Обработка знаний, выраженных в качественной форме. Факты и правила. Структуры и стратегии поиска в пространстве состояний. Эвристический поиск.

Все задачи решаемые ИС и ИИ можно классифицировать по следующим общим основаниям

- 1) Задачи анализа и синтеза. В задаче анализа задана модель сущности и требуется определить неизвестные характеристики модели. В задаче синтеза задаются условия, которым должны удовлетворять характеристики «неиз вестной» модели сущности, и требуется построить модель этой сущности.
- 2) Статические и динамические. В статических задачах явно не учитывают фактор времени и/или не изменяют знания об окружающем мире в процессе своих решений.
- 3) Использование общих утверждений для представления знаний, не содержащих явных ссылок на конкретные сущности, которые необходимо определить.
- 4) Использование частных ссылок, содержащие ссылки на конкретные сущности (объекты).

По типу решаемой задачи различают следующие задачи:

- интерпретация: процесс определения смысла данных (построение описаний по наблюдаемым данным);
- диагностика: процесс соотнесения объекта с некоторым классом объектов и/или обнаружение неисправностей в системе (отклонение параметров системы от нормативных в технике и в живых организмах);
- мониторинг: непрерывная интерпретация данных в реальном масштабе времени и сигнализация о выходе тех или иных параметров за допустимые пределы;

- прогнозирование: построение планов действий объектов, будущих событий на основе моделей прошлого и настоящего. В прогнозирующих системах часто используют динамические модели, в которых значения параметров «подгоняются» под заданную ситуацию. Выводимые из этих моделей следствия составляют основу для прогнозов с вероятностными оценками;
- планирование: конструирование плана действий объектов способных выполнять некоторые функции, т. е. программы действий. Оно основано на моделях поведения реальных объектов, которые позволяют проводить логический вывод последствий планируемой деятельности;
- проектирование: разработка ранее не существовавшего объекта и подготовка спецификаций на создание объектов с заранее определенными свойствами. Степень новизны может быть разной и определяется видом знаний и методами их обработки;

- обучение: системы обучения выполняют такие функции, как диагностика ошибок, подсказывание правильных решений, аккумулирование знаний о гипотетическом «ученике» и его характерных ошибках, диагностирование слабости в познаниях обучаемых и нахождение соответствующих средств для их ликвидации. Системы обучения способны планировать акт общения с учеником;
- управление: интерпретация, прогноз, планирование, моделирование, оптимизация выработанных решений, мониторинг, т. е. функция системы, поддерживающая определенный режим ее функционирования или управления поведением сложной системы в соответствии с заданными спецификациями;
- отладка, ремонт: выработка рекомендаций по устранению неисправностей;
- поддержка принятия решений — совокупность процедур, обеспечивающих лиц принимающих решения необходимой информацией и рекомендациями для процесса принятия решений (выбор и/или, генерация альтернатив).

- Задачи интерпретации данных, диагностики, поддержки принятия решений относятся к задачам анализа, задачи проектирования, планирования и управления — к задачам синтеза. К комбинированному типу задач относятся обучение, мониторинг и прогнозирование.
- Термин «решение задач» (*problem solving*) употребляется в искусственном интеллекте в ограниченном смысле. Речь идет о хорошо определенных задачах, решаемых на основе поисковых алгоритмов.
- Задача считается хорошо определенной, если для нее имеется возможность задать пространство возможных решений (состояний), а также способ просмотра этого пространства с целью поиска конечного (целевого) состояния, соответствующего решаемой задаче. Поиск конечного состояния задачи заключается в применении к каждому состоянию алгоритмической процедуры с целью проверки, не является ли это состояние решением задачи. Данная процедура продолжается до тех пор, пока не будет найдено решение.
- Примерами хорошо определенных задач являются: доказательство теорем, поиск маршрута на графе, планирование робота в среде с препятствиями и т. д.

Общие способы решения задач

- Человек обычно не решает задачу в той форме, в которой она изначально формулируется. Он стремится представить задачу таким образом, чтобы ему удобно было ее решать. Для этого он выполняет преобразование исходного представления задачи с целью сокращения пространства, в котором необходимо выполнять поиск решения задачи. Этап выбора подходящей формы представления задачи настолько обыден, что мы часто не осознаем его важности. В то же время форма или способ представления задачи в значительной мере определяет успех ее решения.
- При выборе способа представления задачи обычно учитывают два обстоятельства: представление задачи должно достаточно точно моделировать реальность; способы представления должны быть такими, чтобы решателю задач было удобно с ним работать.

