Fact Sheet

Course Title: *Artificial Intelligence*
Number: Computer Science 375A
Term: Fall Term, 1981
Lecture Times: Monday, Wednesday, Friday, 9:35-10:35 am
Lecture Location: Life Sciences 2967

Instructor: Dr. Reid G. Smith
Office: Defence Research Establishment Atlantic & Killam Library 4402
Office Hours: By appointment
Telephone: DREA - 426-3100 (leave messages with Jean Faulkner)

Text: Nilsson, Nils J., *Principles of Artificial Intelligence,* Tioga Publishing Company, Palo Alto, California, 1980, 476 pages.

Prerequisite: CS 270 (*Programming Languages*) or equivalent. [Prior knowledge of LISP will be helpful]
Corequisite: CS 360A (*Data Structures and Algorithm Analysis*)

Course Purpose: This course is intended to provide an introductory overview of the field of Artificial Intelligence, to illustrate the basic techniques in a variety of problem domains, and to isolate current problems and topics for future research.

Homework Assignments: There will be approximately 10 homework assignments.

Grading: Grades in the course will be 65% dependent on the homework and 35% dependent on the final exam. It is therefore imperative that you attempt each assignment and turn something in.

Due Date: October 5, 1981.

1) Write a LISP function, ALT, whose value is a list whose elements are alternate elements of its argument.

```
(ALT ‘(A B C)) = (A C)
(ALT ‘((A B) (C D))) = ((A B))
```

2) Write a LISP function, ATLIS, whose value is a list of the unique atoms in an S-expression.

```
(ATLIS ‘(A B C) = (A B C NIL)
(ATLIS ‘((A) V ((V (B) (((D))))))) = (A V B D NIL)
```

3) Write a LISP function, SUBSTITUTE, that substitutes atom X for atom Y in S-expression Z.

```
(SUBSTITUTE ‘X ‘A ‘(PLUS (TIMES A B) (QUOTIENT C (SUB1 A)))) =
    (PLUS (TIMES X B) (QUOTIENT C (SUB1 X)))
```

4) Write a LISP function, ALLPAIRS, that computes a list of all pairs of elements, each pair taking one element from its first argument and one from its second argument.

```
(ALLPAIRS ‘(A B C) ‘(D E)) = ((A D) (A E) (B D) (B E) (C D) (C E))
```

Observe the ordering given in the example.

5) Write a LISP function to convert a list of English words to Pig Latin. Each English word is translated to pig latin by the following rules:
    1) If the first letter of the English word is a vowel then the Pig Latin translation is the same as the English word.
    2) Otherwise, rotate the first letter of the word to the end of the word and repeat until the first letter becomes a vowel. Add the letters “AY” to the end of the word to complete the Pig Latin translation.
    3) For the purposes of this problem, vowels are A E I O and U.
    4) Assume every English word contains at least one vowel.

```
(PIGLATIN ‘(I LIKE LISP BETTER THAN PIG LATIN)) =
    (I IKELAY ISPLAY ETTERBAY ANTHAY IGPAY ATINLAY)
```

You will need the two LISP functions:

```
(EXPLODE ‘ABO = (A B C)    (COMPRESS ‘(A B C)) = ABC
```

6) Write a pair of LISP functions, SUBEXP and LOCATIONS. SUBEXP gives the sub-expression at location x, where x is a list of A’s and D’s that indicates the CAR/CDR chain to follow to reach the desired subexpression. LOCATIONS returns a list of all locations of a given subexpression, in the same format.

```
(LOCATIONS ‘X ‘(PLUS X (TIMES X Y))) = ((D A) (D D A D A))
(SUBEXP ‘(D D A D A) ‘(PLUS X (TIMES X YD)))
```

Hint: You will find MAPCAR doubly helpful here.

In general, many of the function definitions on the assignments were much more complicated than they needed to be. By deciding first on the simplest possible terminating conditions, and then deciding how more complicated cases should be decomposed, one can usually write a simple recursive function to do the job,

Hints about LISP programming:

1) Using the `DEF` function is easier than using `DEFINE`. Instead of
   `(DEFINE ‘((FNAME (ARGLIST) ... ))) use (DEF (FNAME (ARGLIST ... )).`

2) There are a large number of system defined functions which can be useful. It is well worth your while to consider the functions already available before you go ahead and write your own. For instance, a number of students wrote the `PIGLATIN` function with a helping function to check for vowels. The definition of `VOWEL` is trivial if one uses the system defined function `MEMBER`. However, many people did not use `MEMBER` and wasted time writing a function using only `COND` which was 5 or 6 times longer than necessary.

3) When considering manipulations on a list where you want to do something with all the top level elements, use a `MAP` function. Use of `MAPCAR` for the definitions of the `ALLPAIRS`, `PIGLATIN` and `LOCATIONS` functions greatly simplifies the solution.

4) All programming assignments should be well documented with comments.

5) Finally, the format and proper indentation of your programs isimportant so that you can read and debug programs. It also helps you get a good mark on assignments. Consider spending a few minutes to define some useful functions for this purpose.

For example, the function `PRETTY` takes a list of function names as an argument and pretty prints them all:

```
(DEF
  (PRETTY (FLIST)
    (MAPC FLIST (F/L FN) (TERPRI)
                         (PRIN1 ‘$$$ FUNCTION $)
                         (PRINT FN)
                         (PPRINT (GETD FN))))))
```

This definition of `PRETTY` works only for LISP 5.0 on the Cyber. A more sophisticated version might handle errors in the argument and make a distinction between `EXPR's` and `FEXPR's`.

```
(PRETTY FLIST)

FUNCTION PRETTY
(LAMBDA (FLIST)
(MAPC FLIST
      (F/L (FN) (TERPRI) (PRIN1 ‘FUNCTION) (PRINT FN) (PPRINT (GETD FN)))))
```

```
FUNCTION ALT
(LAMBDA (L)
(COND ((OR (NULL L) (NULL (CDR L))) L)
      (T (CONS (CAR L) (ALT (CDDR L))))))

FUNCTION ATLIS
(LAMBDA (L)
(ATLIS2 L NIL))

FUNCTION ATLIS2
(LAMBDA (L COLLECT)
(COND ((ATOM L (COND ((MEMBER L COLLECT) COLLECT) (T (CONS L COLLECT))))
      (T (ATLIS2 (CAR L) (ATLIS2 (CDR L) COLLECT))))))

FUNCTION SUBSTITUTE
(LAMBDA (PATTERN SUB DATA)
(COND ((AND (ATOM DATA) (EQ PATTERN DATA)) SUB)
      ((ATOM DATA) DATA)
      (T (CONS (SUBSTITUTE PATTERN SUB (CAR DATA))
               (SUBSTITUTE PATTERN SUB (CDR DATA))))))

FUNCTION ALLPAIRS
(LAMBDA (L1 L2)
(COND ((NULL L1) NIL)
      (T (APPEND (MAPCAR L2 (F/L (X) (CONS (CAR L1) (CONS X NIL))))
                 (ALLPAIRS (CDR L1) L2)))))

FUNCTION PIGLATIN
(LAMBDA (LIST)
(MAPCAR LIST (F/L (WORD) (COMPRESS (PIGTRANS (EXPLODE WORD))))))

FUNCTION PIGTRANS
(LAMBDA (WORDLST)
(COND ((MEMBER (CAR WORDLST) '(COND ((MEMBER (CAR WORDLST) '(A E I O U))
                                     (APPEND (WORDLST ‘(A Y))))
      (T (PIGTRANS (APPEND (CDR WORDLST) (LIST (CAR WORDLST)))))))

FUNCTION LOCATIONS
(LAMBDA (PATTERN DATA)
(COND ((EQUAL PATTERN DATA) '(NIL))
      ((ATOM DATA) NIL)
      (T
        (APPEND (MAPCAR (LOCATIONS PATTERN (CAR DATA))
                        (F/L (X) (CONS 'A X)))
                (MAPCAR (LOCATIONS PATTERN (CDR DATA))
                        (F/L (X) (CONS ‘D X)))))))

FUNCTION SUBEXP
(LAMBDA (LOC DATA)
(COND ((NULL LOC) DATA)
      ((EQ (CAR LOC) 'A) (SUBEXP (CDR LOC) (CAR DATA)))
      ((EQ (CAR LOC) 'D) (SUBEXP (CDR LOC) (CDR DATA)))
      (T '(ILLEGAL DATA))))
EVAL >> NIL
FIN

GARBAGE COLLECTIONS = 0 0

18.12.01.UCLP, AA50,      0.128KLNS. CVTA036.
```

Due Date: October 14, 1981.

