Report 83-26
Stanford -- KSL

The MRS Casebook.
Michael R.Genesereth, editor,
May 1983

Stanford Heuristic Programming Project
Memo HPP-83- 26

May 1983
Edition 2

The MRS Casebook

edited by

Michael R. Genesereth

Department of Computer Science
School of Humanities and Sciences
Stanford University

Prolog

MRS is a knowledge representation system intended for use by AI researchers in building expert systems. It offers a diverse repertory of commands for asserting and retrieving information, with various representations (e.g. property lists, propositional representation), various inference techniques (e.g. backward chaining, forward chaining, and resolution, all with optional caching and truth maintenance), and various search strategies (e.g. depth-first, breadth-first, and best-first search). The initial system includes a vocabulary of concepts and facts about logic, sets, mappings, arithmetic, and procedures.

What differentiates MRS from many other knowledge representation systems is its ability to reason about and control its own activity. In MRS the system is treated as a domain in its own right. One can write sentences about subroutines and other sentences and allow the system to reason with them, just as it reasons about geology or medicine. In practice, MRS uses this "meta-level" information in deciding how to satisfy its users' goals. Thus, one can switch representations or inference methods by changing these sentences, and one can easily implement a variety of different control schemes.

This document is a book MRS examples. Each case begins with a brief description of the task in English. Next comes a dictionary of terms and the knowledge base. Each example ends with some sample output.

The Case of Kinship Relations

1. Description

The goal of this problem is a system that can answer questions about kinship relations. The initial data base consists of father and mother relations only, but the system must be able to answer questions about other types of relations, such as cousinhood and ancestry.

In the solution given here, all father and mother relations are represented as atomic propositions. In addition there are rules relating other forms of kinship to fatherhood and motherhood. Questions are asked using TRUEP and TRUEPS, and MRS uses depth-first backward chaining to deduce the answers.

2. Vocabulary

(father <x> <y>) means that <x> is the father of <y>.
(mother <x> <y>) means that <x> is the mother of <y>.
(male <y>) means that <y> is male.
(female <x>) means that <x> is female.
(parent <x> <y>) means that <x> is the parent of <y>.
(sibling <x> <y>) means that <x> is the sibling of <y>.
(brother <x> <y>) means that <x> is the brother of <y>.
(sister <x> <y>) means that <x> is the sister of <y>.
(uncle <x> <y>) means that <x> is the uncle of <y>.
(aunt <x> <y>) means that <x> is the aunt of <y>.
(gfather <x> <y>) means that <x> is the grandfather of <y>.
(gmother <x> <y>) means that <x> is the grandmother of <y>.
(ancestor <x> <y>) means that <x> is the ancestor of <y>.
(cousin <x> <y>) means that <x> is the cousin of <y>.

3. Knowledge Base

```
(if (father $x $y) (parent $x $y))
(if (mother $x $y) (parent $x $y))
(if (and (parent $x $z) (parent $x $y) (not (= $y $z))) (sibling $y $z))
(if (and (sibling $x $y) (male $x)) (brother $x $y))
(if (and (sibling $x $y) (female $x)) (sister $x $y))
(if (and (parent $y $z) (brother $x $y)) (uncle $x $z))
(if (and (parent $y $z) (sister $x $y)) (aunt $x $z))
(if (and (parent $y $z) (father $x $y)) (gfather $x $z))
(if (and (parent $y $z) (mother $x $y)) (gmother $x $z))
(if (parent $x $y) (ancestor $x $y))
(if (and (parent $x $y) (ancestor $y $z)) (ancestor $x $z))
(if (and (parent $u $x) (parent $v $y) (sibling $u $v)) (cousin $x $y))
(if (father $x $y) (male $x))
(if (mother $x $y) (female $x))

(father arthur bertram)
```

```
(father arthur bailey)

(father bertram cornish)
(father bertram carey)

(mother beatrice cornish)
(mother beatrice carey)

(father bailey carleton)
(father bailey cassandra)

(mother bessie carleton)
(mother bessie cassandra)

(male cornish)
(male carey)
(male carleton)
(female cassandra)
```

4. Sample Output

```
=> (trueps '(gfather arthur $z))
   (((($z . cornish) (t . t))
     (($z . carey) (t . t))
     (($z . carleton) (t . t))
     (($z . cassandra) (t . t)))

=> (trueps '(cousin $x cassandra))
   (((($x . cornish) (t . t))
     (($x . carey) (t . t)))
```

The Case of the Primate Taxonomy

1. Description

The goal of this problem is a system that can answer questions about the taxonomy of primates. Given only information about an object's genus, the system should be able to determine the object's family, suborder, and order. In particular, it should be able to determine whether the object is a primate.

