8. СТРАТЕГИИ ПОИСКА РЕШЕНИЯ

8.1. Поиск как основа функционирования СОЗ

Системы, основанные на знаниях (СОЗ), в том числе и экспертные системы, имеют специфическую организацию. Суть которой заключается в том, что они осуществляют поиск некоторой цели (конечного состояния) на основе исходных посылок и набора фактов, которые характеризуют некоторое начальное состояние (рис. 8.1).

Рис. 8.1. Организация процедуры поиска.

Причем поиск конечного состояния выполняется автоматически на основе, реализованной в системе, основанной на знаниях, стратегии поиска, которая:

- реализует возможность выбора;
- позволяет выполнять шаги от начального состояния к новым состояниям, более или менее близким к цели.

Таким образом, реализованные в СОЗ стратегии поиска отыскивают цель, «шагая» от одного состояния системы к другому. При этом они обеспечивают распознавание ситуации, когда они находят *цель* или попадают в *тупик*. Как правило, на промежуточных стадиях вычисляется некоторое число (критерий), посредством которого программа поиска оценивает свой ход и определяет дальнейшее направление поиска требуемого конечного состояния.

При этом цель может быть одна или же может иметься некоторый набор приемлемых целей (конечных состояний). Большинство современных СОЗ работают именно по этой модели.

Рассмотрим процесс поиска на конкретном примере. Пусть мы имеем карту дорог (рис. 8.2). По этой карте мы хотим изучить маршрут движения из одного конкретного города в другой. Система, построенная на основе знаний об этой карте дорог, должна поработать с картой и спланировать наше путешествие, например из города «A» в город «F». При решении представленной задачи компьютер преобразует исходную карту и представит ее в виде *дерева поиска*, которое без учета циклов будет иметь вид как на рис. 8.3.

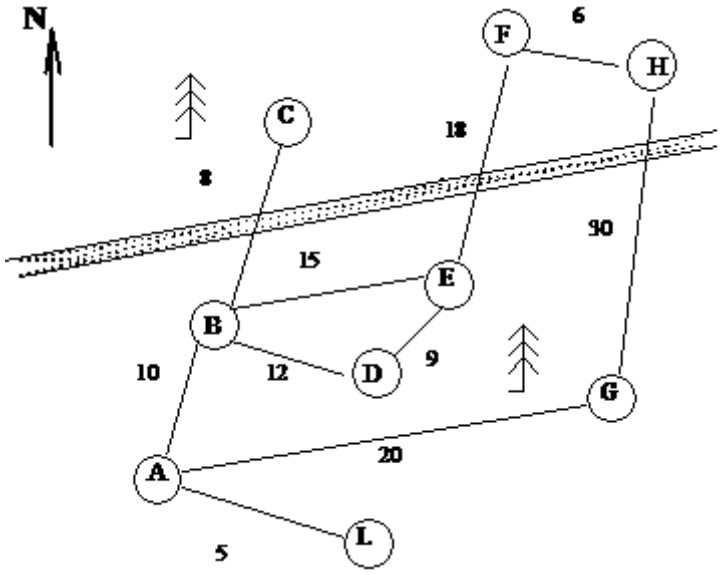

Рис. 8.2. Исходная карта дорог.

Дерево поиска показывает все возможные варианты выбора при движении из «A» (начальное состояние), а также и все варианты выбора в каждом из промежуточных пунктов (состояний). Место, откуда начинается поиск, находится в вершине дерева поиска. Все дороги, начинаясь в «A», либо приводят в тупик, либо заканчиваются в «F».

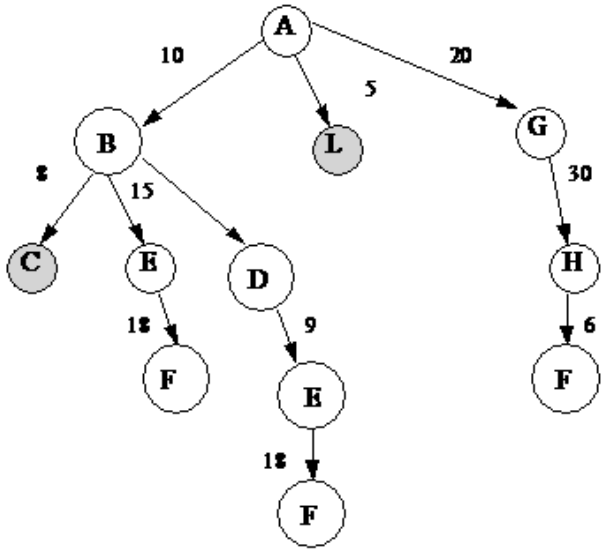

Рис. 8.3. Дерево поиска, построенное на базе исходной карты

Задача компьютера состоит в том, чтобы найти подходящую дорогу из вершины дерева к конечному состоянию (цели) в его нижней части. Человек при взгляде на дерево поиска сразу же выберет подходящую дорогу, но компьютер на это не способен. Какую стратегию поиска применит компьютер?