1) Specify a global database, rules, and a termination condition for a production system to solve the following water-jug problem:

   > Given a 7-liter jug filled with water, an empty 5-liter jug, and an empty 3-liter jug, how can one obtain precisely 1 liter in the 5-liter jug? Water may poured from one jug into another, but never out of a jug into oblivion.

   Draw that part of the search graph corresponding to the moves you tried in finding a solution.

2) Use the following set of rewrite rules to determine whether or not *"The president approves the sale of the new company."* is a grammatical sentence.

   DNP VP → *S*
   V DNP → *VP*
   P DNP → *PP*
   of → *P*
   approves → *V*
   DET NP → *NP*
   DNP PP → *DNP*
   A NP → *NP*
   N → *NP*
   new → *A*
   president → *N*
   company → *N*
   sale → *N*
   the → *DET*

   Describe how the rules can be used to *generate* sentences. What is the global database and the termination condition for such a system? Use the system to generate 5 grammatical (even if not meaningful) sentences.

3) The *monkey-and-bananas* problem is often used in the AI literature to demonstrate ideas about common-sense reasoning. The problem can be stated as follows:

   > A monkey is in a room containing a box and a bunch of bananas. The bananas are hanging from the ceiling out of reach of the monkey. How can the monkey obtain the bananas. (The monkey is supposed to go to the box, push it under the bananas, climb on top of it and grasp the bananas.)

   Specify a global database, rules, and a termination condition for a production system to solve the monkey-and-bananas problem. Show the complete search graph.

1. There are a number of possible representations for the data base. Let the current levels in the jugs be represented by a state vector (JUG3 JUG5 JUG7) where the value of JUGi for i = [3,5,7] is the number of litres of water in the corresponding jug of capacity I litres.

Notice that there are constraints on the state vector -
$(JUG3 \leq 3) \wedge (JU65 \leq 5) \wedge (JUG7 \leq 7)$. Also, $(JUG3 + JUG5 + JUG7) = 7$.

The initial state of the global database is the state vector (0 0 7). Termination condition for the production system is (x 1 6-x) for $0 \leq x \leq 6$.

Production Rules
Assume the existence of a function Size which returns the capacity of a jug; e.g., Size(JUG3) = 3.

```
R1) Fill-jug(Source,Dest)
      Precondition: (Dest < Size(Dest)) ∧ (Source ≥ (Size(Dest) - Dest))
      Action:
                 Source := Source - (Size(Dest) - Dest)
                 Dest := Size(dest)

R2) Empty-jug(Source,Dest)
       Precondition: (Dest < Size(Dest)) ∧ (Source < (Size(Dest) - Dest))
       Action:
                  Dest := Dest + Source
                  Source := 0
```

Note that a rule may be applied to any source-destination pair of jugs provided the precondition is satisfied. Successors of a state vector should not be expanded if they have already been generated previously.

```
                       Search Graph

                          (7 0 0)
               (4 0 3)               (2 5 0)
       (0 4 3)      (4 3 0)        (0 5 2)   (2 2 3)
  (3 4 0)                     (5 0 2)
(3 1 3) Goal
```

2. In order to determine whether "The president approves the sale of the new company" is a grammatical sentence, use the rewrite rules to transform it into the symbol S. If this is possible, the sentence is grammatical.

It is convenient to number the rewrite rules. Whenever you use a rule to transform the sentence, the pattern on the left hand side is replaced by the symbol on the right hand side. At this point, the rule number can be cited to show which rule was applied.

1) *DNP VP $\rightarrow$ S*
2) *V DNP $\rightarrow$ VP*
3) *P DNP $\rightarrow$ PP*
4) *of $\rightarrow$ P*
5) *approves $\rightarrow$ V*
6) *DET NP $\rightarrow$ NP*
7) *DNP PP $\rightarrow$ DNP*
8) *A NP $\rightarrow$ NP*
9) *N $\rightarrow$ NP*
10) *new $\rightarrow$ A*
11) *president $\rightarrow$ N*
12) *company $\rightarrow$ N*
13) *sale $\rightarrow$ N*
14) *the $\rightarrow$ DET*

```
the president approves the  sale  of the   new company
(14)  (11)       (5)   (14) (13) (4) (14) (10) (12)
DET   N          V     DET   N    P   DET  A     N
     (9)                   (9)                  (9)
DET  NP         V      DET  NP   P    DET  A     NP
                                             (8)
DET NP        V        DET NP   P  DET     NP
 (6)                    (6)          (6)
DNP           V        DNP     P      DNP
                                (3)
DNP           V        DNP     PP
                          (7)
DNP           V      DNP
               (2)
DNP       VP
   (1)
   S
```

The rules are used to generate sentences by running them backwards. That is, the symbol on the right hand side is replaced by the pattern on the left hand side. The initial global database for such a system consists of the symbol S.

Termination conditions is a global database which consists entirely of terminal symbols. The terminal symbols in this case consist of [of, approves, new, president, company, sale, the]. The following example shows a sentence generated by this method:

```
S
(1)
DNP VP
(6)
DET NP  VP
        (2)
DET NP  V DNP
          (6)
DET NP  V DET NP
    (9)       (9)
DET   N  V  DET  N
(14) (11)(5) (14)(13)
the president approves the sale
```

3. There are number of equally valid representations for problem there. Using the state vector once more is probably the easiest, however. Let the present state be represented by the state vector `(Mpos, Mht, Bpos, Has-bananas)`.

`Mpos` = horizontal position of the monkey (two dimensional vector)
`Mht` = boolean value (True if the monkey is on the box, False otherwise)
`Bpos` = horizontal position of the box (two dimensional vector)
`Has-bananas` = boolean vaiue (True if the monkey has the bananas in his grasp, False otherwise)

Assume also the existence of the global variable Banana-location, which contains the horizontal position of the bananas. If this production system was actually implemented, it is possible that a number of changes would be required in the database when a rule action is triggered. Assume, therefore, that there are four procedures which will handle all of these implementation details:

`Goto(loc)` moves the monkey to location loc.
`Push-box(loc)` pushes the box to location loc.
`Climb-box` puts the monkey on top of the box.
`Grasp-bananas` puts the bananas in the physical possession of monkey.

Production Rules:

```
R1) Go-to-box(Mpos,Bpos)
       Precondition: Mpos ≠ Bpos
       Action: Goto(Bpos)
                 Mpos := Bpos

R2) Push-box-to-bananas(Mpos,Bpos)
       Precondition: (Mpos = Bpos) ∧ Bpos ≠ Banana-location)
       Action: Push-box(Banana-location)
                 Mpos := Banana-location
                 Bpos := Banana-location

R3) Climb-to-bananas(Mpos,Bpos,Mht)
       Precondition: (Mpos = Banana-location) ∧ (Bpos = Banana-location)
                           ∧ ~(Mht)
       Action: Climb-box
                 Mht := true
```

```
R4) Take-bananas(Mpos,Bpos,Mht,Banana)
       Precondition: (Mpos = Banana-location) ∧ (Bpos = Banana-location)
                           ∧ (Mht) ∧ ~(Banana)
       Action: Grasp-bananas
                 Banana = true
```

The.termination condition for the production system is (Has-bananas = true), or put more simply, the condition Has-bananas.