Because of the forward branchiness of the search space for the knowledge base given below, the system is set up to forward chain. The trace in the output section shows the various conclusions the system draws from the input data.

2. Vocabulary

`(mem <x> <y>)` states that `<x>` is a member of the set `<y>`.
`(subset <x> <y>)` states thats the set of objects `<x>` is a subset of `<y>`.

3. Knowledge Base

```
::: Forward Chaining

(toassert &p fc)

::: Definition of subset

(if (subset $y $z)
    (if (mem $x $y) (mem $x $z)))

::: Genera and Families

(subset common-tree-shrew tupaiidae)
(subset smooth-tailed-tree-shrew tupaiidae)
(subset philippine-tree-shrew tupaiidae)

(subset pen-tailed-shrew ptilocercinae)

(subset common-lemur lemuridae)
(subset gentle-lemur lemuridae)
(subset sportive-lemur lemuridae)
(subset mouse-lemur lemuridae)
(subset dwarf-lemur lemuridae)

(subset indris indridae)
(subset avahi indridae)
(subset sifaka indridae)

(subset aye-aye daubentoniidae)

(subset slender-loris lorisidae)
(subset slow-loris lorisidae)
```

```
(subset angwantibo lorisidae)
(subset potto lorisidae)

(subset bush-baby galagidae)

(subset tarsier tarsiidae)

(subset plumed-and-pygmy-marmosets callithricidae)
(subset tanarin callithricidae)

(subset goeldis-marmoset cebidae)
(subset dourocouli cebidae)
(subset titi cebidae)
(subset saki-pithecia cebidae)
(subset saki-chiropotes cebidae)
(subset howler cebidae)
(subset capuchin cebidae)
(subset squirrel-monkey cebidae)
(subset spider-monkey cebidae)
(subset woolly-spider-monkey cebidae)
(subset woolly-monkey cebidae)

(subset macaque cercopithecidae)
(subset black-ape cercopithecidae)
(subset mangabey cercopithecidae)
(subset baboon-drill cercopithecidae)
(subset gelada cercopithecidae)
(subset guenon cercopithecidae)
(subset patas-monkey cercopithecidae)
(subset common-langur cercopithecidae)
(subset douc-langur cercopithecidae)
(subset snub-nosed-langur cercopithecidae)
(subset pagi-langur cercopithecidae)
(subset proboscis-monkey cercopithecidae)
(subset gueraza cercopithecidae)

(subset gibbon hylobatidae)
(subset siamang hylobatidae)

(subset orangutan pongidae)
(subset chimpanzee pongidae)
(subset gorilla pongidae)

(subset man hominidae)

;;; Families and Suborders

(subset tupaiidae prosimii)
(subset lemuridae prosimii)
(subset indridae prosimii)
(subset daubentoniidae prosimii)
(subset lorisidae prosimii)
(subset galagidae prosimii)
(subset tarsiidae prosimii)

(subset callithricidae anthropoidea)
(subset cebidae anthropoidea)
(subset cercopithecidae anthropoidea)
(subset hylobatidae anthropoidea)
```

```
(subset pongidae anthropoidea)
(subset hominidae anthropoidea)

;;; Suborders and Orders

(subset prosimii primates)
(subset anthropoidea primates)
```

4. Sample Output

```
=> (tracetask '&k)
   DONE

=> (assert '(mem mrg man))

Executing FCDISP on
  (MEM MRG MAN)

Executing FCDISP on
  (MEM MRG HOMINIDAE)

Executing FCDISP on
  (MEM MRG ANTHROPOIDEA)

Executing FCDISP on
  (MEM MRG PRIMATES)

   P356

=> (lookup '(mem mrg primates))
   ((T . T))
```

The Case of Digital Circuit Simulation

1. Problem Description

The goal of this problem is a system that can simulate the behavior of digital circuits, i.e. it should be able to answer questions about the outputs of a circuit given only information about its inputs, its structure, and the behavior of its parts.