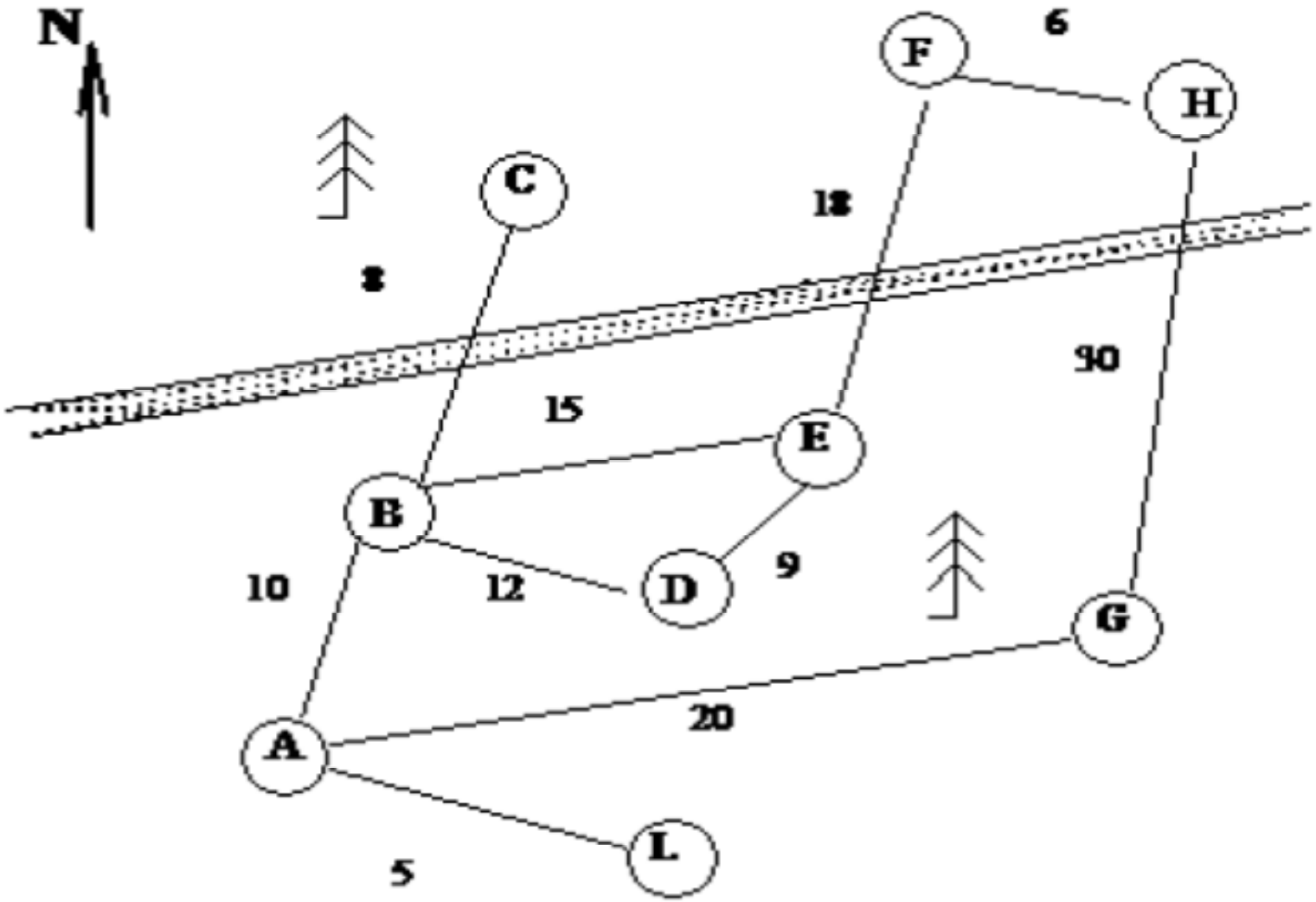
способы представления задач, удобные для их решения на ЭВМ

- представление задач в пространстве состояний;
- представление, сводящее задачу к подзадачам;
- представление задач в виде теорем.

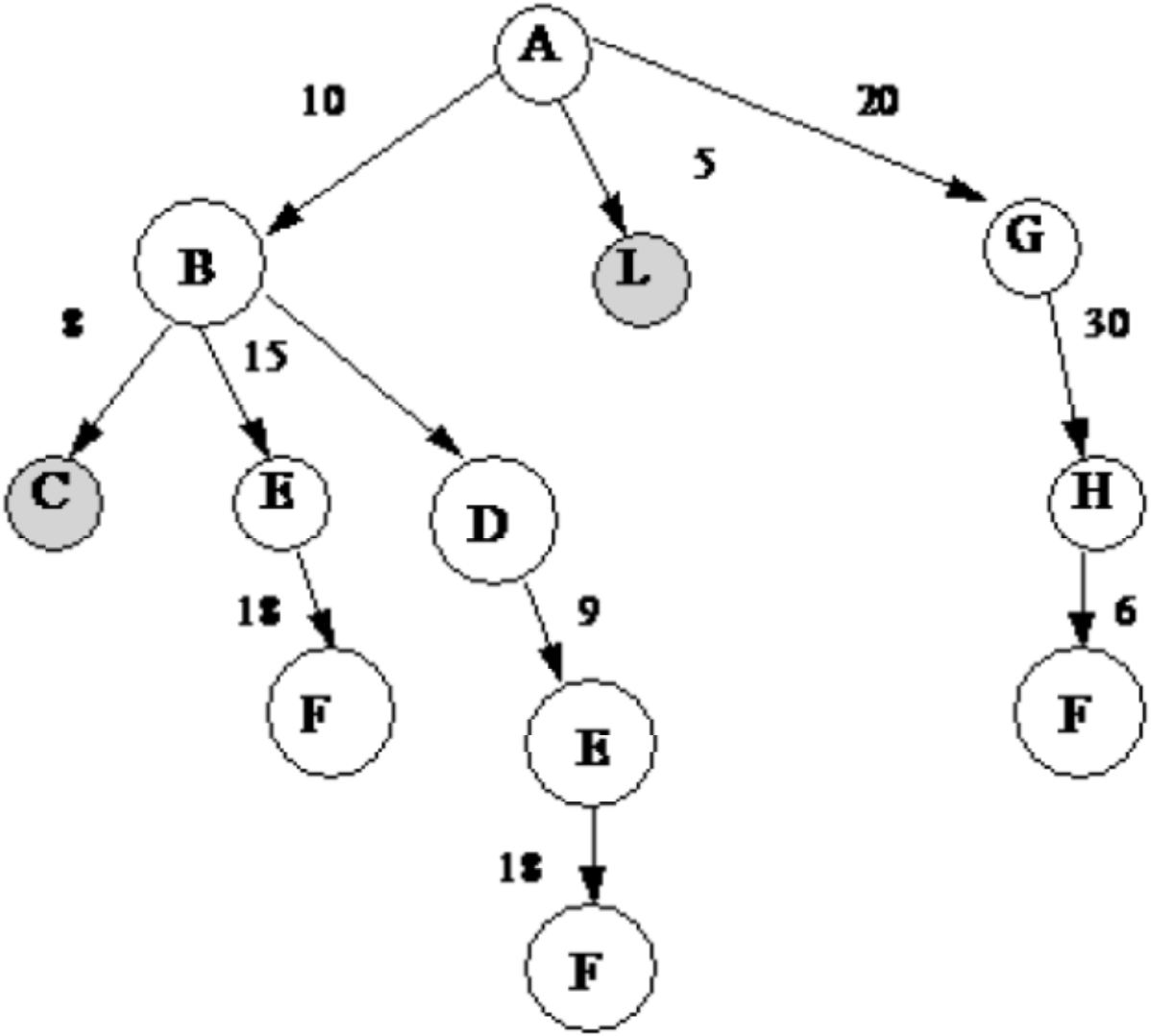
Процесс решения задачи, как правило, включает два этапа: представление задачи и поиск (перебор). Успех решения задачи в значительной мере определяется формой ее представления. Формы представления задачи могут быть различными и зависят как от природы самой задачи, так и от ее решателя. Например, при вычислении интеграла человек стремится путем преобразований представить его так, чтобы воспользоваться его табличными интегралами.