Search Graph

Suppose we start the monkey off at position `M1`, the box is in position `B1`, and the bananas are in position `BAN`, where `M1 ≠ B1 ≠ BAN`. The initial contents of the global database are: `(M1,false,B1,false)`

The search graph is:

```
                              (M1,false,B1,false)
                (B1,false,B1,false)
        (BAN,false,BAN,false)
   (BAN,true,BAN,false)
(BAN,true,BAN,true) Goal
```

Due Date: October 21, 1981.

Suppose you are in the middle of a monopoly game when you land on Polya Place, owned by your opponent. For committing such an outrageous act, you must pay your opponent $23 in rent. Unfortunately, you have only two $100 bills, one $10 bill, and four $1 bills, while your opponent has one $50 dollar bill, two $20 bills, and one $1 bill. Furthermore, you are playing by special rules so that the bank (or anyone else) can not help you make change. Thus, you must find a way to rearrange these 11 bills so that you are $23 poorer and your opponent is $23 richer.

1) Describe a global database, rules, and a termination condition for this problem.
2) Propose a heuristic function h that will estimate how close a state is to a goal state. Is your h function such that A* is admissible? Does it obey the monotone restriction?
3) Using the simple `GRAPH-SEARCH` described in class produce a running A* program to solve this problem. Your program should print the path to a solution in human-readable form. Use the simple form of `EXPAND` that produces all successors. Make each node in the search an atom. (Remember that `GENSYM` will generate new atoms as necessary.) Maintain the necessary information on the property list of each atom you generate.
4) For fun and extra credit: Instrument your program to measure the number of nodes expanded and the number of nodes generated at each level of the search. Try this for more than one h function, including one for which A* is admissible and one for which it is not. Try varying the weights on g and h.

1) There are many possible variations on the representation of the global database, A set of two state vectors, called MY-BILLS and YOUR-BILLS, is adequate for the task. These vectors are of the form (B100 B50 B20 B10 B1).

Bn represents the number of n-dollar bills held by a player. The global database consists of a set of nodes (atoms) with these vectors stored on their property lists. Additional information, such as the total amount held by each player, etc., can also be represented, but is not required because such values can be quickly calculated from the state vectors.

Production Rules

```
Functions - Value(BILLS) returns total sum of bills held
            Add-bill(BILLS,Bn) adds single bill to vector of bills
            Sub-bill(BILLS,Bn) removes single bill from vector of bills

Rules -

Give-a-bill(Bn) --Bn represents an n-dollar bill
      Precondition: Value(MY-BILLS) > 191
                      MY-BILLSn > 1 -- at least one bill to give
      Action: YOUR-BILLS := Add-bill(YOUR-BILLS,Bn)
                MY-BILLS := Sub-bill(MY-BILLS,Bn)
Take-a-bill(Bn) --Bn represents an n-dollar bill
      Precondition: Value(MY-BILLS) < 191
                      YOUR-BILLSn > 1 – at least one bill to take
      Action: MY-BILLS := Add-bill(MY-BILLS,Bn)
                YOUR-BILLS := Sub-bill(YOUR-BILLS,Bn)

Termination_condition: Value(MY-BILLS) = 214 - 23 = 191
```

2) The general form of an evaluation function is as follows: F(n) := (1-w)G(n) + wH(n), where F(n) is the total estimated cost of the optimal path solution, G(n) is the actual measured cost from the start node to the present position, and H(n) is the estimated cost from the present node to the goal node (i.e., the node where the termination condition is satisfied). w is the weighting factor to determine whether G(n) or H(n) has more influence on the final evaluation value.

Almost everyone used a simple difference heuristic, something like this:
H(n) := ABS(Value(MY-BILLS) - 191).
This heuristic measures the distance to the goal by deciding how many dollars apart the present node is from the goal node.

This is not a particularly accurate heuristic, because it fails to take into account that a single exchange of a large denomination bill can take the next node many dollars closer to the goal. Also, it may be necessary to increase the difference between a node and the goal state before the goal state is finally reached.

An H(n) which approximates more closely the number of bill exchanges required to reach the goal does not need to be complex.

One simple possibility is ABS(Sum(YOUR-BILLS) - 6) where Sum is a function which adds up all Bn in in the state vector. For this function, the goal node ((1 1 2 0 1) (1 10 0 1 4)) has the heuristic value H(n) := ABS(5 - 5) + ABS(6 - 6) = 0.

This takes—advantage of the fact that when we have finished, the opponent will have to hold 6 bills, and we will have to hold 5 bills.

There are also many other possible H(n) functions, such as an H(n) which calculates the number of one dollar bills which have to be exchanged to reach the goal node, and then adds 1 for every remaining difference of $50 between Value(MY-BILLS) and $191. The more ingenuity that is applied in the construction of the heuristic function, the smaller the search is.

Admissibility of A*

An algorithm A is admissible iff the algorithm always terminates in an optimal path from starting node s to a goal node whenever a path from s to a goal node exists.

A discussion of admissibility is found in the text pp. 76-79. Nilsson derives several important results regarding the GRAPH-SEARCH procedure. The main point of his results is that they prove the GRAPH-SEARCH algorithm is admissible, but only if the heuristic function H(n) satisfies the following condition: $H(n) \leq H^*(n)$ for all n in the set of graph nodes. (This is also the condition for an algorithm to be A*. All A* algorithms are admissable.)

That is, your GRAPH-SEARCH is only admissible if your heuristic function is a lower bound on the optimum cost to reach the goal node. To show you have an admissible search algorithm, it is necessary to prove this precondition. Most did not prove this; however an answer which demonstrated an understanding of the requirements was sufficient for almost full marks.

Also note that the function H(n) which was most popular (i.e., the difference in amount of money from amount of money in goal state) did not satisfy this condition (although appropriate weighting could make the algorithm effectively an admissible A*). The easiest function to write which definitely means the search will be admissible A* is H(n) := 0. (Obviously, $0 \leq H(n)$ for all n).

Regarding the requirements of the monotone restriction, most H functions did not satisfy the monotone restriction, which states $H(n1) \leq C(n1,n2) + H(n2)$.

In words, this says that the heuristic estimate of the remaining cost to reach the goal is always equal to or less than the real cost to reach a successor node plus the estimate of reaching the goal from that successor. The heuristic function most widely used, the dollar difference from a node to the goal, definitely fails to satisfy the monotone restriction. Most other H(n) functions also failed to satisfy the monotone restriction.

Due Date: October 30, 1981.

Note: Hand in Homework Set 3 at the same time.

1) Problem 3.1 in Nilsson.
2) Problem 3.3 in Nilsson.
3) In qualitative terms, discuss how a better static evaluator, a better plausible move generator (a mechanism for heuristically ordering moves for further exploration), and a more competent opponent affect the balance between depth and breadth of search.