A simple but interesting example is that of a full-adder. A full-adder is a circuit that computes one column of a binary sum. It takes three inputs, viz. the two bits to be added and the carry bit from the previous column, and produces two outputs, viz. a sum bit and a carry bit to the next column. For example, three full-adders could be strung together to produce the binary sum shown below.

```
  101
  111
 ----
 1100
```

The structure for one particular full-adder (F1) is shown below. It consists of two "xor" gates (X1 and X2), two "and" gates (A1 and A2), and a single "or" gate (O1). The output of an "xor" gate is "on" if and only if its inputs are different. The output of an "and" gate is "on" if and only if both of its inputs are "on". The output of an "or" gate is "off" if and only if both its inputs are "off".

In the solution presented here, the circuit's input andstructure are specified by appropriate atomic propositions. The behavior of each part is specified as a set of rules relating the values of its outputs to the values of its inputs. One alternative is for the user to ask questions using truep and trueps, and MRS computes the results using depth-first backward chaining. Another alternative is to set up MRS to forward chain on stated values and cache its results. Due to the branchiness of the search space, this alternative is somewhat more efficient than the previous alternative. Both alternatives are illustrated in the sample output below.

2. Vocabulary

`(type <x> <y>)` means that `<x>` is a part of type `<y>`.
`(conn <x> <y>)` means that port `<x>` is connected to port `<y>`.
`(in <i> <x>)` designates the `<i>`th input of device `<x>`.
`(out <i> <x>)` designates the `<i>`th output of device `<x>`.
`(val <x> <t> <v>)` means that `<v>` is the value on port `<x>` at time `<t>`.

3. Knowledge Base

```
(type x1 xorg)
(type x2 xorg)
(type a1 andg)
(type a2 andg)
```

```
(type o1 org)

(conn (in 1 f1) (in 1 x1))
(conn (in 1 f1) (in 1 a1))
(conn (in 2 f1) (in 2 x1))
(conn (in 2 f1) (in 2 a2))
(conn (in 3 f1) (in 2 x2))
(conn (in 3 f1) (in 1 a2))
(conn (out 1 x1) (in 1 x2))
(conn (out 1 x1) (in 2 a2))
(conn (out 1 a2) (in 1 o1))
(conn (out 1 a1) (in 2 o1))
(conn (out 1 x2) (out 1 f1))
(conn (out 1 o1) (out 2 f1))

(if (and (conn $x $y) (val $x $t $z)) (val $y $t $z))

(if (and (type $x andg) (val (in 1 $x) $t on) (val (in 2 $x) $t on))
    (val (out 1 $x) $t on))
(if (and (type $x andg) (val (in 1 $x) $t off))
    (val (out 1 $x) $t off))
(if (and (type $x andg) (val (in 2 $x) $t off))
    (val (out 1 $x) $t off))

(if (and (type $x org) (val (in 1 $x) $t off) (val (in 2 $x) $t off))
    (val (out 1 $x) $t off))
(if (and (type $x org) (val (in 1 $x) $t on))
    (val (out 1 $x) $t on))
(if (and (type $x org) (val (in 1 $x) $t on))
    (val (out 1 $x) $t on))

(if (and (type $x xorg) (val (in 1 $x) $t on) (val (in 2 $x) $t on))
    (val (out 1 $x) $t off))
(if (and (type $x xorg) (val (in 1 $x) $t on) (val (in 2 $x) $t off))
    (val (out 1 $x) $t on))
(if (and (type $x xorg) (val (in 1 $x) $t off) (val (in 2 $x) $t on))
    (val (out 1 $x) $t on))
(if (and (type $x xorg) (val (in 1 $x) $t off) (val (in 2 $x) $t off))
    (val (out 1 $x) $t off))
```

4. Sample Output

```
=> (assert '(val (in 1 f1) 1 on))
   p1

=> (assert '(val (in 2 f1) 1 off))
   p2

=> (assert '(val (in 3 f1) 1 on)
   p3

=> (trueps '(val (out $n f1) 1 $z))
   (((($n . 1) ($z . off) (t . t))
    (($n . 2) ($z . on)  (t . t)))

=> (assert '(toassert &p fc))
   p4
```

```
=> (assert '(val (in 1 f1) 2 on))
   p5

=> (assert '(val (in 2 f1) 2 on))
   p6

=> (assert '(val (in 3 f1) 2 off))
   p7

=> (lookups '(val (out $n f1) 2 $z))
   (((($n . 1) ($z . off) (t . t))
    (($n . 2) ($z . on)  (t . t)))
```

The Case of Income Tax Consultation

1. Problem Description

The goal of this problem is a consultation system that can determine the number of exemptions a person can claim on his federal income tax return. The program should take the initiative and ask its user whatever questions are necessary to make this determination. When it is done, it should be able to explain its results.