Рассмотрим пример, который заставит нас думать как компьютер. Мы находимся в темной комнате, на стене которой дерево поиска, а у нас фонарик, который освещает всего два соседних пункта. Нам известно, где расположена вершина дерева, откуда надо начать поиск. Как наиболее эффективно перемещать луч фонарика, чтобы найти маршрут до «F»?

При отсутствии какой-либо дополнительной информации компьютер для поиска использует стратегию «грубой силы». При этом программа поиска осуществляет поиск цели, идя от состояния к состоянию, и систематически исследуя все возможные пути. Для определения каждого последующего шага применяется простая механическая стратегия, которая имеет две основные разновидности: поиск в глубину и поиск в ширину.

8.2. Стратегии поиска в глубину и ширину

Стратегия *поиска в глубину* основана на полном исследовании, одного из возможных вариантов, до изучения других вариантов. Поиск выполняется по несложной методике.

Например, компьютер всегда первой исследует неизвестную левую ветвь дерева. Когда процесс поиска заходит в тупик, он возвращается вверх в последний пункт выбора, где имеются неизученные альтернативные варианты движения, и затем осуществляется следующий вариант выбора (рис. 8.4).

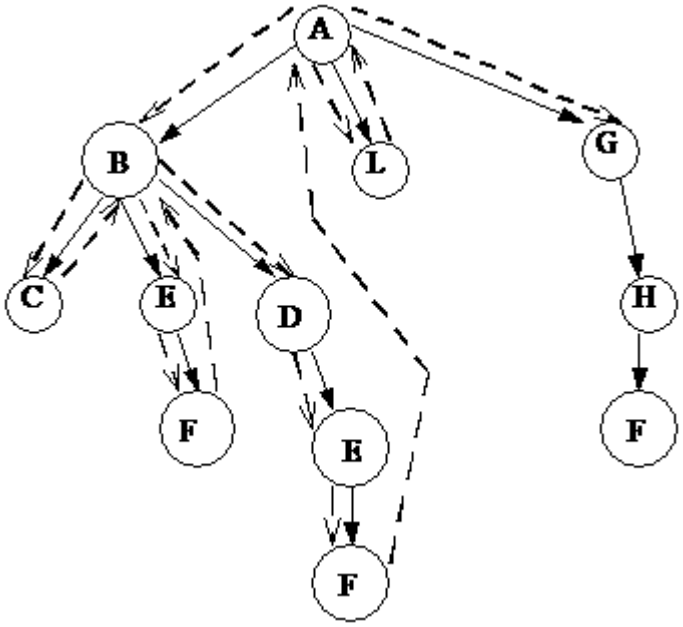

Рис. 8.4. Стратегия поиска в глубину

В среднем поиск в глубину подходит для решения проблемы, где все пути поиска от вершины дерева до его основания имеют одинаковую длину.

Стратегия *поиска в ширину* предусматривает переход в первую очередь к вершинам, ближайшим к стартовой вершине (т.е. отстающим от нее на одну связь), затем к вершинам, отстающим на две связи, затем на три и т. д., пока не будет найдена целевая вершина (рис. 8.5).

Программа поиска продвигается по дереву решений слева направо, расширяя каждый из маршрутов, и отбрасывая те из них, которые являются тупиковыми. Таким образом, поиск в ширину пригоден при решении проблем, где ветви поиска в дереве (от вершины до основания) имеют неодинаковую длину, и не существует указаний на то, какая именно дорога приводит к цели.