1. Here is the graph for problem 1 (in a simple list format), where N6 corresponds to the integer "6" in the problem.

```
  ((N6 6 (N3 N3)
         (N4 N2)
    (N3 3 (N2 N1)
    (N4 4 (N2 N2)
         (N3 N1)
    (N2 2 (N1 N1))
    (N1 0))
```

There are a number of different traces of AO*, operation that could occur as a result of different orderings in the above graph. Here is the trace for one of them.

```
Partial Solution Graph:
  N6    Cost: 6
    2 Connector: => N3 N3
    2 Connector: => N4 N2

Selected Nonterminal Leaf Node: N6
Revising Cost of node: N6 -- Current Cost: 6
  New cost: 8
Partial Solution Graph:
  N6    Cost: 8
    2 Connector: => N3 N3 Marked
    2 Connector: => N4 N2
  N3    Cost: 3
    2 Connector: => N2 N1

Selected Nonterminal Leaf Node: N3
Revising Cost of node: N3 -- Current Cost: 3
  New cost: 4
Adding Parents: N6
Revising Cost of node: N6 -- Current Cost: 8
Partial Solution Graph:
  N6    Cost: 8
    2 Connector: => N3 N3
    2 Connector: => N4 N2 Marked
  N4    Cost: 4
    2 Connector: => N2 N2
    2 Connector: => N3 N1
  N2    Cost: 2
    2 Connector: => N1 N1

Selected Nonterminal Leaf Node: N4
Revising Cost of node: N4 -- Current Cost: 4
  New cost: 6
Adding Parents: N6
Revising Cost of node: N6 -- Current Cost: 8
  New cost: 10
```

```
Partial Solution Graph:
  N6    Cost: 10
    2 Connector: => N3 N3 Marked
    2 Connector: => N4 N2
  N3    Cost: 4
    2 Connector: => N2 N1 Marked
  N2    Cost: 2
    2 Connector: => N1 N1
  N1    Cost: 8

Selected Nonterminal Leaf Node: N2
Revising Cost of node: N2 -- Current Cost: 2
  Node Solved
Adding Parents: N3 N4
Revising Cost of node: N3 -- Current Cost: 4
  Node Solved
Adding Parents: N6
Revising Cost of node: N4 -- Current Cost: 6
  Node Solved
Revising Cost of node: N6 -- Current Cost: 18
  Node Solved
Partial Solution Graph:
  N6    Cost: 18
    2 Connector: => N3 N3 Marked
    2 Connector: => N4 N2
  N3    Cost: 4
    2 Connector: => N2 N1 Marked
  N2 Cost: 2
    2 Connector: => N1 N1 Marked
  N1    Cost: 0

(SUCCESS 10 N6 N3 N2 N1)
```

2. Here is a trace of the solution by MINIMAX with Alpha-Beta (as shown in class). General point: For those of you who got a different optimum backed-up value using the right-to-left ordering than was shown in Nilsson with left-to-right ordering. That doesn't make sense--does it! Naturally you should get the same answer--the only thing that will vary is the work involved.

```
  Node evaluated, value -2
  Node evaluated, value 2
  Node evaluated, value -3
  Node evaluated, value 3
  Node evaluated, value -1
  Node evaluated, value -1
  Node evaluated, value 1
MINIMAX trims 1 at level 2
  Node evaluated, value 1
  Node evaluated, value 0
MINIMAX trims 1 at level 4
MINIMAX trims 1 at level 2
  Node evaluated, value 2
  Node evaluated, value 3
  Node evaluated, value -3
MINIMAX trims 1 at level 3
  Node evaluated, value 0
  Node evaluated, value -3
  Node evaluated, value 1
  Node evaluated, value 5
  Node evaluated, value 1
MINIMAX trims 1 at level 5
  Node evaluated, value -5
MINIMAX trims 1 at level 5
  Node evaluated, value 2
  Node evaluated, value 5
  Node evaluated, value 3
  Node evaluated, value -2
  Node evaluated, value 2
  Node evaluated, value 0
MINIMAX trims 1 at level 5
  Node evaluated, value 3
  Node evaluated, value 3
  Node evaluated, value -3
  Node evaluated, value 5
  Node evaluated, value 0
Backed-up value and Move Sequence: 1 (2 2 2 1 2 1)
```

(Note that the move sequence above also uses right-to-left ordering.)

3.

(a) If the static evaluator were perfect, it would be unnecessary to apply it beyond those situations achievable in one move. The search would be shallow but wide.

(b) A good plausible move generator offers up the most likely move choices first. One expects the breadth of the search can be reduced and the depth extended.

(c) A more competent opponent may mean a shift either to breadth or depth depending on the kind of competence involved. Is the opponent competent because he never makes the careless mistakes one would associate with not looking at the obviously available moves? Or is he competent because he follows the likely lines of play far into deep combinations?

Due Date: November 6, 1981.

1) Problem 4.1 in Nilsson.

2) Problem 4.6 (a,b) in Nilsson.

3) Problem 5.1 in Nilsson.

4) Problem 5.4b in Nilsson.

5) Problem 5.9 in Nilsson.

6) Formulate as predicate calculus expressions the facts given below and prove the result requested. (Watch out, this is a tricky one.)

   Every two people have a mutual friend. Everyone is either a female or a male. There exists at least one male and one female in the world. Prove that a male and female exist who are friends.

Artificial Intelligence Comp. Sci. 375
Sample Solutions for Assignment 5

1) functions - Father(x, y) returns true iff y is the father of x

Mother(x, y) returns true iff y is the mother of x

Ancestor(x, y) returns true iff y is an ancestor of X

$(\forall x)[\text{Ancestor}(\text{Bill}, x) \Rightarrow$

$[\text{Father}(\text{Bill}, x) \vee \text{Mother}(\text{Bill}, x)$

$\vee \quad (\exists y)[\ [\text{Father}(\text{Bill}, y) \vee \text{Mother}(\text{Bill}, y)]$

$\wedge \text{Ancestor}(y, x)\ ]\ ]$

considering the predicate calculus representations as functions and objects may make it clear that an expression such as (Ancestor(Father(x,

is not legal.

2) a. $(\forall x)[P(x) \Rightarrow P(x)]$

$(\forall x)[\sim P(x) \vee P(x)]$ eliminate implication

$\sim P(x) \vee P(x)$ eliminate universal quantifier

b. $\{\sim\{(\forall x)\ P(x)\}\} \Rightarrow (\exists x)[\sim P(x)]$

$\sim\{\sim\{(\forall x)\ P(x\}\} \vee (\exists x)[\sim P(x)]$ remove implication

