Lesson 7: State Space Problem Solving

What's It All About?

1. Some problems that can readily be solved within the state space framework are presented.
2. The state space problem solving model is introduced.
3. The example problems are represented in terms of the state space paradigm.
4. A Prolog program is written for the "Missionaries and Cannibals" problem.

Example Problems

A Waterjug Problem

You have a sink with an unlimited supply of water at your disposal. You have two unmarked jugs, an M-gallon jug and an N-gallon jug. How can you get exactly X gallons into a particular one of the two jugs, with no water left in the other jug?

An instance of the problem: Given a 3-gallon jug and a 4-gallon jug, how can you get exactly 2 gallons in the 3-gallon jug.

The Towers of Hanoi

A very small version of the "Towers of Hanoi" problem: Three pegs. Three disks – large, medium, small. The disks are place on the pegs subject to the constraint that a larger disk can never appear on top of a smaller disk. A move consists of transferring a disk, the top one, from one peg to another, placing it on top of whatever disks may be present. The task is to transfer all of the pegs from the first peg to the third peg.

The Missionaries and Cannibals Problem

Three missionaries and three cannibals, along with one boat that fits at most two people (and requires at least one person for operation), are on the left bank of a river. The most salient thing about missionaries and cannibals in "cohabitation" is that if ever the cannibals in any one spot (left bank, right bank, on the boat) outnumber the missionaries, the outnumbered missionaries will be consumed – eaten! The goal of this problem is to get all six individuals safely across the river from the left bank to the right bank.

The Eight Queens Problem

Place 8 queens on a chessboard in such a way that no queen can attack any other queen.

State Space Problem Solving

State space problem solving is a standard AI methodology that can be used to solve a variety of problems.

The basic idea is to represent a world as a collection of objects which are defined in terms of properties and their values. For a given world, this can generally be done in a number of ways.

A **state** of the world consists of the collection of objects together with the values of their properties.

Operators, called "state space operators", serve to transform one state of the world into another state of the world.

A **state space solution** consists of a **sequence of operators** that transforms the world from some **initial state** into a **goal state**.

How do you find a state space solution to a state space problem? You perform **state space search**.

State Space Problem Description

A **state space problem description** is a triple consisting of:

- I, a set of possible initial states
- G, a set of goal states
- O, a set of state space operators mapping S to S

where S is a state space which "fits" the problem.

State Space Problem Solution

A **state space problem solution** is a triple consisting of:

- i, one of the possible initial states
- g, one of the possible goal states
- x, a finite sequence of operators that transforms the initial state into the goal state

in the context of a given state space.

Example: State Space Problem Description for Towers of Hanoi

Represent the three disks by symbols L (large) and M (medium) and S (small). Represent the three pegs as lists, imagining the disks arranged from left to right in increasing order of size.

Then ...

- I = {`((S M L) () () )`}
- G = {`(() () (S M L) )`}
- O = {`M12, M13, M21, M23, M31, M32`}, where
  - M12 - move a disk from peg 1 to peg 2
  - M13 - move a disk from peg 1 to peg 3
  - M21 - move a disk from peg 2 to peg 1
  - M23 - move a disk from peg 2 to peg 3
  - M31 - move a disk from peg 3 to peg 1
  - M32 - move a disk from peg 3 to peg 2

One possible state space solution:

M13 ⇒ M12 ⇒ M32 ⇒ M13 ⇒ M21 ⇒ M23 ⇒ M13

How was this solution obtained? **By means of state space search!**

A Word on State Space Search

State space search is a process in which you begin by considering an initial state to be the root of a tree, called the **state space tree**. You grow the state space tree (you might prefer to maintain a graph) by making use of applicable **state space operators** to determine the children of a node. When, in the process of growing the tree, a goal node turns up, a solution is at hand. The solution can be observed by considering the sequence of state space operators that lead from the initial state to the goal state.