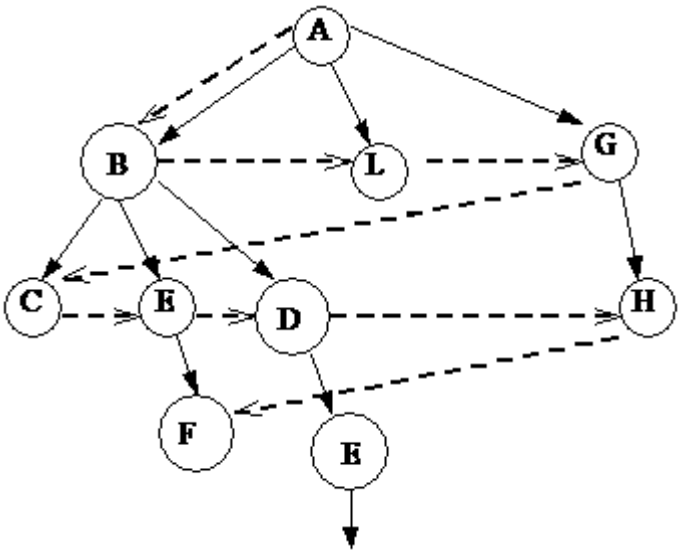

Рис. 8.5. Стратегия поиска в ширину.

Обе эти стратегии предполагают последовательный перебор возможных вариантов. Поиск будет более эффективным, если некоторый механизм в пунктах выбора сам сможет делать наиболее желательный выбор. Это, так называемая, эвристика поиска. Эвристика - это эмпирическое правило, с помощью которого человек - эксперт в отсутствии формулы или алгоритма пытается осуществить свои намерения.

8.3. Стратегия эвристического поиска

Поиск на графах при решении задач, как правило, невозможен без решения проблемы комбинаторной сложности, возникающей из-за быстрого роста числа альтернатив. Эффективным средством борьбы с этим служит эвристический поиск.

Один из путей использования эвристической информации о задаче – это получение численных эвристических оценок для всех вершин пространства состояния. Численная оценка вершины указывает, насколько данная вершина перспективна с точки зрения достижения цели. Идея состоит в том, чтобы всегда продолжать поиск, начиная с наиболее перспективной вершины, выбранной из всего множества кандидатов.

Существует целый ряд эвристических методов. Простейший из них носит название «взбирание на гору». Его принцип состоит в следующем:

- если есть приемлемые варианты выбора, надо выбрать наилучший из них, используя любой критерий рассуждения;
- если попали в тупик, надо вернуться в последнее место, где имеются альтернативные варианты выбора, и сделать иной выбор, наиболее близкий к наилучшему выбору.

Чтобы программа поиска маршрута выполняла эвристический поиск, она должна располагать некоей *базой* для осуществления выбора. Предположим, в качестве базы программа примет соответствующее направления каждой дороги определенному азимуту: предпочтение будет отдаваться той дороге, которая

более всего совпадает по направлению с прямой, соединяющей стартовую и целевую вершины (рис. 8.6).

Такой подход не всегда действенен, но это хорошая стратегия. В нашем примере программа исследует маршрут A → B → C и найдет тупик, затем вернется назад, в то место, где существует альтернатива выбору, т.е. в «B», а затем сразу отправится по маршруту B → E → **F**, который приведет к цели.

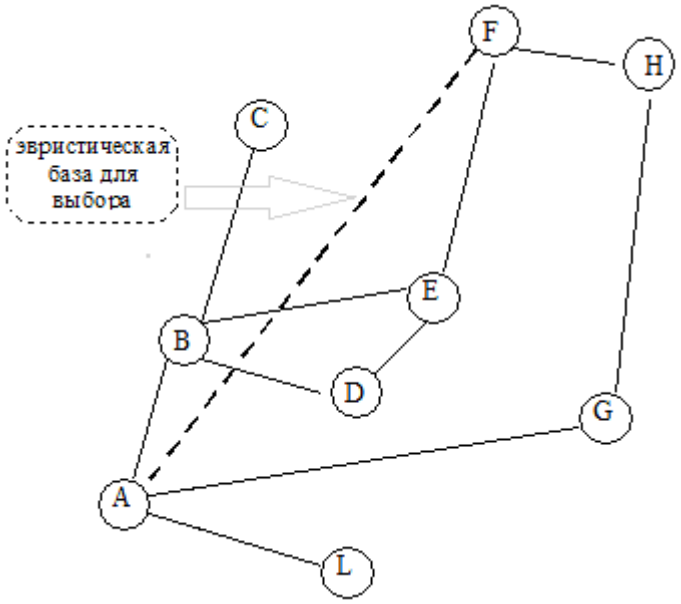

Рис. 8.6. Эвристическая стратегия по методу «взбирание на гору»

Мы рассмотрели разные варианты поиска, но возникает вопрос: откуда взялось само дерево поиска? Обычно машина создает дерево поиска сама на основе предлагаемых ей начальных сведений, причем делает это постепенно в процессе самого поиска.