$\forall x\ P(x) \vee (\exists x)[\sim P(x)]$ skolemize

$\forall x\ P(x) \vee \sim P(A)$ with constant A

$(\forall x)[\ P(x) \vee \sim P(A)]$ prenex form

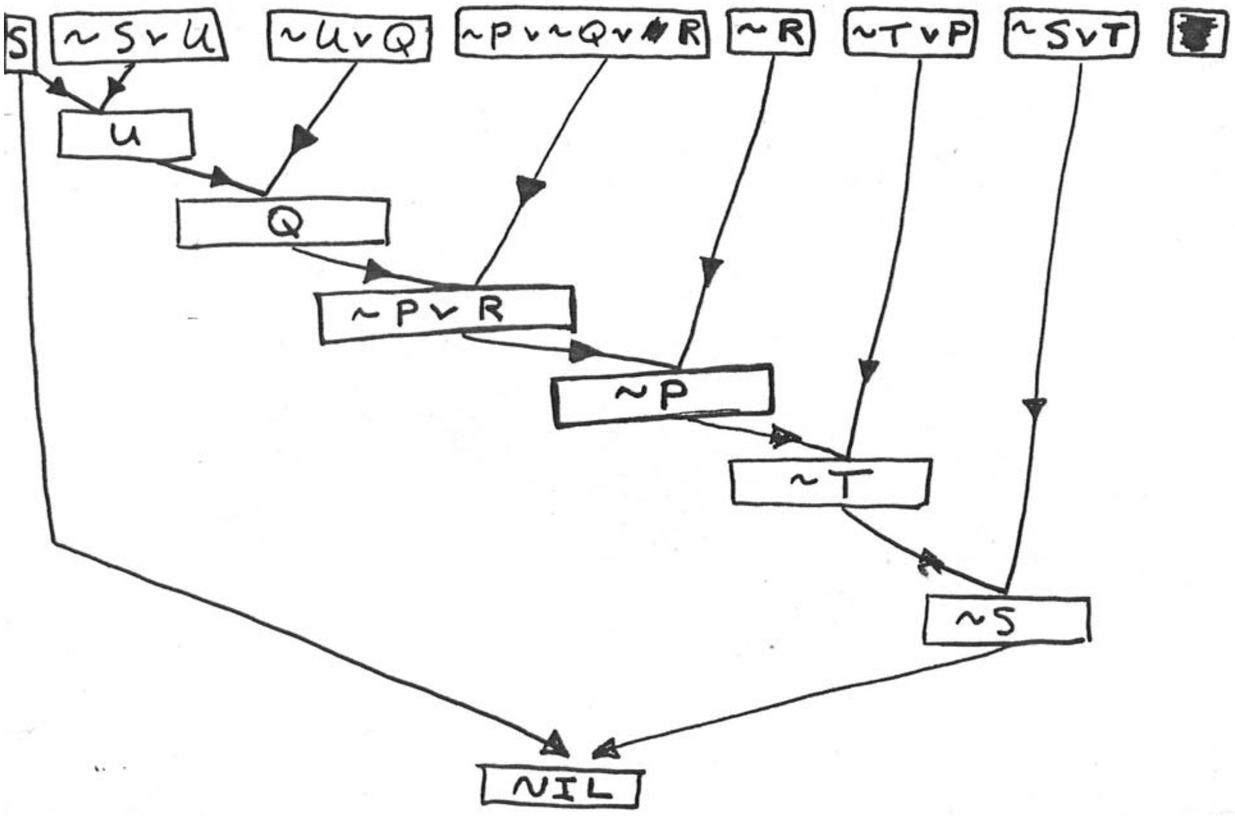
2b. $P(x) \vee \sim P(A)$ remove universal quantifier.

reference - section 4.2 in text pp. 145-149

3) The linear input form resolution strategy means every resolvent has one parent clause in the base set.

Base set

C1	$\sim P \vee \sim Q \vee R$
C2	$\sim S \vee T$
C3	$\sim T \vee P$
C4	S
C5	$\sim R$
C6	$\sim S \vee U$
C7	$\sim U \vee Q$



4) Prove $(\forall z)[Q(z) \Rightarrow P(z)] \Rightarrow (\exists x)[[Q(x) \Rightarrow P(A)] \wedge [Q(x) \Rightarrow P(B)]]$

in order to prove this wff, negate it and use refutation resolution, that is, resolve it to NIL.

$\sim\{(\forall z)[Q(z) \Rightarrow P(z)] \Rightarrow (\exists x)[[Q(x) \Rightarrow P(A)] \wedge [Q(x) \Rightarrow P(B)]]$

convert to clause form

① remove implications

$\sim\{\{\sim(\forall z)[\sim Q(z) \vee P(z)]\} \vee (\exists x)[[\sim Q(x) \vee P(A)] \wedge [\sim Q(x) \vee P(B)]$

② reduce scope of ~ sign

$\{\sim\sim(\forall z)[\sim Q(z) \vee P(z)]\} \wedge \sim(\exists x)[[\sim Q(x) \vee P(A)] \wedge [\sim Q(x) \vee P(B)$

$(\forall z)[\sim Q(z) \vee P(z)] \wedge (\forall x)[\sim[\sim Q(x) \vee P(A)] \vee \sim[\sim Q(x) \vee P(B$

$(\forall z)[\sim Q(z) \vee P(z)] \wedge (\forall x)[[\sim\sim Q(x) \wedge \sim P(A)] \vee [\sim\sim Q(x) \wedge \sim P($

$(\forall z)[\sim Q(z) \vee P(z)] \wedge (\forall x)[[Q(x) \wedge \sim P(A)] \vee [Q(x) \wedge \sim P(B)]]$

③ ~~standardize variables~~ $\forall$ equivalence (p.139)

$(\forall x)[\sim Q(x) \vee P(x) \wedge (\forall x)[[Q(x) \wedge \sim P(A)] \vee [Q(x) \wedge \sim P(B)]]$

④ Pre-nex form

$(\forall x)[[\sim Q(x) \vee P(x)] \wedge [[Q(x) \wedge \sim P(A)] \vee [Q(x) \wedge$

⑤ remove universal quantifiers.

$[[\sim Q(x) \vee P(x)] \wedge [[Q(x) \wedge \sim P(A)] \vee [Q(x) \wedge \sim P(B)]]]$

⑥ conjunctive normal form (use distributive laws)

$[[\sim Q(x) \vee P(x)] \wedge [[Q(x)] \wedge [\sim P(A) \vee \sim P(B)]]]$

⑦ clause form

$(\sim Q(x) \vee P(x))$

$(Q(x))$

$(\sim P(A) \vee \sim P(B))$

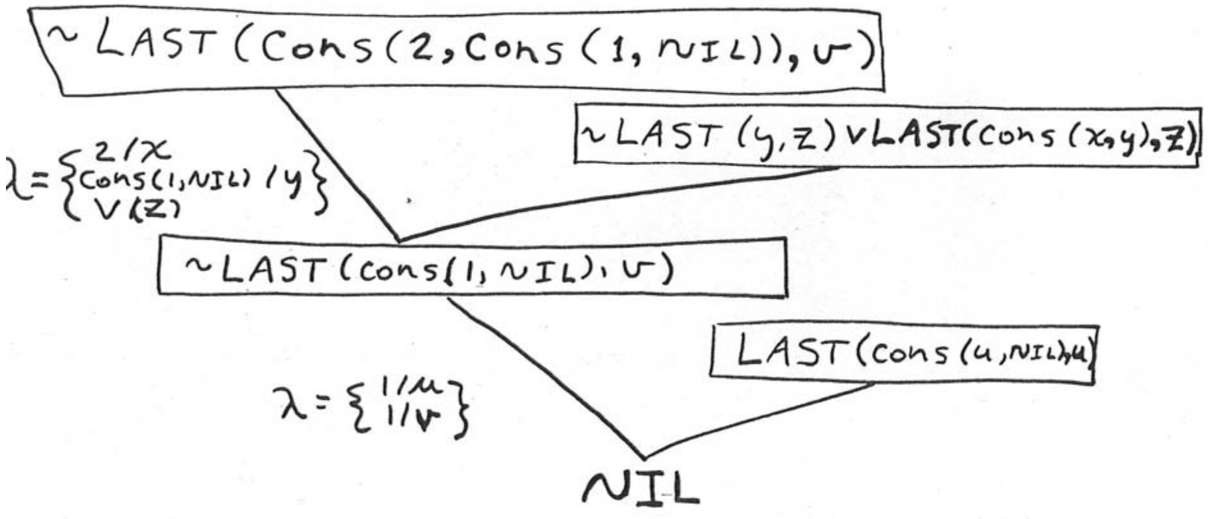
⑧ standardize variables apart

$$\left.\begin{array}{l}\sim Q(x) \vee P(x) \\ Q(z) \\ \sim P(A) \vee \sim P(B)\end{array}\right] \text{ base set}$$

5) given axioms - ① $(\forall u)[Last(cons(u, NIL), u)]$
clause form - $(Last(cons(u, NIL), u))$
② $(\forall x)(\forall y)(\forall z)\ [LAST(y, z) \Rightarrow LAST(cons(x, y), z)]$
clause form $\sim LAST(y, z) \vee LAST(cons(x, y), z)$