There is a great deal to be said about state space search, which is generally discussed in a first AI course. For example, there are varieties of search, ranging from "blind" searches to "heuristic" searches. The two most basic blind searches are called **breadth first search** and **depth first search**. One of the most used heuristic searches is called **best first search**. There are also techniques for pruning the search tree in order to expedite search.

Example: State Space Problem Description for Missionaries and Cannibals

Represent a state by a list of five elements, such that:

- The first element represents the number of missionaries on the left bank
- The second element represents the number of canibals on the left bank
- The third element, `l` or `r`, represents bank on which the missionaries and cannibals happen to be
- The fourth element represents the number of missionaries on the right bank
- The fifth element represents the number of cannibals on the right bank

Then ...

- I = {`(3,3,l,0,0)`}
- G = {`(0,0,r,3,3)`}
- O = {`mlr, mmlr, mclr, cclr, clr, mrl, mmrl, mcrl, ccrl, crl`}, where
  - mlr - transfer one missionary from the left bank to the right bank
  - mmlr - transfer two missionaries from the left bank to the right bank
  - mclr - transfer one missionary and one cannibal from the left bank to the right bank
  - cclr - transfer two cannibals from the left bank to the right bank
  - clr - transfer one cannibal from the left bank to the right bank
  - mrl - transfer one missionary from the left bank to the right bank
  - mmrl - transfer two missionaries from the left bank to the right bank
  - mcrl - transfer one missionary and one cannibal from the left bank to the right bank
  - ccrl - transfer two cannibals from the left bank to the right bank
  - crl - transfer one cannibal from the left bank to the right bank

State Space Program for Missionaries and Cannibals

The Move Making Predicate

```
% ----------------------------------------------------------------------
% --- make_move(S,T,SSO) :: Make a move from state S to state T by SSO

make_move([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA],mlr) :-
  mlr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]).
make_move([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA],mmlr) :-
  mmlr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]).
```

```
make_move([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA],mclr) :-
  mclr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]).
make_move([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA],cclr) :-
  cclr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]).
make_move([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA],clr) :-
  clr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]).
make_move([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA],mrl) :-
  mrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]).
make_move([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA],mmrl) :-
  mmrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]).
make_move([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA],mcrl) :-
  mcrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]).
make_move([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA],ccrl) :-
  ccrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]).
make_move([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA],crl) :-
  crl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]).

mlr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]) :-
  MLB > 0,
  MLA is MLB - 1,
  CLA = CLB,
  MRA is MRB + 1,
  CRA = CRB.

mmlr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]) :-
  MLB > 1,
  MLA is MLB - 2,
  CLA = CLB,
  MRA is MRB + 2,
  CRA = CRB.

mclr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]) :-
  MLB > 0, CLB > 0,
  MLA is MLB - 1,
  CLA is CLB - 1,
  MRA is MRB + 1,
  CRA is CRB + 1.

cclr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]) :-
  CLB > 1,
  MLA = MLB,
  CLA is CLB - 2,
  MRA = MRB,
  CRA is CRB + 2.

clr([MLB,CLB,l,MRB,CRB],[MLA,CLA,r,MRA,CRA]) :-
  CLB > 0,
  MLA = MLB,
  CLA is CLB - 1,
  MRA = MRB,
  CRA is CRB + 1.

mrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]) :-
  MRB > 0,
```

```
  MLA is MLB + 1,
  CLA = CLB,
  MRA is MRB - 1,
  CRA = CRB.

mmrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]) :-
  MRB > 1,
  MLA is MLB + 2,
  CLA = CLB,
  MRA is MRB - 2,
  CRA = CRB.

mcrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]) :-
  MRB > 0, CRB > 0,
  MLA is MLB + 1,
  CLA is CLB + 1,
  MRA is MRB - 1,
  CRA is CRB - 1.

ccrl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]) :-
  CRB > 1,
  MLA = MLB,
  CLA is CLB + 2,
  MRA = MRB,
  CRA is CRB - 2.

crl([MLB,CLB,r,MRB,CRB],[MLA,CLA,l,MRA,CRA]) :-
  CRB > 0,
  MLA = MLB,
  CLA is CLB + 1,
  MRA = MRB,
  CRA is CRB - 1.
```