8.4. Формализация задач в пространстве состояний

Процесс решения задачи, как правило, включает два этапа: представление задачи и реализацию стратегии поиска. Полное представление задачи в пространстве состояний включает в себя:

- описание всех или только начальных состояний;
- задание операторов, отображающие одни состояния в другие;
- задание целевого состояния.

Поиск решения задачи в пространстве состояний сводится к определению последовательности операторов, отображающих исходное состояние в целевое. Таким образом, представление задачи в пространстве состояний определяется совокупностью трех составляющих — тройкой

$$< S_0, F, G>$$

где S_0 – множество начальных состояний, но может состоять и из всего одного элемента; F – множество операторов, отображающих одно состояние в другое и G – множество целевых состояний.

Для описания состояний могут использоваться векторы, матрицы, графы, списки и т.д. Выбор формы описания состояний определяется тем, чтобы применение оператора, преобразующего одно состояние в другое, оказалось наиболее простым. При этом операторы отображения (преобразования) наиболее часто представляются в виде набора правил, задающих возможность перехода из одного состояния в другое.

Так, для рассмотренного ранее примера поиска пути на карте дорог, любое текущее состояние можно описать в виде списка городов, пройденных к текущему моменту, т.е. включенных в маршрут поиска на данный момент. Тогда начальное состояние будет описываться списком, состоящим только из одного элемента, соответствующего начальному пункту пути:

[A]

Конечным состояниям будут соответствовать любые списки, первые элементы которых соответствуют целевой вершине, т.е. конечному пункту маршрута:

[F ¦ _]

и в данном примере конечному состоянию могут соответствовать списки:

[F ¦ [E,B,A]]
[F ¦ [H,G,A]] и т. д.

В свою очередь операторы преобразования одного состояния в другое могут быть представлены в двух различных видах, а именно в виде:

- набора фактов, определяющих дороги между городами;
- набора правил, определяющих возможность перемещения из одного города в другой.

В качестве такого набора фактов для рассматриваемого примера можно указать предикаты, описывающие существующие дороги между городами:

```
дорога (a, b)
дорога (a, g)
дорога (a, e)
  . . .
дорога (g, h)
```

Что касается правил, то для случая простейшей стратегии поиска на графе дорог (рис. 8.4) набор правил может быть следующим:

```
маршрут(Кон_город,Кон_город,[Путь]).
маршрут(Город,Кон_город,[Город|Путь_до_конца]):-
                дорога(Город,Город_к),
                маршрут(Город_к,Кон_город,Путь_до_конца)
```

Графическая иллюстрация правила приведена на рис. 8.7. Из этого рисунка видно, что правило описывает простейшую стратегию поиска в глубину, без учета ограничений по доступу к вершинам графа и возможного зацикливания.

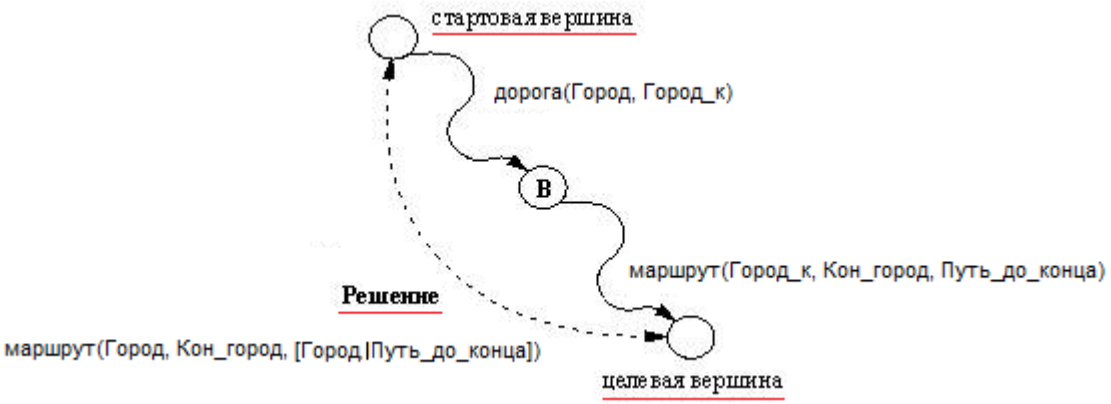

Рис. 8.8. Графическая интерпретация задачи поиска в глубину.