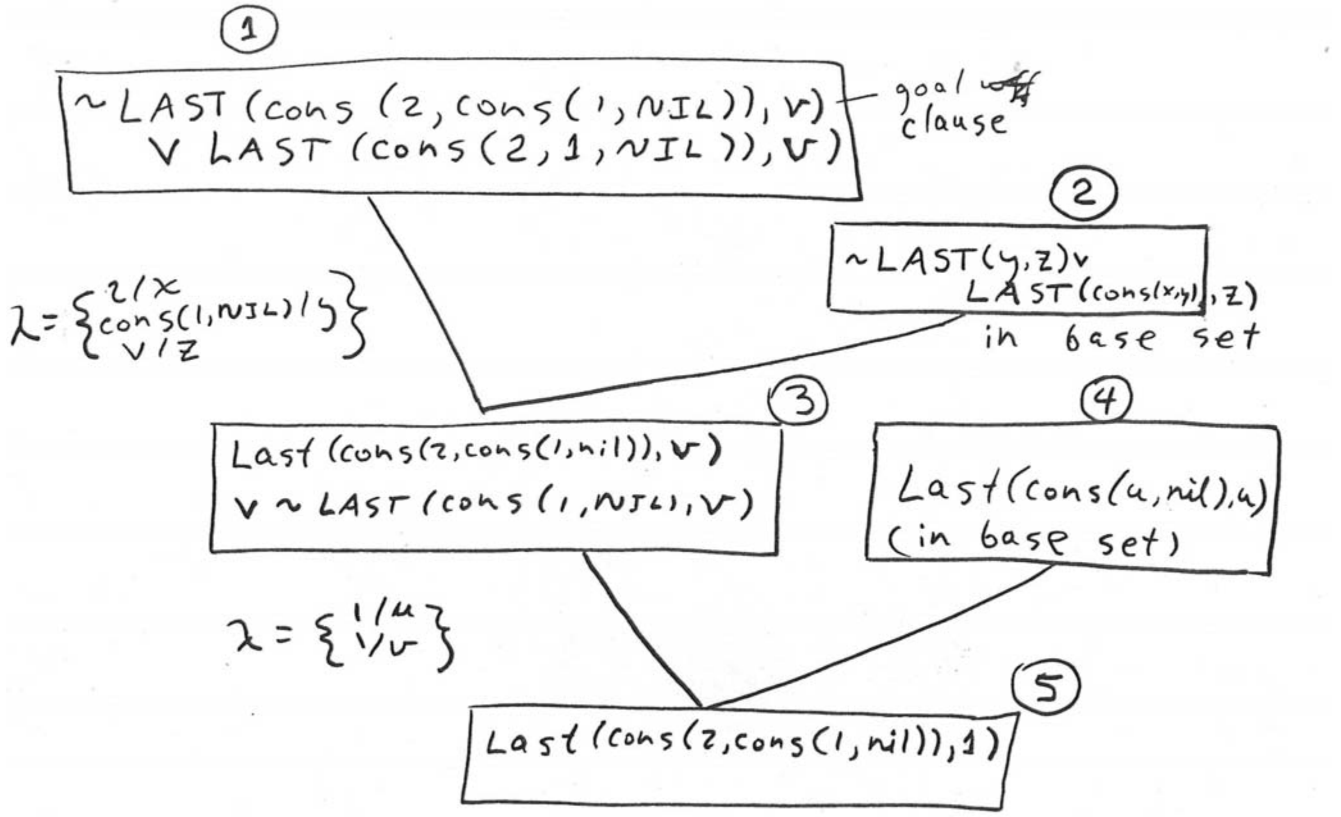
goal - prove $(\exists v)\ LAST(cons(2, cons(1, NIL)), v)$
negate goal wff, add to the base set, and use ~~refu~~ resolution refutation.

$\sim (\exists v)\ LAST(cons(2, cons(1, NIL)), v)$
$(\forall v) \sim LAST(cons(2, cons(1, NIL)), v)$
clause form - $LAST(cons(2, cons(1, NIL)), v)$


~LAST(Cons(2, Cons(1, NIL)), v)
~LAST(y, z) v LAST(Cons(x, y), z)
λ = {2/x, cons(1,NIL)/y, v/z}
~LAST(cons(1, NIL), v)
LAST(cons(u, NIL), u)
λ = {1/u, 1/v}
NIL

answer extraction - create a tautology by combining the goal wff with it's negatation. ie $A \vee \sim A$.
Then follow the same steps as in the proof to find the answer.

reference in text pp. 178-183 section 5.4.2


①
~LAST (cons (2, cons(1, NIL)), v) V LAST (cons(2, 1, NIL)), v)
goal ~~off~~ clause
②
~LAST(y, z) v LAST(cons(x,y)), z)
in base set
λ = {2/x, cons(1,NIL)/y, v/z}
③
Last (cons(2, cons(1, nil)), v) v ~LAST (cons (1, NIL), v)
④
Last(cons(u, nil), u)
(in base set)
λ = {1/u, 1/v}
⑤
Last (cons (2, cons (1, nil)), 1)

Thus the root of the resolution provides us with the answer - the last element of the list is 1.

this method can be used to compute the last element of longer lists. For each additional element in the list, the initial goal clause ① will have to be resolved with clause ② ~~one~~ again in order to remove the first element of the list. After this iterative process is complete, resolution of clauses ③ and ④ will remain much the same, providing the answer at the root clause ⑤.

6) Every two people have a mutual friend

① $(\forall x)(\forall y)(\exists z)[friend(x,z) \wedge friend(y,z)]$

Everyone is either male or female

② $(\forall x)[(male(x) \wedge \sim female(x)) \vee (\sim male(x) \wedge female(x))]$

There exists at least one male and one female

③ $(\exists x)\ male(x) \qquad (\exists x)\ female(x)$

commutative property of friend

④ $(\forall x)(\forall y)[friend(x,y) \Rightarrow friend(y,x)]$

Goal - prove there exists a male and female who are friends.

$(\exists x)(\exists y)[\ male(x) \wedge female(y) \wedge friends(x,y)]$

to prove this wff, negate it and add to the base set. Prove by resolution refutation.

$\sim(\exists x)(\exists y)[male(x) \wedge female(y) \wedge friends(x,y)]$
$(\forall x)\sim(\exists y)[\ldots]$
$(\forall x)(\forall y)\sim[\ldots]$
$(\forall x)(\forall y)[\sim male(x) \vee \sim[female(y) \wedge friends(x,y)]]$
$(\forall x)(\forall y)[\sim male(x) \vee \sim female(y) \vee \sim friends(x,y)]$

C1 $\sim male(x) \vee \sim female(y) \vee \sim friends(x,y)$

Convert ① to clause form
$(\forall x)(\forall y)(\exists z)[friend(x,z) \wedge friend(y,z)]$
$(\forall u)(\forall v)(\exists z)[friend(u,z) \wedge friend(v,z)]$
$(\forall u)(\forall v)[friend(u,g(u,v)) \wedge friend(v,g(u,v))]$

C2 $friend(u,g(u,v))$ C3 $friend(u,g(u,v))$

convert ② to clause form

$(\forall x)\,[(male(x) \wedge \sim female(x)) \vee (\sim male(x) \wedge female(x))]$

new variable w

$(\forall w)\,[(male(w) \wedge \sim female(w)) \vee (\sim male(w) \wedge female(w))$

use distribution rules – $(A \wedge B) \vee (C \wedge D) \Rightarrow (A \vee C) \wedge (B \vee C) \wedge (A \vee D) \wedge (B \vee D)$