The Feast State Predicate

```
% ----------------------------------------------------------------------
% --- feast_state(S) :: S is a state where a missionary is in peril

feast_state([MLB,CLB,_,_,_]) :-
  CLB > MLB, MLB > 0.
feast_state([_,_,_,MRB,CRB]) :-
  CRB > MRB, MRB > 0.
```

The Solver

```
% ----------------------------------------------------------------------
% --- solve(Start,Solution) :: succeeds if a Solution represents a path
% --- from the Start=start to finish.

solve :-
  extend_path([[3,3,1,0,0]],[],Solution),
  write_solution(Solution).

extend_path(PathSoFar,SolutionSoFar,Solution) :-
  PathSoFar = [[0,0,r,3,3]|_],
  Solution = SolutionSoFar.
extend_path(PathSoFar,SolutionSoFar,Solution) :-
  PathSoFar = [CurrentState|_],
  make_move(CurrentState,NextState,Move),
  not(member(NextState,PathSoFar)),
  not(feast_state(NextState)),
  Path = [NextState|PathSoFar],
  Soln = [Move|SolutionSoFar],
  extend_path(Path,Soln,Solution).
```

The Solver - long form

```
% ----------------------------------------------------------------------
% --- solve(Start,Solution) :: succeeds if a Solution represents a path
% --- from the Start=start to finish.

solve :-
  extend_path([[3,3,1,0,0]],[],Solution),
  write_solution(Solution).

extend_path(PathSoFar,SolutionSoFar,Solution) :-
  PathSoFar = [[0,0,r,3,3]|_],
  show('PathSoFar',PathSoFar),
  show('SolutionSoFar',SolutionSoFar),
  Solution = SolutionSoFar.
extend_path(PathSoFar,SolutionSoFar,Solution) :-
  show('PathSoFar',PathSoFar),
  show('SoluSoFar',SolutionSoFar),
  PathSoFar = [CurrentState|_],
  show('CurrState',CurrentState),
  make_move(CurrentState,NextState,Move),
  show('NextState',NextState),
  not(member(NextState,PathSoFar)),
  not(feast_state(NextState)),
  Path = [NextState|PathSoFar],
```

```
  Soln = [Move|SolutionSoFar],
  extend_path(Path,Soln,Solution).
```

Predicate to write the solution

```
write_solution(S) :-
  nl, write('Solution ...'), nl, nl,
  write_the_solution(S),nl.

write_the_solution([]).
write_the_solution([H|T]) :-
  write_the_solution(T),
  elaborate(H,E),
  write(E),nl.

elaborate(mlr,Elaboration) :-
  Elaboration = 'Transfer a missionary \n  from the left bank to the right bank.'.
elaborate(mmlr,Elaboration) :-
  Elaboration = 'Transfer two missionaries \n  from the left bank to the right bank.'.
elaborate(mclr,Elaboration) :-
  Elaboration = 'Transfer a missionary and a cannibal \n  from the left bank to the right bank.'.
elaborate(cclr,Elaboration) :-
  Elaboration = 'Transfer two cannibals \n  from the left bank to the right bank.'.
elaborate(clr,Elaboration) :-
  Elaboration = 'Transfer a cannibal \n  from the left bank to the right bank.'.
elaborate(mrl,Elaboration) :-
  Elaboration = 'Transfer a missionary \n  from the right bank to the left bank.'.
elaborate(mmrl,Elaboration) :-
  Elaboration = 'Transfer two missionaries \n  from the right bank to the left bank.'.
elaborate(mcrl,Elaboration) :-
  Elaboration = 'Transfer a missionary and a cannibal \n  from the right bank to the left bank.'.
elaborate(ccrl,Elaboration) :-
  Elaboration = 'Transfer two cannibals \n  from the right bank to the left bank.'.
elaborate(crl,Elaboration) :-
  Elaboration = 'Transfer a cannibal \n  from the right bank to the left bank.'.
```