- Процедура поиска решения в пространстве состояний состоит в том, чтобы найти последовательность операторов, которая преобразует начальное состояние в целевое. Решением задачи будет указанная последовательность операторов.
- Деревом называется ориентированный граф, в каждую вершину которого входит только одна дуга, за исключением одной вершины, называемой корнем дерева.
- Таким образом, в дереве каждая вершина, за исключением корня, является концом ровно одной дуги и началом одной или нескольких дуг.
- Существуют два типа структур взаимосвязи подзадач: И-структуры и И-ИЛИ-структуры. В структурах типа И для решения основной задачи требуется решить все подзадачи. В структурах И-ИЛИ подзадачи разбиваются на группы, внутри которых они связаны отношением И, а между группами — отношением ИЛИ.
- Поиск конечного состояния выполняется автоматически на основе, реализованной в системе стратегии поиска, которая обеспечивает возможность выбора и позволяет выполнять шаги от начального состояния к новым состояниям, более или менее близким к цели

- Таким образом, реализованные в таких системах стратегии поиска отыскивают цель, «шагая» от одного состояния системы к другому, обеспечивая при этом распознавание ситуаций, когда они находят цель или попадают в тупик. Как правило, эти системы на промежуточных стадиях вычисляют 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).

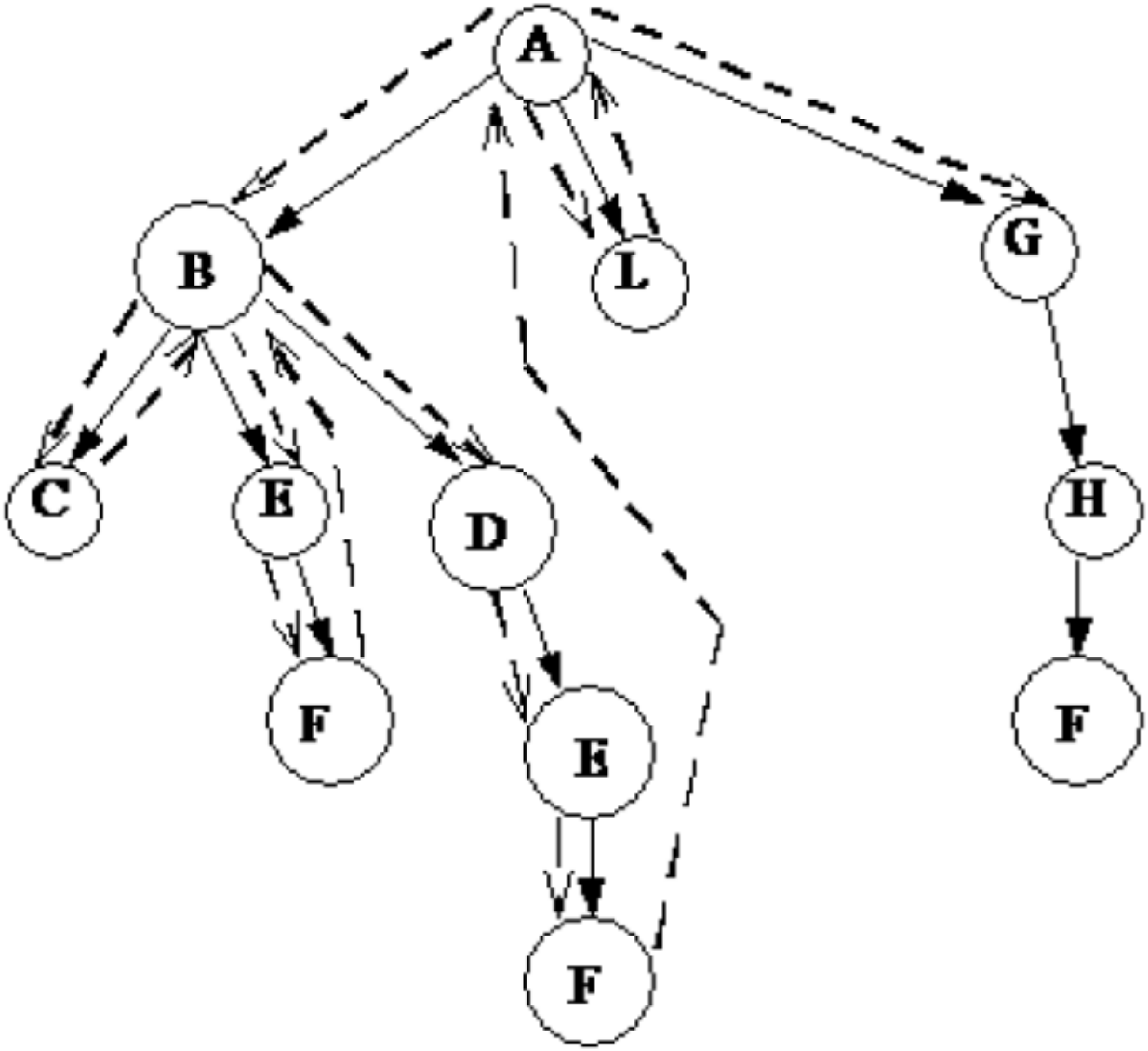

N
F
6
H
C
18
8
90
15
E
B
10
12
D
9
G
20
A
L
5

- 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

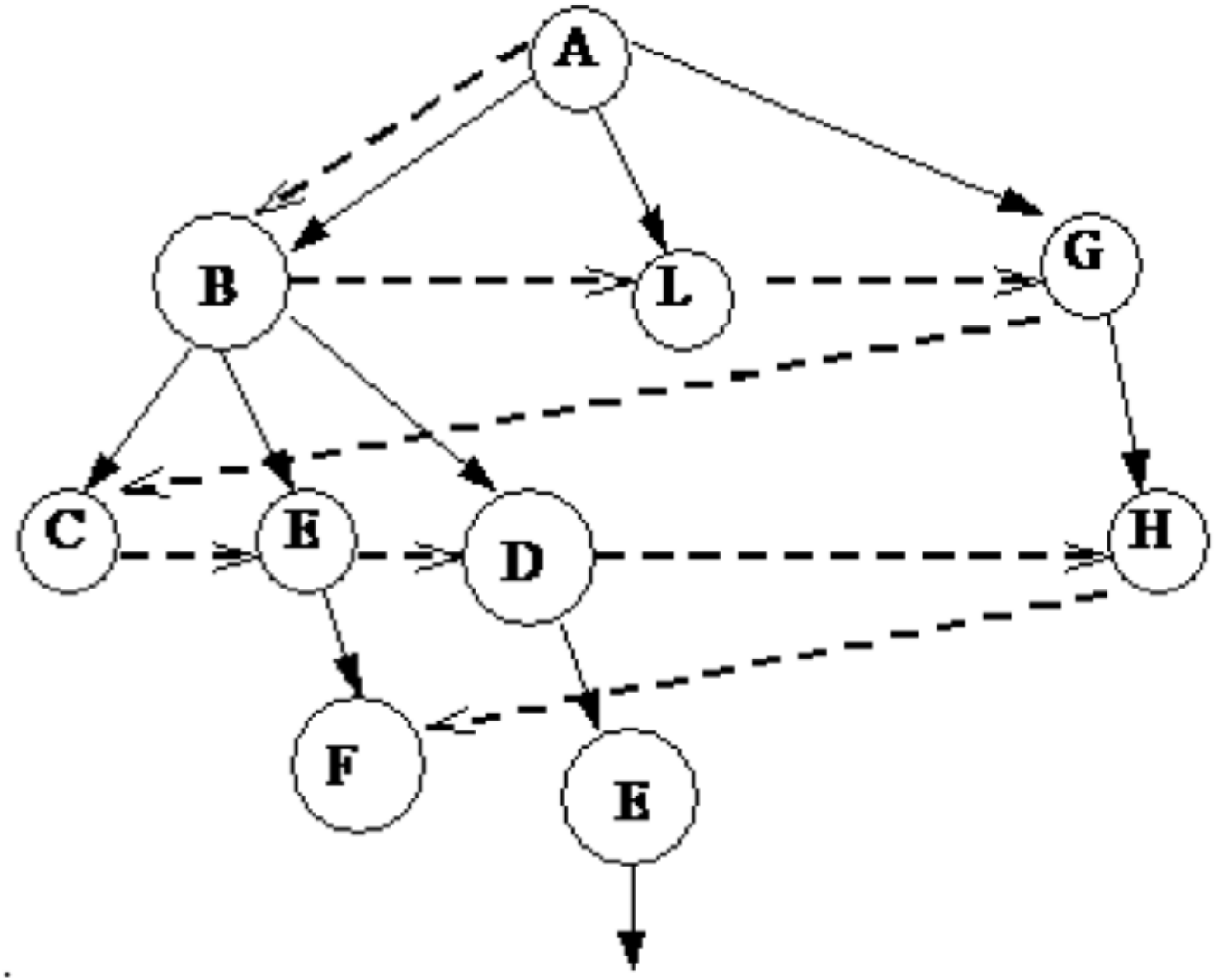

A
10
5
20
B
L
G
8
15
30
C
E
D
H
18
9
6
F
E
F
18
F

- 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».
- Задача компьютера состоит в том, чтобы найти подходящую дорогу из вершины дерева к конечному состоянию (цели) в его нижней части.
- 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 поиска каждого последующего шага применяется простая механическая стратегия, которая имеет две основные разновидности: поиск в глубину и поиск в ширину.

