевые слова (чем больше, тем лучше), либо отобранные по некоторым особым признакам. Размер реферата (коэффициент сжатия) или количество ключевых слов задается пользователем. Результатом работы такой системы может являться некоторый новый текстовый документ (реферат или набор ключевых слов) или же данный документ, в котором ключевые слова или наиболее информативные предложения выделены по тексту.