The solution shown here uses the rules from the 1983 edition of Lasser's income tax guide. The program uses depth-first backward chaining in trying to determine the number of the client's exemptions. Notable are the use of ASK to ask the client appropriate questions and the use of TEMPLATE to set up English-like translations of those questions.

2. Vocabulary

`(exemption <p> <e>)` states that person `<p>` has exemption `<e>`.
`(exemptions <p> <n>)` states that person `<p>` has `<n>` exemptions.

3. Knowledge Base

```
(if (and (bagof $e (exemption $x $e) $s) (len $s $n))
    (exemptions $x $n))

(if (dependent $x $y)
    (exemption $x $y))

(if (and (married $x) (spouse $x $y) (blind $y))
    (exemption $x spouseblind))

(if (and (married $x) (spouse $x $y) (age $y $n) (> $n 65))
    (exemption $x spouseage))

(if (blind $x)
    (exemption $x blind))

(if (and (age $x $n) (> $n 65))
    (exemption $x age))

(exemption $x self)

(len () 0)

(if (and (len $l $m) (+ 1 $m $n))
    (len ($x . $l) $n))

(if (and (test2 $x $y)
    (provable (test1 $x $y))
    (provable (test3 $x $y))
    (provable (test4 $x $y)))
```

```
    (dependent $x $y))

(if (and (sharedhome $x $y) (legalrelationship $x $y))
    (test1 $x $y))

(if (and (relation $y $x $z)
    (or (= $z son) (= $z daughter) (= $z child)
        (= $z grandchild) (= $z stepchild)
        (= $z brother) (= $z sister)
        (= $z stepbrother) (= $z stepsister)
        (= $z father) (= $z mother)
        (= $z parent) (= $z stepparent)
        (= $z grandparent) (= $z greatgrandparent)
        (= $z uncle) (= $z aunt)
        (= $z nephew) (= $z niece)
        (= $z fatherinlaw) (= $z motherinlaw)
        (= $z soninlaw) (= $z daughterinlaw)
        (= $z brotherinlaw) (= $z sisterinlaw)
        (and (= $z fosterchild) (sharedhome $x $z))))
    (test1 $x $y))

(if (and (support $x $y)
    (morethanhalf $x $y))
    (test2 $x $y))

(if (fulltimestudent $y)
    (test3 $x $y))

(if (and (gross $y $d) (< $d 1000))
    (test3 $x $y))

(if (and (age $y $n) (< $n 19))
    (test3 $x $y))

(if (and (resident $y $z) (member $z (US Canada Mexico)))
    (test4 $x $y))

(if (UScitizen $y)
    (test4 $x $y))

;;; Meta-level Facts

(if (askable &r)
    (totrueps (&r . &l) asks))

(toassert (askable &x) trfc)

(askable age)
(askable blind)
(askable fulltimestudent)
(askable gross)
(askable legalrelationship)
(askable married)
(askable morethanhalf)
(askable relation)
(askable resident)
(askable sharedhome)
(askable spouse)
(askable support)
```

```
(askable uscitizan)

(function age)
(function gross)
(function relation)
(function resident)
(function spouse)

(template (age &x &y) (&x is &y years old))
(template (blind &x) (&x is blind))
(template (fulltimestudent &x) (&x is a full time student))
(template (gross &x &y) (&x earned a total of &y dollars this year))
(template (legalrelationship &x &y)
     (the relationship between &x and &y is legal))
(template (married &x) (&x is married))
(template (morethanhalf &x &y)
     (&x supplies more than half of the support for &y))
(template (relation &x &y &z) (&x is the &z of &y))
(template (resident &x &p) (&x lives in &p))
(template (sharedhome &x &y) (&x and &y share the same home))
(template (spouse &x &y) (&y is the spouse of &x))
(template (support &x &y) (&x supported &y this year))
(template (uscitizen &x) (&x is a citizen of the United States))
```