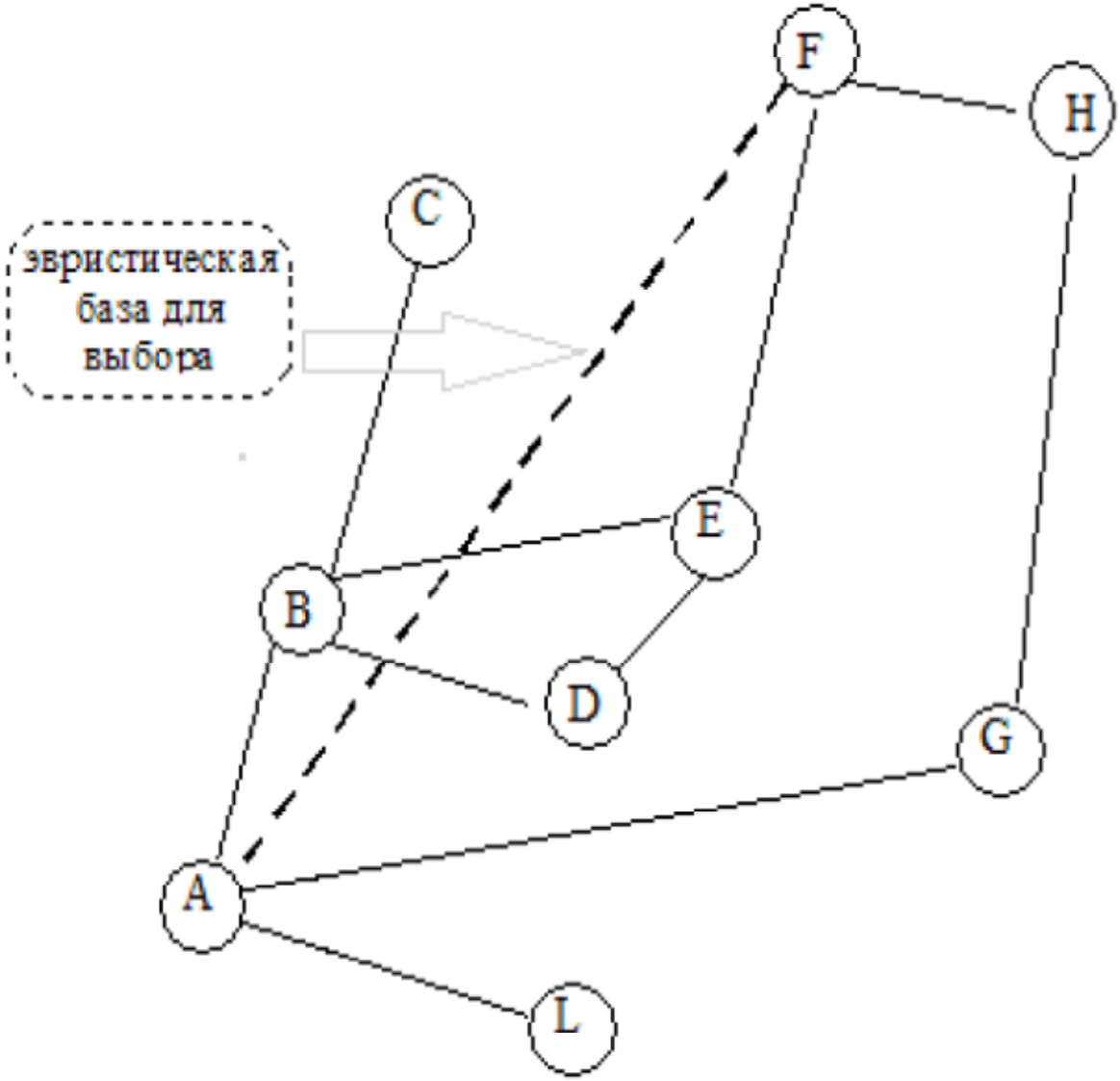
Стратегия поиска в глубину



Стратегия поиска в ширину



Стратегия эвристического поиска



- **Реализованные в CLIPS стратегии поиска**
- CLIPS является одной из оболочек, позволяющих проектировать и разрабатывать интеллектуальные системы. В том числе и тех, что базируются на поиске в пространстве состояний. CLIPS поддерживает семь различных стратегий разрешения конфликтов: стратегия глубины (depth strategy), стратегия ширины (breadth strategy), стратегия упрощения (simplicity strategy), стратегия усложнения (complexity strategy), LEX (LEX strategy), MEA (MEA strategy) и случайная стратегия (random strategy).

- По умолчанию в CLIPS установлена стратегия глубины. Текущая стратегия может быть изменена и установлена командой set-strategy, которая переупорядочит текущий план решения задачи, базируясь на новой стратегии.
- set-strategy depth
- set-strategy breadth

- • Стратегия глубины. Только что активированное правило помещается выше всех правил с таким же приоритетом. Например, допустим, что факт – А активировал правила 1 и 2 и факт Б активировал правило 3 и правило 4, тогда, если факт А добавлен перед фактом Б, в плане решения задачи правила 3 и 4 будут располагаться выше, чем правила 1 и 2. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.
- • Стратегия ширины. Только что активированное правило помещается ниже всех правил с таким же приоритетом. Например, допустим, что факт А активировал правила 1 и 2 и факт Б активировал правила 3 и 4, тогда, если факт А добавлен перед фактом В, в плане решения задачи правила 1 и 2 будут располагаться выше, чем правила 3 и 4. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

- Стратегия упрощения. Между всеми правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или большей определенностью (specificity). Определенность правила вычисляется по числу сопоставлений, которые нужно сделать в левой части правила. Каждое сопоставление с константой или заранее связанной с фактом переменной добавляет к определенности единицу. Каждый вызов функции в левой части правила, являющийся частью условных элементов : , = или test, также добавляет к определенности единицу. Логические функции and, or и not не увеличивают определенность правила, но их аргументы могут это сделать. Вызовы функций, сделанные внутри функций, не увеличивают определенность правила. Например, следующее правило имеет определенность, равную 5.
- (defrule example
- (item ?x ?y ?x)
- (test (and (numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))
- =>)
- Сравнение заранее связанной переменной >x с константой и вызовы функций numberp, < и > добавляют единицу к определенности правила. В итоге получаем определенность, равную 5. Вызовы функций and и + не увеличивают определенность правила.

- • Стратегия усложнения. Между правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или меньшей определенностью.
- • Стратегия LEX. Между правилами с одинаковым приоритетом только что активированные правила размещаются с применением одноименной стратегии, впервые использованной в системе OPSS. Для определения места активированного правила в плане решения задачи используется "новизна" образца, который активировал правило. CLIPS маркирует каждый факт или объект временным тегом для отображения относительной новизны каждого факта или объекта в системе. Образцы, ассоциированные с каждой активацией правила, сортируются по убыванию тегов для определения местоположения правила.
- Активация правила, выполненная более новыми образцами, располагается перед активацией, осуществленной более поздними образцами.

- Для определения порядка размещения двух активаций правил, поодиночке сравниваются отсортированные временные теги для этих двух активаций, начиная с наибольшего временного тега. Сравнение продолжается до тех пор, пока не остается одна активация с наибольшим временным тегом. Эта активация размещается выше всех остальных в плане решения задачи.
- Если активация некоторого правила выполнена большим числом образцов, чем активация другого правила, и все сравниваемые временные теги одинаковы, то активация другого с большим числом временных тегов помещается перед активацией с меньшим. Если две активации имеют одинаковое количество временных тегов и их значения равны, то правило с большей определенностью помещается перед активацией с меньшей. В отличие от системы OPSS, условный элемент not в CLIPS имеет псевдовременной тег, который также используется в данной стратегии разрешения конфликтов. Временной тег условного элемента not всегда меньше, чем временной тег образца.