$(\forall w)\,(male(w) \vee \sim male(w)) \wedge (\sim female(w) \vee \sim male(w)) \wedge (male(w) \vee female(w)) \wedge (\sim female(w) \vee female($

eliminate tautologies

$(\forall w)\,(\sim female(w) \vee \sim male(w)) \wedge (male(w) \vee female(w$

C4 $\sim female(w) \vee \sim male(w)$

C5 $male(w) \vee female(w)$

Convert ③ to clause form

C6 $male(A)$ C7 $female(B)$

convert ④ to clause form

C8 $\sim friend(z_1, z_2) \vee friend(z_2, z_1)$

<u>Base Set</u>

C1 $\sim male(x) \vee \sim female(y) \vee \sim friend(x, y)$

C2 $friend(u, g(u, v))$

C3 $friend(v, g(u, v))$

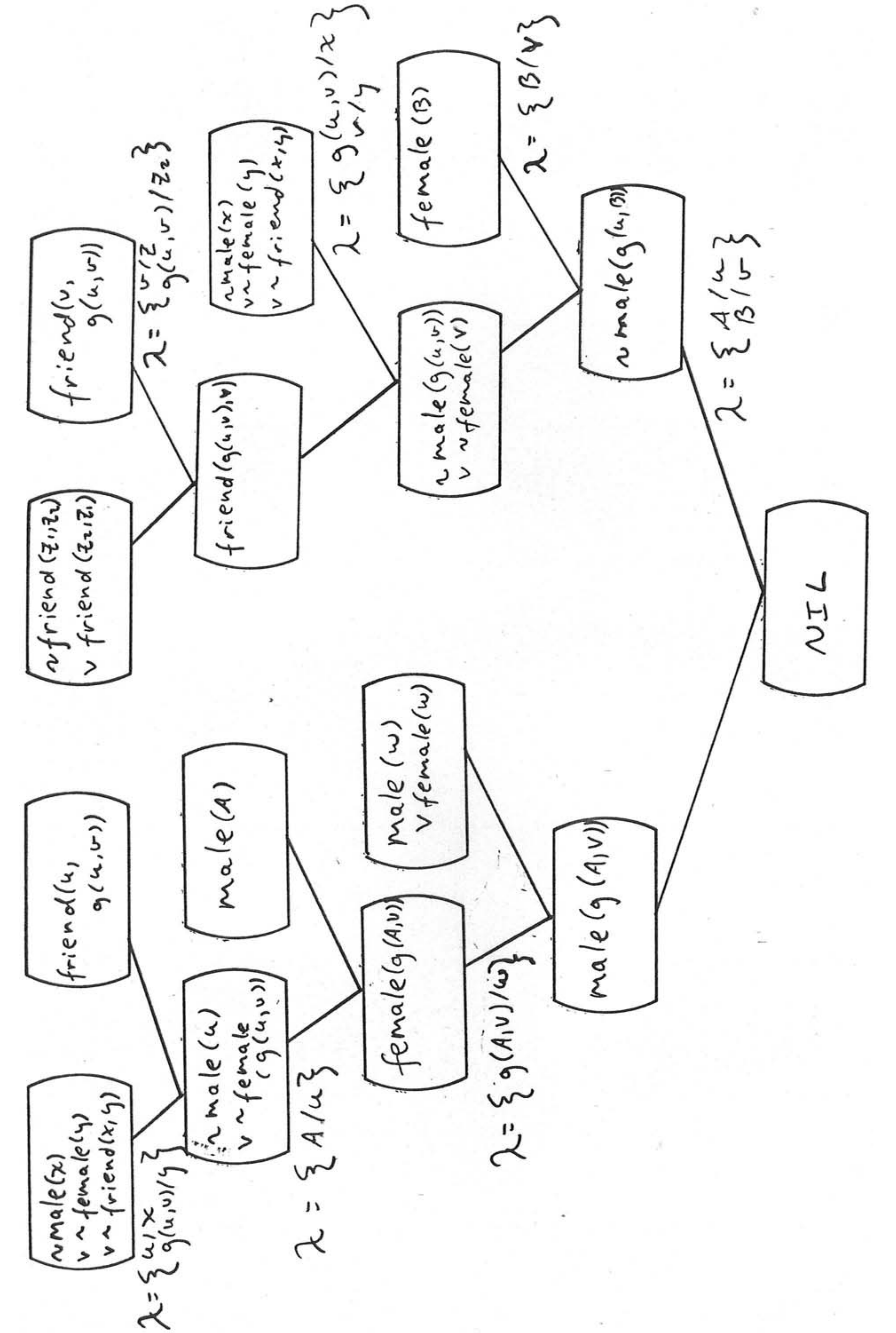
C4 ~~friend(~~ $\sim female(w) \vee \sim male(w)$

C5 $male(w) \vee female(w)$

C6 $male(A)$

C7 $female(B)$

C8 $\sim friend(z_1, z_2) \vee friend(z_2, z_1)$


~friend(z1,z2) v friend(z2,z1)
friend(u, g(u,v))
λ = {v/z, g(u,v)/z2}
friend(g(u,v),v)
~male(x) v ~female(y) v ~friend(x,y)
λ = {g(u,v)/x, v/y}
~male(g(u,v)) v ~female(v)
female(B)
λ = {B/v}
~male(g(u,B))
λ = {A/u, B/v}
NIL
~male(x) v ~female(y) v ~friend(x,y)
friend(u, g(u,v))
λ = {u/x, g(u,v)/y}
~male(u) v ~female(g(u,v))
male(A)
λ = {A/u}
female(g(A,v))
male(w) v female(w)
λ = {g(A,v)/w}
male(g(A,v))

last problem

1. Every two people have a mutual friend.

$(\forall x)(\forall y)(\exists z)[friend(x,z) \land friend(y,z)]$

2. Everyone is either male or female

$(\forall x)[(male(x) \land \neg female(x)) \lor (\neg male(x) \land female(x))]$

3. There exists at least one male and one female.

$(\exists x)\, male(x) \qquad (\exists x)\, female(x)$

4. Theorem: There exists a male and a female who are friends.

$(\exists x)(\exists y)[male(x) \land female(y) \land friend(x,y)]$

Need to know friend is commutative

5. $(\forall x)(\forall y)\, friend(x,y) \rightarrow friend(y,x)$

Negate 4. $\neg(\exists x)(\exists y)[male(x) \land female(y) \land friend(x,y)]$

$(\forall x)\neg(\exists y)[\;\dots\;]$

$(\forall x)(\forall y)\neg[\;\dots\;]$

$(\forall x)(\forall y)\; \neg male(x) \lor \neg female(y) \lor \neg friend(x,y)$

C1 $\neg male(x) \lor \neg female(y) \lor \neg friend(x,y)$

Convert 1 to clause form

$(\forall x)(\forall y)(\exists z)[friend(x,z) \land friend(y,z)]$

$(\forall u)(\forall v)(\exists z)[friend(u,z) \land friend(v,z)]$

$(\forall u)(\forall v)[\text{friend}(u, g(u,v)) \wedge \text{friend}(v, g(u,v))]$

C2 $\text{friend}(u, g(u,v))$

C3 $\text{friend}(v, g(u,v))$

Convert 2 to clause form:

$(\forall x)[(\text{male}(x) \wedge \neg\text{female}(x)) \vee (\neg\text{male}(x) \wedge \text{female}(x))]$

new variable w $(\forall w)[(\text{male}(w) \neg\text{female}(w)) \vee (\neg\text{male}(w) \wedge \text{female}(w))]$

$(A \wedge B) \vee (C \wedge D) \rightarrow (A \vee C) \wedge (B \vee C) \wedge (A \vee D) \wedge (B \vee D)$