Demo - Short Form

```
bash-3.2$ swipl
<<redacted>>

?- consult('mc.pro').
%  inspector.pro compiled 0.00 sec, 5 clauses
% mc.pro compiled 0.00 sec, 63 clauses
true.
```

```
?- solve.

Solution ...

Transfer a missionary and a cannibal
  from the left bank to the right bank.
Transfer a missionary
  from the right bank to the left bank.
Transfer two cannibals
  from the left bank to the right bank.
Transfer a cannibal
  from the right bank to the left bank.
Transfer two missionaries
  from the left bank to the right bank.
Transfer a missionary and a cannibal
  from the right bank to the left bank.
Transfer two missionaries
  from the left bank to the right bank.
Transfer a cannibal
  from the right bank to the left bank.
Transfer two cannibals
  from the left bank to the right bank.
Transfer a missionary
  from the right bank to the left bank.
Transfer a missionary and a cannibal
  from the left bank to the right bank.

true

?-
```

Demo - Color Coded Long Form

```
bash-3.2$ swipl
<<redacted>>

?- consult('mc.pro').
%  inspector.pro compiled 0.00 sec, 5 clauses
% mc.pro compiled 0.00 sec, 63 clauses
true.


?- solve.

PathSoFar = [[3,3,1,0,0]]
SolnSoFar = []
CurrState = [3,3,1,0,0]
NextState = [2,3,r,1,0]
NextState = [1,3,r,2,0]
NextState = [2,2,r,1,1]
PathSoFar = [[2,2,r,1,1],[3,3,1,0,0]]
```

```
SolnSoFar = [mclr]
CurrState = [2,2,r,1,1]
NextState = [3,2,l,0,1]
PathSoFar = [[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mrl,mclr]
CurrState = [3,2,l,0,1]
NextState = [2,2,r,1,1]
NextState = [1,2,r,2,1]
NextState = [2,1,r,1,2]
NextState = [3,0,r,0,3]
PathSoFar = [[3,0,r,0,3],[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [cclr,mrl,mclr]
CurrState = [3,0,r,0,3]
NextState = [3,2,l,0,1]
NextState = [3,1,l,0,2]
PathSoFar = [[3,1,l,0,2],[3,0,r,0,3],[3,2,l,0,1],[2,2,r,1,1],
[3,3,l,0,0]]
SolnSoFar = [crl,cclr,mrl,mclr]
CurrState = [3,1,l,0,2]
NextState = [2,1,r,1,2]
NextState = [1,1,r,2,2]
PathSoFar = [[1,1,r,2,2],[3,1,l,0,2],[3,0,r,0,3],[3,2,l,0,1],
[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mmlr,crl,cclr,mrl,mclr]
CurrState = [1,1,r,2,2]
NextState = [2,1,l,1,2]
NextState = [3,1,l,0,2]
NextState = [2,2,l,1,1]
PathSoFar = [[2,2,l,1,1],[1,1,r,2,2],[3,1,l,0,2],[3,0,r,0,3],
[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mcrl,mmlr,crl,cclr,mrl,mclr]
CurrState = [2,2,l,1,1]
NextState = [1,2,r,2,1]
NextState = [0,2,r,3,1]
PathSoFar = [[0,2,r,3,1],[2,2,l,1,1],[1,1,r,2,2],[3,1,l,0,2],
[3,0,r,0,3],[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mmlr,mcrl,mmlr,crl,cclr,mrl,mclr]
CurrState = [0,2,r,3,1]
NextState = [1,2,l,2,1]
NextState = [2,2,l,1,1]
NextState = [1,3,l,2,0]
NextState = [0,3,l,3,0]
PathSoFar = [[0,3,l,3,0],[0,2,r,3,1],[2,2,l,1,1],[1,1,r,2,2],
[3,1,l,0,2],[3,0,r,0,3],[3,2,l,0,1],[2,2,r,1,1],
[3,3,l,0,0]]
SolnSoFar = [crl,mmlr,mcrl,mmlr,crl,cclr,mrl,mclr]
CurrState = [0,3,l,3,0]
NextState = [0,1,r,3,2]
PathSoFar = [[0,1,r,3,2],[0,3,l,3,0],[0,2,r,3,1],[2,2,l,1,1],
[1,1,r,2,2],[3,1,l,0,2],[3,0,r,0,3],[3,2,l,0,1],
[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [cclr,crl,mmlr,mcrl,mmlr,crl,cclr,mrl,mclr]
CurrState = [0,1,r,3,2]
NextState = [1,1,l,2,2]
```

```
PathSoFar = [[1,1,l,2,2],[0,1,r,3,2],[0,3,l,3,0],[0,2,r,3,1],
[2,2,l,1,1],[1,1,r,2,2],[3,1,l,0,2],[3,0,r,0,3],
[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mrl,cclr,crl,mmlr,mcrl,mmlr,crl,cclr,mrl,mclr]
CurrState = [1,1,l,2,2]
NextState = [0,1,r,3,2]
NextState = [0,0,r,3,3]
PathSoFar = [[0,0,r,3,3],[1,1,l,2,2],[0,1,r,3,2],[0,3,l,3,0],
[0,2,r,3,1],[2,2,l,1,1],[1,1,r,2,2],[3,1,l,0,2],
[3,0,r,0,3],[3,2,l,0,1],[2,2,r,1,1],[3,3,l,0,0]]
SolnSoFar = [mclr,mrl,cclr,crl,mmlr,mcrl,mmlr,crl,cclr,mrl,mclr]


Solution ...

Transfer a missionary and a cannibal
  from the left bank to the right bank.
Transfer a missionary
  from the right bank to the left bank.
Transfer two cannibals
  from the left bank to the right bank.
Transfer a cannibal
  from the right bank to the left bank.
Transfer two missionaries
  from the left bank to the right bank.
Transfer a missionary and a cannibal
  from the right bank to the left bank.
Transfer two missionaries
  from the left bank to the right bank.
Transfer a cannibal
  from the right bank to the left bank.
Transfer two cannibals
  from the left bank to the right bank.
Transfer a missionary
  from the right bank to the left bank.
Transfer a missionary and a cannibal
  from the left bank to the right bank.

true

?-
```