- В качестве примера рассмотрим следующие шесть активаций правил, приведенные в LEX-порядке (запятая в конце строки активации означает наличие логического элемента not). Учтите, что временные теги фактов не обязательно равны индексу, но если индекс факта больше, то больше и его временной тег. Для данного примера примем, что временные теги равны индексам.
- rule-6: f-1, f-4
- rule-5: f-1, f-2, f-3,
- rule-1: f-1, f-2, f-3
- rule-2: f-3, f-1
- rule-4: f-1, f-2,
- rule-3: f-2, f-1
- Далее показаны те же активации с индексами фактов в том порядке, в котором они сравниваются стратегией LEX.
- rule-6: f-4, f-1
- rule-5: f-3, f-2, f-1,
- rule-1: f-3, f-2, f-1
- rule-2: f-3, f-1
- rule-4: f-2, f-1,
- rule-3: f-2, f-1

- Стратегия MEA. Между правилами с одинаковым приоритетом только что активированные правила размещаются с использованием одноименной стратегии, впервые использованной в системе OPSS. Основное отличие стратегии MEA от LEX в том, что в стратегии MEA не производится сортировка образцов, активировавших правило. Сравниваются только временные теги первых образцов двух активаций. Активация с большим тегом помещается в план решения задачи перед активацией с меньшим. Если обе активации имеют одинаковые временные теги, ассоциированные с первым образцом, то для определения размещения активации в плане решения задачи используется стратегия LEX. Как и в стратегии LEX, условный элемент not имеет псевдовременной тег.

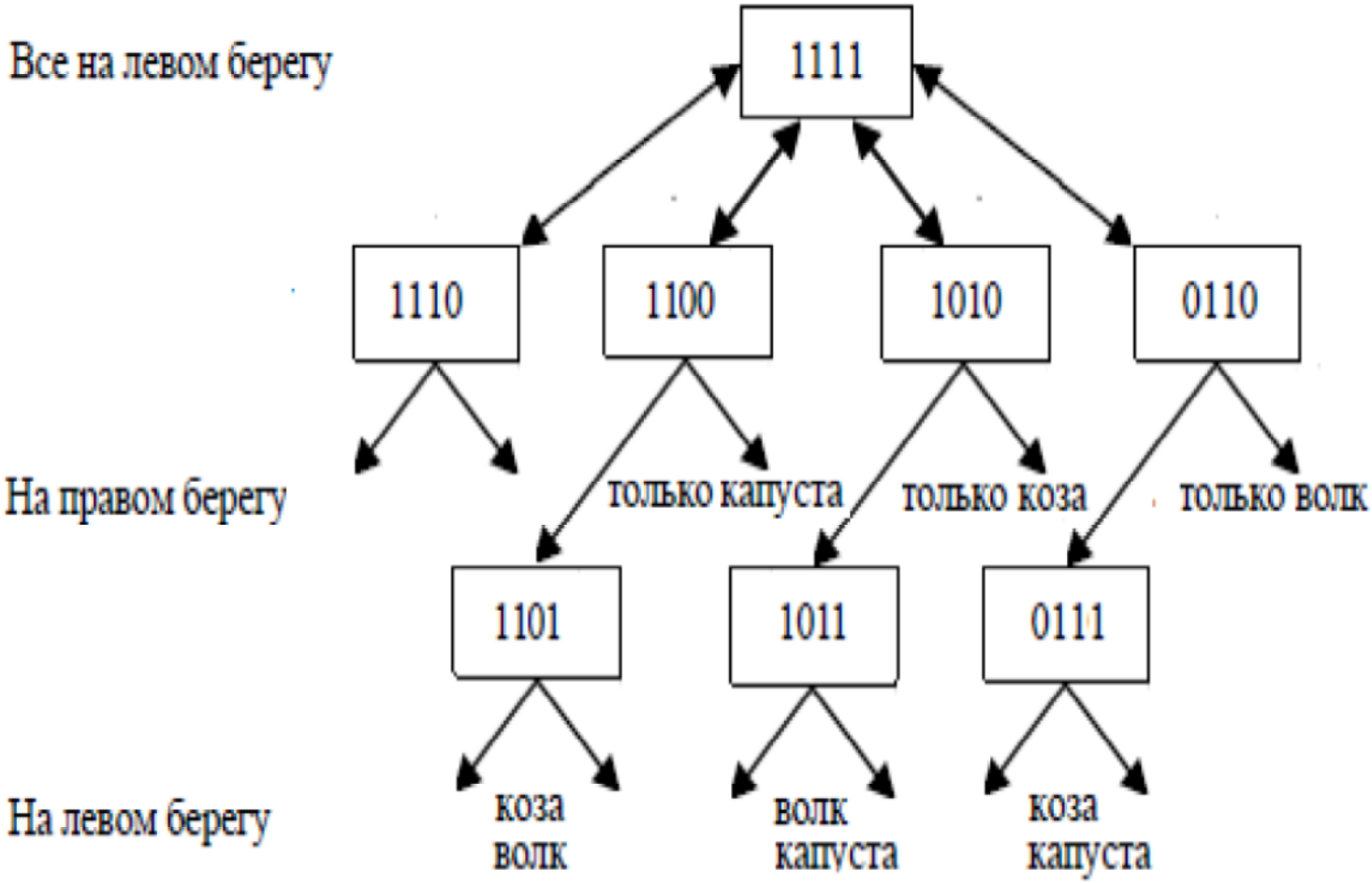
- В качестве примера рассмотрим следующие шесть активаций, приведенные в MEA-порядке (запятая на конце активации означает наличие логического элемента not).
- rule-2: f-3, f-1
- rule-3: f-2, f-1
- rule-6: f-1, f-4
- rule-5: f-1, f-2, f-3,
- rule-1: f-1, f-2, f-3
- rule-4: f-1, f-2,

- • Случайная стратегия. Каждой активации назначается случайное число, которое используется для определения местоположения среди активаций с одинаковым приоритетом. Это случайное число сохраняется при смене стратегий, таким образом, тот же порядок воспроизводится при следующей установке случайной стратегии (среди активаций в плане решения задачи, когда стратегия заменена на исходную).