Пространство состояний - это граф, вершины которого соответствуют ситуациям, встречающимся в задаче (проблемные ситуации), а решение задачи сводится к поиску пути в этом графе. Используя граф состояний, процедуры поиска строят на нем дерево решений.

Решаемая задача сводится к построению пути между исходной ситуацией («стартовой» вершиной) и нужной конечной ситуацией («целевой» вершиной). Однако при выполнении поиска на графе могут возникать проблемы с обработкой альтернативных путей поиска.

Существует еще одна разновидность данного класса задач, когда каждому разрешенному ходу или действию может быть приписана его стоимость. *Стоимость решения* - это сумма стоимостей дуг, из которых состоит путь из стартовой вершины в целевую. И он может быть иной, чем в случае, когда стоимости не заданы. Для рассматриваемого примера это может быть поиск наиболее короткого маршрута из одного города в другой. При этом даже если стоимости не заданы, может возникнуть оптимизационная задача: найти кратчайшее решение.

8.5. Представление пространства состояний в виде базы знаний

Но говоря о возможных стратегиях поиска в пространстве состояний, надо выяснить, как можно представить пространство состояний в базах знаний, используя для этих целей, например, язык Пролог. Для описания пространства состояний введем в рассмотрение предикат

после (X,Y) ,

который истинен только тогда, когда в пространстве состояний существует разрешенный переход из вершины X в вершину Y. Будем говорить, что Y – это преемник вершины X. Если с ходом связаны их стоимости, то можно добавить третий аргумент, а именно, стоимость хода:

после (X,Y,S)

Эти отношения можно задавать в программе явным образом при помощи набора соответствующих фактов. Однако такой подход непрактичен и нереален для тех случаев, когда пространство состояний устроено достаточно сложно.

Поэтому предикаты «после/2)» или «после/3)» обычно определяется при помощи правил вычисления вершин - приемников некоторой заданной вершины. Другим важным вопросом является способ представления состояний, т.е. самих вершин. Это представление должно быть компактно, но в то же время обеспечивать эффективное выполнение необходимых операций:

- операций вычисления вершин – приемников,
- а возможно и стоимостей соответствующих ходов.

Несколько в завуалированной форме этот подход уже использовался в предыдущих примерах, а предикат дорога/2 полный аналог предикату после/2. Но для более полного представления об использовании стратегии поиска в пространстве состояний рассмотрим более подробный пример.

8.6. Пример использования поиска в глубину с формированием пространства состояний

Рассмотрим известную логическую задачу о волке, козе и капусте. Старик должен переправить на другой берег реки волка, козу и капусту (рис. 8.9). При этом существует два ограничения:

- Его лодка такова, что за один раз может вместе с ним перевезти только кого-то одного: либо волка, либо козу, либо капусту.
- Звери мирно ведут себя только в присутствии старика. Стоит только ему отлучиться, как волк тут же съест козу или коза съест капусту.

Рис. 8.9. Графическая интерпретация задачи.

Рассмотрим возможность решения этой задачи, используя пространство состояний. Представим пространство состояний в виде направленного графа, любая вершина которого соответствует текущему состоянию процесса перевозки, а дуги возможным переходам между этими состояниям.

Каждую вершину будем представлять списком из четырех элементов: по одному на волка, козу, капусту и старика. Эти элементы будут принимать значения: 0 – если объект на левом берегу и 1 – если на правом берегу. Тогда начальное состояние процесса перевозки, при котором все находятся на левом берегу, будет иметь вид: [1,1,1,1].

По условию задачи требуется переместить всех на правый берег (конечное состояние [0,0,0,0]), учитывая при этом и два указанных ранее ограничения. Исходя из первого ограничения можно определить все возможные состояния

(вершины) и переходы между ними. Хотелось бы подчеркнуть возможные, но не обязательно допустимые. О допустимых состояниях речь пойдет дальше, а сейчас говорим только о возможных состояниях, на основании которых можно описать все пространство состояний. Начальные уровни графа возможных состояний и переходов приведены на рис. 8.10.