Unit Testing

Code: Two Unit Tests

```
test__mlr :-
  write('\n>>> Testing the mlr predicate ...'),nl,
  test__mlr_1,
  test__mlr_2.

test__mlr_1 :-
  write('Testing:  mlr\n'),
  Before = [3,3,1,0,0],
  show('Before',Before),
  mlr(Before, After),
  show('After ',After).

test__mlr_2 :-
  write('Testing:  mlr\n'),
  Before = [0,3,1,3,0],
  show('Before',Before),
  mlr(Before, After),
  show('After ',After).
test__mlr_2 :-
  write('Cannot do it: No missionary on the left bank.\n').

test__feast_state :-
  write('\n>>> Testing the feast_state predicate ...'),nl,
  test__fs([2,3,1,1,0]),
  test__fs([0,3,1,3,0]),
  test__fs([2,1,r,1,2]),
  test__fs([3,1,r,0,2]).

test__fs(State) :-
  feast_state(State),
  write(State), write(' is a feast state'), nl.
test__fs(State) :-
  write(State), write(' is not a feast state'), nl.

test :-
  test__mlr,
  test__feast_state.
```

Demo: Running the Unit Tests

```
?- test.

>>> Testing the mlr predicate ...
Testing:  mlr
Before = [3,3,l,0,0]
After  = [2,3,r,1,0]
Testing:  mlr
Before = [0,3,l,3,0]
Cannot do it: No missionary on the left bank.

>>> Testing the feast_state predicate ...
[2,3,l,1,0] is a feast state
[0,3,l,3,0] is not a feast state
[2,1,r,1,2] is a feast state
[3,1,r,0,2] is not a feast state
true

?-
```