Волк, коза, капуста

- Фермеру нужно перевезти волка, козу и капусту на другой берег по одному, если нельзя оставлять некоторых из них наедине без фермера

Состояния


Все на левом берегу
1111
1110
1100
1010
0110
На правом берегу
только капуста
только коза
только волк
1101
1011
0111
На левом берегу
коза
волк
волк
капуста
коза
капуста

```
% 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
```

```
% 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
```

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

```
% 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
```

Clips система прямого вывода

```
(deffacts InitFactList
        (usemargin TRUE)
        (defaultcolor black)
        (fact 0 1)
  )
(defrule HelloWorldRule "This is a comment"
(initial-fact) =>
(printout t "HELLO WORLD!" crlf)
)
(defrule inFacts
(usemargin TRUE) =>
(printout t "Hello my fact" crlf)
(assert (activate_my))
)
(defglobal
?*factall* = 5)
(defrule react
(fact ?x ?y) =>
(if (<= ?x ?*factall*) then
(assert (fact (+ ?x 1) (* ?y (+ ?x 1))) ) )
```

```
(deffuction factorial (?a)
(if (or (not (integerp ?a)) (< ?a 0)) then
(printout t "Factorial error! " crlf)
else
(if (= ?a 0) then 1 else
(* ?a (factorial (- ?a 1))))))
```

- Функции работы с составными величинами являются одной из отличительных особенностей языка CLIPS. В их число входят:
- insert$ – добавление новых элементов в составную величину;
- first$ – получение первого элемента составной величины;
- rest$ – получение остатка составной величины;
- length$ – определение числа элементов составной величины;
- delete-member$ – удаление элементов составной величины;
- replace-member$ – замена элементов составной величины.

```
(deffacts fermer
        (sost 0 0 0 0)
        (start 1)
        (start 0)
)
(defrule conflict
         (start ?x)
=>
         (assert(confl ?x  1 1 0))
         (assert(confl  1  ?x 1 0))
         (assert(confl ?x  0 0 1))
         (assert(confl  0  ?x 0 1))
)
(defrule exit
        (sost 1 1 1 1)
     =>         (printout t " End of branch  algor 1 1 1 1" crlf)
)
```

```
(defrule move_wolf
        (sost ?x ?y ?z ?x)
        (not (confl ~?x  ?y ?z ~?x))
        (not (sost ~?x  ?y ?z ~?x))
=>
(printout t "perevezli volka na  c "  ?x ?y ?z ?x " in " (- 1 ?x) ?y ?z (- 1 ?x)   crlf)
        (assert(sost (- 1 ?x) ?y ?z (- 1 ?x)))
)

(defrule move_koza
        (sost ?x ?y ?z ?z)
        (not (confl ?x  ?y ~?z ~?z))
        (not (sost ?x  ?y ~?z ~?z))
=>
(printout t "perevezli kozu  c "  ?x ?y ?z ?z  " in "  ?x ?y (- 1 ?z) (- 1 ?z) crlf)
        (assert(sost  ?x ?y  (- 1 ?z) (- 1 ?z) ))
)
```

```
(defrule move_kapusta
        (sost ?x ?y ?z ?y)
        (not (confl ?x  ~?y ?z ~?y))
        (not (sost ?x  ~?y ?z ~?y))
=>
(printout t "perevezli kapustu  c "  ?x ?y ?z ?y  " in "  ?x (- 1 ?y)  ?z (- 1 ?y) crlf  )
 (assert(sost  ?x (- 1 ?y)  ?z (- 1 ?y) ))

)



(defrule move_fermer
        (sost ?x ?y ?z ?w)
        (not (confl ?x ?y ?z ~?w))
        (not (sost ?x ?y ?z ~?w))
=>
(printout t "pereshel fermer na  bereg  c "  ?x ?y ?z ?w " in " ?x ?y ?z (- 1 ?w)
    crlf)
        (assert(sost  ?x  ?y  ?z (- 1 ?w) ))
)
```

```
CLIPS> (clear)
CLIPS> (load "e2.clp")
Defining deffacts: fermer
Defining defrule: conflict +j
Defining defrule: exit +j
Defining defrule: move_wolf +j+j+j
Defining defrule: move_koza +j+j+j
Defining defrule: move_kapusta +j+j+j
Defining defrule: move_fermer +j+j+j
TRUE
CLIPS> (reset)
CLIPS> (run)
perevezli kozu  c 0000 in 0011
pereshel fermer na  bereg  c 0011 in 0010
perevezli volka na  c 0010 in 1011
perevezli kozu  c 1011 in 1000
perevezli kapustu  c 1000 in 1101
perevezli volka na  c 1101 in 0100
perevezli kozu  c 0100 in 0111
pereshel fermer na  bereg  c 1101 in 1100
perevezli kozu  c 1100 in 1111
 End of branch  algor 1 1 1 1
```

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (sost 0 0 0 0)
f-2     (start 1)
f-3     (start 0)
f-4     (confl 0 1 1 0)
f-5     (confl 1 0 1 0)
f-6     (confl 0 0 0 1)
f-7     (confl 1 1 1 0)
f-8     (confl 1 0 0 1)
f-9     (confl 0 1 0 1)
f-10    (sost 0 0 1 1)
f-11    (sost 0 0 1 0)
f-12    (sost 1 0 1 1)
f-13    (sost 1 0 0 0)
f-14    (sost 1 1 0 1)
f-15    (sost 0 1 0 0)
f-16    (sost 0 1 1 1)
f-17    (sost 1 1 0 0)
f-18    (sost 1 1 1 1)
For a total of 19 facts.
```

Пробуем проследить путь в списке