4. Sample Output

```
=> (truep '(exemptions mrg $n))

 MRG IS $3 YEARS OLD.
Please type a value for $3 that makes this statement true.
$3 = 34

 MRG IS BLIND.
Please type "true", "false", or "unknown".
=> false

 MRG IS MARRIED.
Please type "true", "false", or "unknown".
=> f

 MRG SUPPORTED $1 THIS YEAR.
Are there any values that make this statement true?
Please type "true", "false", or "unknown".
=> yes
$1 = Sally
Are there any other values?
Please type "true", "false", or "unknown".
=> y
$1 = Katie
Are there any other values?
Please type "true", "false", or "unknown".
=> n

 MRG SUPPLIES MORE THAN HALF OF THE SUPPORT FOR SALLY.
Please type "true", "false", or "unknown".
=> y

 SALLY IS THE $7 OF MRG.
```

```
Please type a value for $7 that makes this statement true.
$7 = sister

 SALLY IS $8 YEARS OLD.
Please type a value for $8 that makes this statement true.
$8 = 27

 SALLY EARNED A TOTAL OF $9 DOLLARS THIS YEAR.
Please type a value for $9 that makes this statement true.
$9 = 2300

 SALLY IS A FULL TIME STUDENT.
Please type "true", "false", or "unknown".
=> y

 MRG SUPPLIES MORE THAN HALF OF THE SUPPORT FOR KATIE.
Please type "true", "false", or "unknown".
=> y

 KATIE IS THE $10 OF MRG.
Please type a value for $10 that makes this statement true.
$10 = friend

 MRG AND KATIE SHARE THE SAME HOME.
Please type "true", "false", or "unknown".
=> y

 THE RELATIONSHIP BETWEEN MRG AND KATIE IS LEGAL.
Please type "true", "false", or "unknown".
=> yes

 KATIE IS $11 YEARS OLD.
Please type a value for $11 that makes this statement true.
$11 = 23

 KATIE EARNED A TOTAL OF $12 DOLLARS THIS YEAR.
Please type a value for $12 that makes this statement true.
$12 = 750

 KATIE LIVES IN $13.
Please type a value for $13 that makes this statement true.
$13 = us

   ((/$N . 2) (/$14 . 1) (/$15 . 0) (/$2 KATIE SELF) (T . T))
```

The Case of the Deadly Wumpus

1. Problem Description

The Wumpus is a vicious, foul-smelling beast that inhabits a maze of subterranean caves in deepest, darkest Africa. Many good men have gone to their deaths hunting the Wumpus. Your mission, should you choose to accept it, is to create a program that can track down the Wumpus in its lair and shoot an arrow through its heart.

The maze of the Wumpus consists of 20 interconnected caves. Each cave is connected by tunnels to three other caves. Unfortunately for you, the hazards in the maze include not only the Wumpus but also pits and bats. If one is so unfortunate as to fall into a pit, one will perish. If one encounters bats, they will pick one up and deposit one elsewhere. Needless to say, encountering the Wumpus is lethal. Fortunately, the stench of the Wumpus is perceptible one cave away. There is a chilling draft blowing through all caves next to a cave containing a pit. Bats can be heard squeaking one cave away as well.

Your program is to be mounted on a high technology robot with powerful sensors and effectors. Each time the robot moves to a new cave, it will call the ASSERT command to place in your program's database the name of the current cave, the fact that it has been seen, and the names of the caves to which it is connected. If there are drafts, squeaks, or smells, it will assert those facts as well; otherwise it will assert their negations. The robot will then call TRUEP to determine what action it should perform. Legal actions include movement to another cave, shooting an arrow, starting a new game, and quitting. The vocabulary for all these assertions and actions is given below. The robot can move in one step to any cave it has already seen or to an unseen cave immediately connected to one it has already seen. The robot can shoot an arrow into an adjoining cave only. If the robot is killed, it is sent to heaven, to which there are no adjoining caves. Starting a new game will set up a new maze and deposit the robot there in a randomly selected cave.

Each action of the robot demands a certain amount of energy. A movement requires one unit of energy. Shooting an arrow requires 15 units of energy. Getting killed consumes 25 units of energy. Starting a new game requires 25 units of energy also. Your job is to write a program that directs the robot in pursuit of the Wumpus while using as little energy as possible. Good luck.

2. Vocabulary

(here <c>) states that cave <c> is the current location.
(seen <c>) states that cave <c> has already been visited.
(next <c> <d>) states that cave <c> is connected to cave <d>.
(draft <c>) states that there is a draft blowing through cave <c>.
(squeak <c>) states that squeaking is audible in cave <c>.