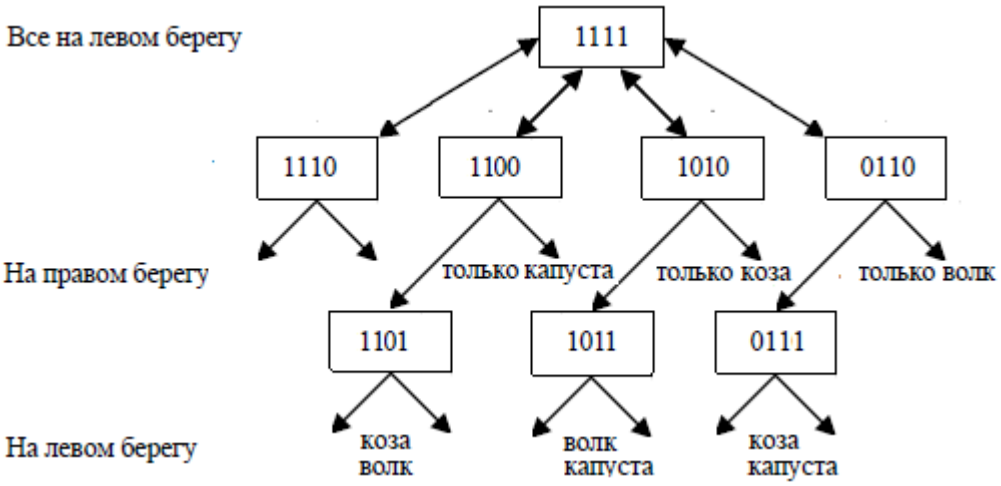

Рис. 8.10. Фрагмент графа (пространства состояний) рассматриваемой задачи.

Из анализа этого фрагмента следует, что все возможные переходы между состояниями определяются только тем, что старик может с одной стороны на другую перевести один любой объект, который находится на той стороне, с которой он начинает свое движение. А коль так, то все возможные переходы могут быть описаны следующим набором фактов:

```
% возможные перемещения
move([1,G,C,1], [0,G,C,0]).        % везем вправо волка
move([W,1,C,1], [W,0,C,0]).        % везем вправо козу
move([W,G,1,1], [W,G,0,0]).        % везем вправо капусту
move([W,G,C,1], [W,G,C,0]).        % порожняком идем вправо
move([W,G,C,0], [W,G,C,1]).        % порожняком идем влево
move([0,G,C,0], [1,G,C,1]).        % везем влево волка
move([W,0,C,0], [W,1,C,1]).        % везем влево козу
move([W,G,0,0], [W,G,1,1]).        % везем влево капусту
```

Набор этих фактов основан на предикате вида move(Sost_x,Sost_y), который описывает возможность перехода из Sost_x в некоторое возможное состояние Sost_y. Так, например, первая строка описывает возможность находящихся на левом берегу старика и волка (значение их элементов 1) переехать на правый берег (значение их элементов станет 0). При этом местоположение как козы (G), так и капусты (C) абсолютно не важно. Самым главным является то, что оно при таком переходе не изменится. Поэтому как в Sost_x, так и в Sost_y они имеют одни и те же значения. Изложенное по отношению к первому факту будет справедливо и для всех остальных фактов.

Приведенный набор фактов наиболее прост для понимания и изложения материала. Поэтому остановимся именно на нем, хотя для людей знакомых со SWI-Prolog'ом можно привести более короткую запись фактов, описывающих все возможные переходы в нашей задаче:

```
move([X,G,C,X], [S,G,C,S]) :- S is \X+2.   % везем волка
move([W,X,C,X], [W,S,C,S]) :- S is \X+2.   % везем козу
move([W,G,X,X], [W,G,S,S]) :- S is \X+2.   % везем капусту
move([W,G,C,X], [W,G,C,S]) :- S is \X+2.   % порожняком идем
```

Вне зависимости от формы записи – это восемь фактов или четыре – у нас появилась возможность описать программно все пространство состояний для рассматриваемой задачи.

Однако, до текущего момента мы не учитывали второе ограничение этой задачи. Другими словами не все возможные состояния являются допустимыми. С этой целью обратимся еще раз к анализу начальных уровней полного графа состояний (рис. 8.11).