```
(defrule exit
        (sost 1 1 1 1 $?x)

=>
        (printout t " End of branch  algor 1 1 1 1 " $?x "  "
crlf)

)

(defglobal
?*val* = 0
)
```

```
(defrule move_wolf
        (sost ?x ?y ?z ?x $?s)
        (not (confl ~?x  ?y ?z ~?x))
        (not (exists (sost ~?x  ?y ?z ~?x $?)))
=>

        (bind ?*val*  (+ 1 ?*val*))
        (printout t "perevezli volka na  c "  ?x ?y ?z ?x " in "
(- 1 ?x) ?y ?z (- 1 ?x) "   " $?s ?*val*   crlf)
        (assert(sost (- 1 ?x) ?y ?z (- 1 ?x) $?s ?*val*))
)
```

```
(defrule move_koza
        (sost ?x ?y ?z ?z $?s)
        (not (confl ?x  ?y ~?z ~?z))
        (not (sost ?x  ?y ~?z ~?z $?))
=>
        (bind ?*val*  (+ 1 ?*val*))
        (printout t "perevezli kozu  c "  ?x ?y ?z ?z  " in "  ?x ?y (- 1 ?z) (- 1 ?z) "   " $?s  ?*val*  crlf)
        (assert(sost  ?x ?y  (- 1 ?z) (- 1 ?z) $?s ?*val*))
)
(defrule move_kapusta
        (sost ?x ?y ?z ?y $?s)
        (not (confl ?x  ~?y ?z ~?y))
        (not (sost ?x  ~?y ?z ~?y $?))
=>
(bind ?*val*  (+ 1 ?*val*))
(printout t "perevezli kapustu  c "  ?x ?y ?z ?y  " in "  ?x (- 1 ?y)  ?z (- 1 ?y) "   " $?s   ?*val* crlf  )
(assert(sost  ?x (- 1 ?y)  ?z (- 1 ?y) $?s ?*val*))
)
(defrule move_fermer
        (sost ?x ?y ?z ?w $?s)
        (not (confl ?x ?y ?z ~?w))
        (not (sost ?x ?y ?z ~?w $?))
=>
        (bind ?*val*  (+ 1 ?*val*))
        (printout t "pereshel fermer na  bereg  c "  ?x ?y ?z ?w " in " ?x ?y ?z (- 1 ?w)   "   " $?s   ?*val* crlf)
        (assert(sost  ?x  ?y  ?z (- 1 ?w) $?s ?*val*))
)
```

- (load "e3.clp")
- Defining deffacts: fermer
- Defining defrule: conflict +j
- Defining defrule: exit +j
- Defining defglobal: val
- Defining defrule: move_wolf +j+j+j+j+j+j
- Defining defrule: move_koza +j+j+j
- Defining defrule: move_kapusta +j+j+j
- Defining defrule: move_fermer +j+j+j
- TRUE
- CLIPS> (reset)
- CLIPS> (run)
- perevezli kozu c 0000 in 0011   (0)1
- pereshel fermer na bereg c 0011 in 0010   (0 1)2
- perevezli volka na c 0010 in 1011   (0 1 2)3
- perevezli kozu c 1011 in 1000   (0 1 2 3)4
- perevezli kapustu c 1000 in 1101   (0 1 2 3 4)5
- perevezli volka na c 1101 in 0100   (0 1 2 3 4 5)6
- perevezli kozu c 0100 in 0111   (0 1 2 3 4 5 6)7
- pereshel fermer na bereg c 1101 in 1100   (0 1 2 3 4 5)8
- perevezli kozu c 1100 in 1111   (0 1 2 3 4 5 8)9
- End of branch algor 1 1 1 1 (0 1 2 3 4 5 8 9)

- Стратегия в ширину (другой результат)
- CLIPS> (clear)
- CLIPS> (set-strategy breadth)
- depth
- CLIPS> (load "e3.clp")
- CLIPS> (reset)
- CLIPS> (run)
- pereshel fermer na bereg c 0000 in 0001   (0)1
- perevezli kapustu c 0000 in 0101   (0)2
- perevezli kozu c 0000 in 0011   (0)3
- perevezli volka na c 0000 in 1001   (0)4
- pereshel fermer na bereg c 0101 in 0100   (0 2)5
- pereshel fermer na bereg c 0011 in 0010   (0 3)6
- pereshel fermer na bereg c 1001 in 1000   (0 4)7
- perevezli kozu c 0100 in 0111   (0 2 5)8
- perevezli volka na c 0100 in 1101   (0 2 5)9
- perevezli volka na c 0010 in 1011   (0 3 6)10
- pereshel fermer na bereg c 1101 in 1100   (0 2 5 9)11
- perevezli kozu c 1100 in 1111   (0 2 5 9 11)12
- End of branch algor 1 1 1 1 (0 2 5 9 11 12)
- CLIPS>

- CLIPS> (set-strategy lex)
- breadth
- CLIPS> (reset)
- CLIPS> (run)
- perevezli kozu c 0000 in 0011 (0)1
- pereshel fermer na bereg c 0011 in 0010 (0 1)2
- perevezli volka na c 0010 in 1011 (0 1 2)3
- perevezli kozu c 1011 in 1000 (0 1 2 3)4
- perevezli kapustu c 1000 in 1101 (0 1 2 3 4)5
- perevezli volka na c 1101 in 0100 (0 1 2 3 4 5)6
- perevezli kozu c 0100 in 0111 (0 1 2 3 4 5 6)7
- pereshel fermer na bereg c 1101 in 1100 (0 1 2 3 4 5)8
- perevezli kozu c 1100 in 1111 (0 1 2 3 4 5 8)9
- End of branch algor 1 1 1 1 (0 1 2 3 4 5 8 9)
- CLIPS>