(smell <c>) states that the stench of th Wumpus pervades cave <c>.
(pit <c>) states that there is a pit in cave <c>.
(bats <c>) states that there are bats in cave <c>.
(wumpus <c>) states that the Wumpus is lurking in cave <c>.
(safe <c>) states that the cave <c> is free of hazards.

3. Knowledge Base

```
;;; Demons

(if (and (seen $c) (next $c $d)
         (unknown (seen $d)) (unknown (not (safe $d))))
    (wumpustask (move $c)))

(if (and (here $c) (next $c $d)
         (unknown (seen $d)) (unknown (not (safe $d))))
    (wumpustask (move $d)))

(if (and (seen $c) (next $c $d) (unknown (seen $d)) (safe $d))
    (wumpustask (move $c)))

(if (and (here $c) (next $c $d) (unknown (seen $d)) (safe $d))
    (wumpustask (move $d)))

(if (and (wumpus $c) (next $d $c) (seen $d))
    (wumpustask (move $d)))

(if (and (wumpus $c) (here $d) (next $d $c))
    (wumpustask (shoot $c)))

(if (here heaven)
    (wumpustask (start)))

;;; Facts about the Wumpus World

(if (seen $c)
    (not (pit $c)))

(if (seen $c)
    (not (bats $c)))

(if (seen $c)
    (not (wumpus $c)))

(if (and (next $c $d) (not (draft $c)))
    (not (pit $d)))

(if (and (next $c $d) (not (squeak $c)))
    (not (bats $d)))

(if (and (next $c $d) (not (smell $c)))
    (not (wumpus $d)))

(if (and (not (bats $c)) (not (pit $c)) (not (wumpus $c)))
    (safe $c))

(if (and (draft $c)
```

```
    (bagof $d (and (next $c $d) (unknown (not (pit $d)))) ($f)))
    (pit $f))

(if (and (squeak $c)
    (bagof $d (and (next $c $d) (unknown (not (pit $d)))) ($f)))
    (bats $f))

(if (and (smell $c)
    (bagof $d (and (next $c $d) (unknown (not (wumpus $d)))) ($f)))
    (wumpus $f))

(if (bats $c) (not (safe $c)))

(if (pit $c) (not (safe $c)))

(if (wumpus $c) (not (safe $c)))

;;; Meta-level facts

(toassert (seen &c) fc)
(toassert (draft &c) fc)
(toassert (squeak &c) fc)
(toassert (smell &c) fc)
(toassert (bats &c) fc)
(toassert (pit &c) fc)
(toassert (not (draft &c)) fc)
(toassert (not (squeak &c)) fc)
(toassert (not (smell &c)) fc)
(toassert (not (pit &c)) fc)
(toassert (not (bats &c)) fc)
(toassert (not (wumpus &c)) fc)

(totruep (wumpus &c) pr-lookup)
(totruep (safe &c) pr-lookup)
(totruep (not (safe &c)) pr-lookup)
```

4. Sample Output

```
=> (START)
Here's Major MRG descending into the lair of the Wumpus.
   Down.
   Down.
   Down.
You are in cave G
   Brrr! There's a draft here.
There are tunnels to Q P D

=> (MOVE D)
You are in cave D
There are tunnels to E G L

=> (MOVE L)
You are in cave L
   I hear squeaking.
There are tunnels to I D K

=> (MOVE D)
You are in cave D
```

```
There are tunnels to E G L

=> (MOVE E)
You are in cave E
There are tunnels to D C N

=> (MOVE N)
You are in cave N
   Brrr! There's a draft here.
There are tunnels to P E T

=> (MOVE E)
You are in cave E
There are tunnels to D C N

=> (MOVE C)
You are in cave C
There are tunnels to T J E

=> (MOVE J)
You are in cave J
   What a stench!
There are tunnels to S K C

=> (SHOOT S)
Congratulations! You killed the Wumpus.
Welcome to heaven!
Score = 1
Penalty = 45

=> (START)
You are in cave A
   I hear squeaking.
There are tunnels to P E C

=> (MOVE C)
Aieee! Bats!  You have been moved.
You are in cave A
   I hear squeaking.
There are tunnels to P E C

=> (MOVE E)
You are in cave E
   Brrr! There's a draft here.
   I hear squeaking.
There are tunnels to J D A

=> (MOVE D)
Aieee! Bats!  You have been moved.
You are in cave A
   I hear squeaking.
There are tunnels to P E C

=> (MOVE P)
You are in cave P
   Brrr! There's a draft here.
There are tunnels to A S Q

=> (MOVE Q)
```