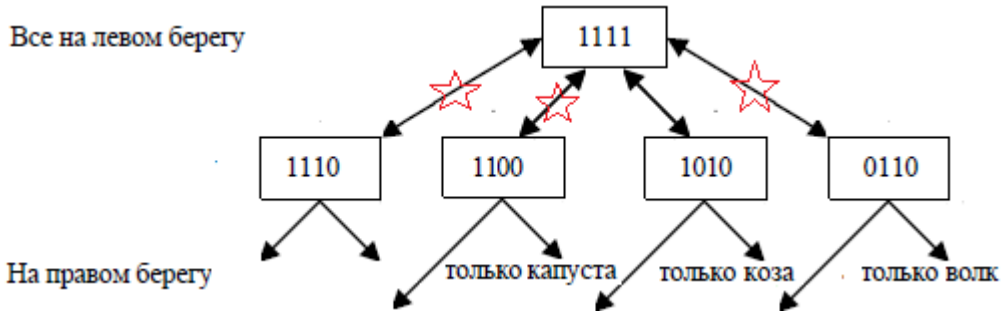

Рис. 8.11. Фрагмент графа с учетом ряда ограничений.

Из него видно, что уже на первом уровне три из возможных состояний являются недопустимыми. Так, например, при переходе [1111] –> [1110], когда старик один уехал на другой берег, совсем не ясно кто раньше: коза съест капусту или волк козу. Аналогично и в других случаях. То есть множество возможных состояний надо ограничить. Это проще сделать, указав состояния, которые недопустимы. Их допустимо описать совокупностью фактов вида:

```
% конфликтные состояния
conflict(1,1,_,0). % фермер-справа, слева – волк, коза
conflict(_,1,1,0). % фермер-справа, слева – коза, капуста
conflict(0,0,_,1). % фермер-слева, справа – волк, коза
conflict(_,0,0,1). % фермер-слева, справа – коза, капуста
```

Но вместо фактов можно записать более короткие правила, которые следуют из того, что как козу с капустой, так и волка с козой нельзя оставлять на том берегу, на котором старик отсутствует.

```
% конфликтные состояния
conflict([X,X,_,F]) :- F =\= X.
conflict([_,X,X,F]) :- F =\= X.
```

В предыдущем разделе для описания пространства состояний был введен в рассмотрение предикат после (X,Y), который истинен только тогда, когда в пространстве состояний существует разрешенный переход из вершины X в вершину Y. Аналогом этого предиката для нашей задачи будет предикат next/2, который определим в виде правила:

```
% выбор преемника
next(Sost_x,Sost_y) :-
                move(Sost_x,Sost_y),
                not(conflict(Sost_y)).
```

После того, как закончено описания пространства состояния, можно сформировать процедуру route/2, реализующие стратегию поиска в глубину. Процедура содержит два правила. Первое из них определяет отсечение в работе процедуры при достижении конечного состояния с выводом на дисплей списка вершин дерева решений. Само дерево решений формируется вторым правилом процедуры route, а последовательность обхода вершин сохраняется в списке P.

```
% конечное состояние – все на правом берегу
route([0,0,0,0],P) :- writeln(P),!.
route(Sost_x,P)    :-
   next(Sost_x,Sost_y),       % выбор преемника
   not(member(Sost_y,P)),     % исключение повторного посещения
   route(Sost_y,[Sost_x|P]). % переход к следующему состоянию
```

Однако при простейшей реализации стратегии поиска в глубину возможны для нашей задачи ситуации при которых старик перевезет, например, козу с левого берега на правый, потом подумает и повезет ее обратно на левый берег. Действительно, все состояния [1,1,1,1] –> [1,0,1,0] –> [1,1,1,1], как и переходы между ними являются доступными в сформулированном нами пространстве состояний. Для исключения подобных зацикливаний во второе правило route вводится дополнительное условие, которое обеспечивает проверку того, чтобы вновь найденная вершина не присутствовала бы в уже сформированном дереве решений. То есть это дополнительное третье ограничение в постановке задачи, которое явно не было описано в формулировке задачи, но предполагалось, исходя из здравого смысла человеческого интеллекта.

Описанные факты и правила можно загрузить, например, в SWI-Prologa, а затем в качестве цели задать запрос вида:

?– route([1,1,1,1],[]),

где первый параметр – это список, определяющий начальное состояние, а второй пустой список. Результат выполнения запроса будет аналогичен тому, что приведен на рис. 8.12. Из списка вершин дерева решений следует порядок перемещений лодки со стариком: коза – вправо, пустой – влево, волк – вправо, коза – влево, капуста – вправо, пустой – влево, коза – вправо.