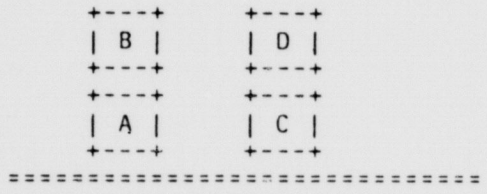
```
Oops!  You've just fallen into a pit.
Too bad.
Welcome to heaven!

=> (START)
You are in cave H
There are tunnels to O N S

=> (MOVE S)
You are in cave S
   What a stench!
There are tunnels to E H Q

=> (MOVE H)
You are in cave H
There are tunnels to O N S

=> (MOVE N)
You are in cave N
There are tunnels to C E H

=> (MOVE S)
You are in cave S
   What a stench!
There are tunnels to E H Q

=> (SHOOT Q)
Congratulations! You killed the Wumpus.
Welcome to heaven!
Score = 2
Penalty = 96

=> (START)
You are in cave Q
There are tunnels to L K M

=> (MOVE M)
You are in cave M
There are tunnels to J Q J

=> (MOVE J)
You are in cave J
   I hear squeaking.
There are tunnels to M M C

=> (MOVE Q)
You are in cave Q
There are tunnels to L K M

=> (MOVE K)
You are in cave K
   I hear squeaking.
There are tunnels to A L Q

=> (MOVE L)
You are in cave L
   What a stench!
There are tunnels to Q N K
```

```
=> (SHOOT N)
Congratulations! You killed the Wumpus.
Welcome to heaven!
Score = 3
Penalty = 122
```

The Case of the Blocks World Planner

1. Problem Description

The goal of this problem is the creation of a planner for the Blocks World. The World in this case consists of 5 blocks of equal size arranged on a table. There is a single hand that can move a block from one position to another, provided the block being moved is clear and the destination is clear. The table is always clear by definition. The following diagram shows the initial state.



The solution below uses RESIDUE to design a set of tasks to achieve any legal arrangement of blocks.

2. Vocabulary

(on <x> <y> <t>) means that block <x> is on <y> at time <t>.
(clear <x> <t>) means that <x> is clear at time <t>.
(execution <k> <s> <t>) means that task <k> should be executed from <s> to <t>.
(nonexecution <k> <s> <t>) means that <k> should not be executed.

3. Knowledge Base

```
(if (and (known (on $x $y $b))
     (execution (move $x $y $z) $s $t)
     (clear $x $s)
     (clear $z $s))
     (on $x $z $t))

(if (and (known (execution (move $x $y $z) $s $t))
     (unknown (= $u $x))
     (on $u $v $s))
     (on $u $v $t))

(if (and (known (on $x $y $s))
     (object $z)
     (nonexecution (move $x $y $z) $s $t))
     (on $x $y $t))

(if (and (known (on $x $y $b))
     (execution (move $x $y table) $s $t)
     (clear $x $s))
     (clear $y $t))
```

```
(if (and (known (execution (move $x $y $z) $s $t))
    (unknown (= $u $x))
    (clear $u $s))
    (clear $u $t))

(if (and (known (clear $x $s))
    (object $y) (object $z)
    (nonexecution (move $x $y $z) $s $t))
    (clear $x $t))

(if (and (> $t 1) (- $t 1 $s) (unprovable (disqualified $k $s $t)))
    (assumable (execution $k $s $t)))

(if (= t t)
    (assumable (nonexecution $k $s $t)))

(if (and (nonexecution $k $u $v)
    (>= $s $u)
    (<= $t $v))
    (disqualified $k $s $t))

(if (and (execution $j $s $t)
    (unknown (= $j $k)))
    (disqualified $k $s $  t))


(object a)
(object b)
(object c)
(object d)
(object table)

(on b a 1)
(on d c 1)
(on a table 1)
(on c table 1)

(clear b 1)
(clear d 1)
(clear table $t)
```

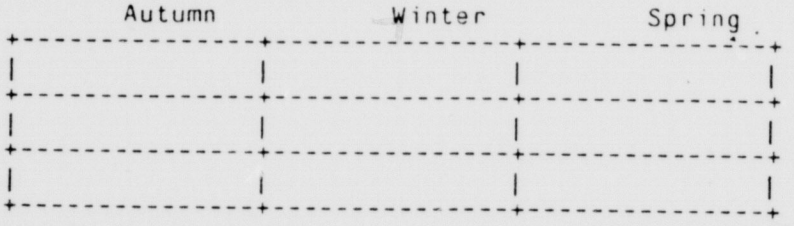
4. Sample Output

```
=> (residue (quote (on a c 9)))
   ((execution (move d c table) 8 9)
    (execution (move b a table) 7 8)
    (execution (move a table c) 6 7)
    (nonexecution (move b a table) 1 6)
    (nonexecution (move b a d) 1 6)
    (nonexecution (move d c table) 1 6)
    (nonexecution (move d c b) 1 6)
    (= t t))
```

The Case of the MSAI Program

1. Problem Description

The goal of this problem is to design a schedule of courses for a student in a one-year college program (dubbed MSAI). A blank schedule form is shown below. Of course, there are constraints on how the form can be filled out. A student can take at most 3 courses in each of 3 quarters; a student must take at least 9 courses; and all courses must be distinct. Five courses are required; the other 4 are elective. A student can only take a course in a quarter in which it is offered and only if he has already taken the prerequisites.



The solution below uses RESIDUE to design an appropriate program. A single default rule generates the space of possibilities. Each design decision is checked for consistency with previous design decisions before being made. Every consistent alternative becomes a possible design.

One interesting feature of this solution is that it allows an arbitrary degree of interaction. One can supply a complete curriculum, in which case the call to RESIDUE simply checks its validity. One can supply a partial curriculum and ask what courses are legal for the remaining slots. One can supply a partial curriculum or no curriculum at all and let the program do all of the planning.

2. Vocabulary

(msai <p>) states that the program <p> satisfies the msai requirements.
(full <p>) states that the program <p> is full.
(reqts <p>) states that the program <p> includes all required courses.
(c <q> <p> <c>) states that course <c> is taken in quarter <q> of program <p>.
(offered <c> <q>) states that course <c> is offered in quarter <q>.
(prereq <c> <d>) states that course <c> is a prerequisite for course <d>.

3. Knowledge Base

```
;;; Meta Rule

(if (and (offered $c $q) (unprovable (not (c $q $p $c))))
    (assumable (c $q $p $c)))

;;; Requirements
```

```
(if (and (reqts $p) (full $p))
    (msai $p))

(if (and (c $q1 $p cs223)
    (c $q2 $p cs222)
    (c $q3 $p cs156)
    (c $q4 $p cs142)
    (c $q5 $p cs161))
    (reqts $p))

(if (and (fullq f $p) (fullq w $p) (fullq s $p))
    (full $p))

(if (and (c $q $p $c1)
    (c $q $p $c2) (unknown (= $c1 $c2))
    (c $q $p $c3) (unknown (= $c1 $c3)) (unknown (= $c2 $c3)))
    (fullq $q $p))

;;; Consistency rules

(if (bagof $d (c $q $p $d) ($x $y $z))
    (not (c $q $p $c)))

(if (and (c $q1 $p $c) (unknown (= $q1 $q2)))
    (not (c $q2 $p $c)))

(if (and (prereq $c $d) (unknown (c $q1 $p $c)))
    (not (c $q2 $p $d)))

(if (and (prereq $c $d) (c $q1 $p $c) (before $q2 $q1))
    (not (c $q2 $p $d)))

;;; Data Base

(before f w)
(before f s)
(before w s)

(offered cs102 s)
(offered cs142 s)
(offered cs222 s)
(offered cs224 s)
(offered cs225 s)
(offered cs143 w)
(offered cs145 w)
(offered cs223 w)
(offered cs226 w)
(offered cs102 f)
(offered cs142 f)
(offered cs156 f)
(offered cs161 f)
(offered cs204 f)
(offered cs206 f)

(prereq cs223 cs222)
(prereq cs102 cs224)
(prereq cs142 cs145)
```

4. Sample Output

```
=> (residue '(msai mrg))
   ((C F MRG CS156) (C F MRG CS142) (C F MRG CS161)
    (C W MRG CS223) (C W MRG CS226) (C W MRG CS145)
    (C S MRG CS222) (C S MRG CS225) (C S MRG CS102) (= T T))

=> (assert '(c s mrg cs142))
   P303

=> (assert '(c f mrg cs102))
   P304

=> (residue '(msai mrg))
   ((C F MRG CS156) (C F MRG CS161)
    (C W MRG CS223) (C W MRG CS226) (C W MRG CS143)
    (C S MRG CS222) (C S MRG CS225) (= T T))
```