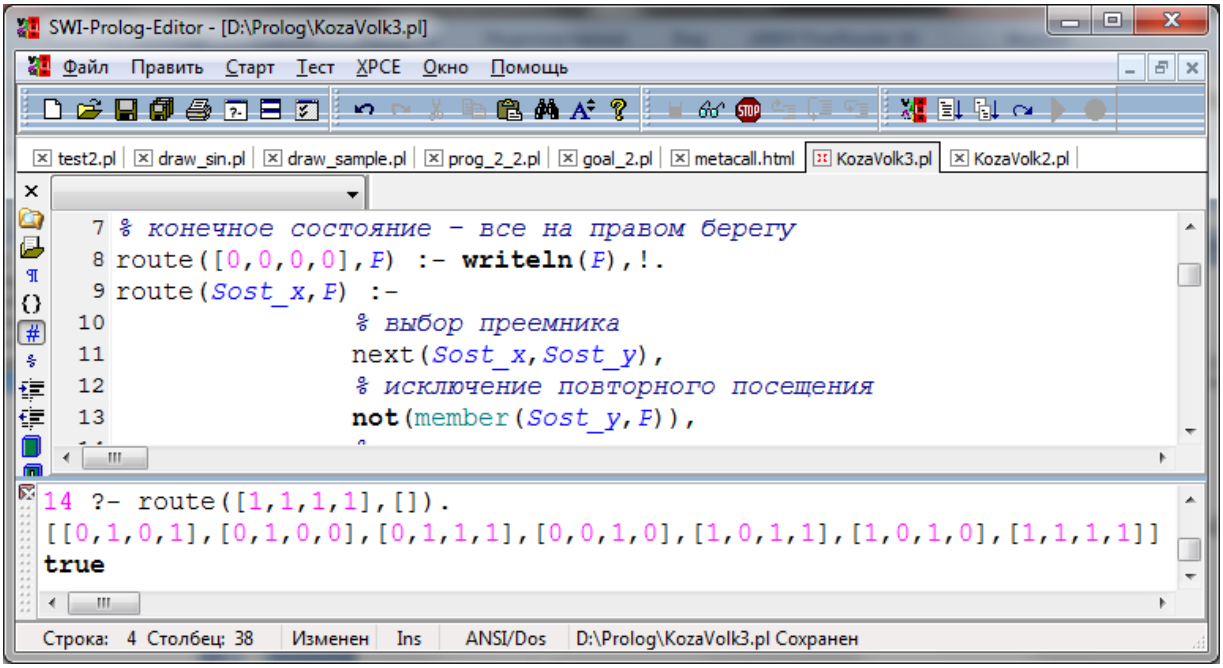

Рис. 8.12. Фрагмент графа с учетом ряда ограничений.

Листинг программы решения данной логической задачи с формированием пространства состояний и реализацией стратегии поиска в глубину с учетом ограничений на языке SWI-Prolog может иметь следующий вид:

```
% ------------------------------------------------------------
% Листинг программы решения задачи "Волк, коза и капуста"
% ------------------------------------------------------------
% W-волк, G-коза, C-капуста, F-фермер
% Состояние: 1-левый берег, 0-правый берег

% конечное состояние – все на правом берегу
route([0,0,0,0],P) :- writeln(P),!.
route(Sost_x,P) :-
   next(Sost_x,Sost_y),       % выбор преемника
   not(member(Sost_y,P)),     % исключение повторного посещения
   route(Sost_y,[Sost_x|P]). % переход к следующему состоянию

% выбор преемника
next(Sost_x,Sost_y) :-  move(Sost_x,Sost_y),
                        not(conflict(Sost_y)).

% возможные перемещения
move([X,G,C,X],[S,G,C,S]) :- S is \X+2.   % везем волка
move([W,X,C,X],[W,S,C,S]) :- S is \X+2.   % везем козу
move([W,G,X,X],[W,G,S,S]) :- S is \X+2.   % везем капусту
move([W,G,C,X],[W,G,C,S]) :- S is \X+2.   % порожняком идем

% конфликтные состояния
conflict([X,X,_,F]) :- F =\= X.
conflict([_,X,X,F]) :- F =\= X.
```