$(\forall w)\ (\text{male}(w) \vee \neg\text{male}(w)) \wedge (\neg\text{female}(w) \vee \neg\text{male}(w)) \wedge$

$(\text{male}(w) \vee \text{female}(w)) \wedge (\neg\text{female}(w) \vee \text{female}(w)))$

eliminate tautologies

$(\forall w)\ (\neg\text{female}(w) \vee \neg\text{male}(w)) \wedge (\text{male}(w) \vee \text{female}(w))$

C4 $\neg\text{female}(w) \vee \neg\text{male}(w)$
C5 $\text{male}(w) \vee \text{female}(w)$

Convert 3 to clause form

C6 $\text{male}(A)$ C7 $\text{female}(B)$

Convert 5 to clause form

C8 $\neg\text{friend}(z_1, z_2) \vee \text{friend}(z_2, z$

use Set

C1. $\neg\text{male}(x) \vee \neg\text{female}(y) \vee \neg\text{friend}(x,y)$

C2 $\text{friend}(u, g(u,v))$

C3 $\text{friend}(v, g(u,v))$

C4 $\neg\text{female}(w) \vee \neg\text{male}(w)$

C5 $\text{male}(w) \vee \text{female}(w)$

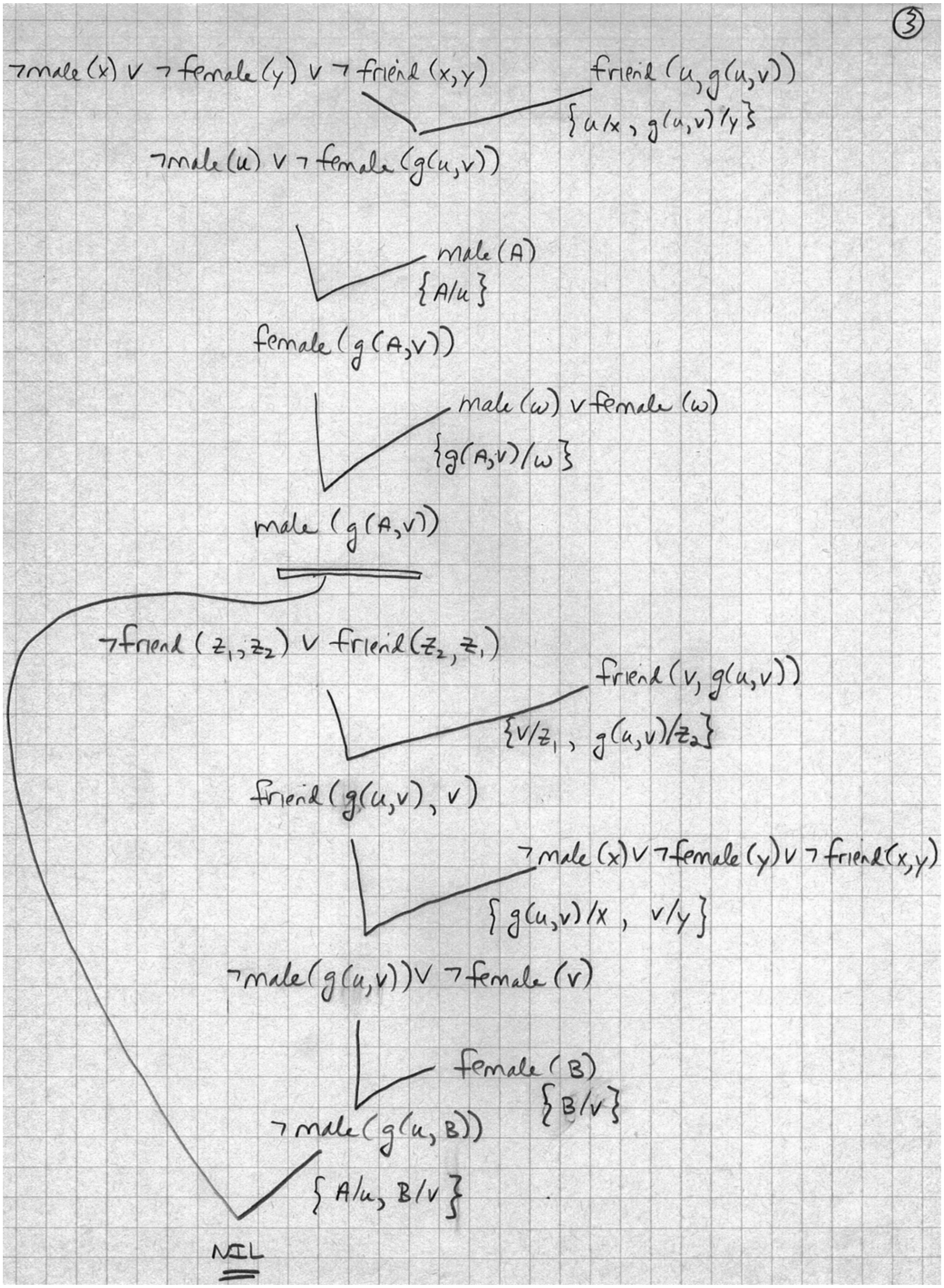
C6 $\text{male}(A)$

C7 $\text{female}(B)$

C8 $\neg\text{friend}(z_1, z_2) \vee \text{friend}(z_2, z_1)$

to be consistent with class notes

friend should be in upper case


3
¬male(x) ∨ ¬female(y) ∨ ¬friend(x,y)
friend(u, g(u,v))
{u/x, g(u,v)/y}
¬male(u) ∨ ¬female(g(u,v))
male(A)
{A/u}
female(g(A,v))
male(w) ∨ female(w)
{g(A,v)/w}
male(g(A,v))
¬friend(z1, z2) ∨ friend(z2, z1)
friend(v, g(u,v))
{v/z1, g(u,v)/z2}
friend(g(u,v), v)
¬male(x) ∨ ¬female(y) ∨ ¬friend(x,y)
{g(u,v)/x, v/y}
¬male(g(u,v)) ∨ ¬female(v)
female(B)
{B/v}
¬male(g(u,B))
{A/u, B/v}
NIL

Due Date: November 16, 1981.

Consider the following problem: There are three cubic blocks, labeled *A, B,* and *C*, lying on a table. The goal is to have them stacked three high with *A* on top, *B* in the middle, and *C* on the bottom (supported by the table). If we define a predicate *ON(x,y)* to mean block *x* is resting on top of block *y,* then the goal can be expressed as $ON(A,B) \wedge ON(B,C)$. The only operator which is available is *place(x,y),* which by definition places block *x* on top of block *y.* Note that *place* is only allowed to move *one* block at a time.

You are to set up an appropriate representation for this problem for each of the following problem-solving methods: state space search, problem reduction search, resolution proof, and **PLANNER**-like program. For the state space search, include a state description, operators for moving from one state to another, and start and goal states. For the problem reduction search, describe a problem description, problem reduction operators for reducing a problem to its constituent subproblems, the main problem or goal, and which subproblems can be considered primitive and why. For the resolution proof, formulate as predicate calculus expressions the facts given and the problem to be solved. Write a **PLANNER**-like program which would solve the problem. Don't worry about the exact syntax that **PLANNER** requires; just follow the basics given in Bobrow and Raphael's "New Programming Languages for Artificial Intelligence Research." If you want to use primitives from some other AI language (e.g., **CONNIVER**, **QLISP**, etc.), feel free to do so. You can also make up your own syntax, as long as you explain exactly how it is to be interpreted in terms of AI language features.

Construct a trace of your **PLANNER**-like program showing the exact order of function calls that it makes as it solves the problem. This solution should include an example of backtracking in the case that *ON(A,B)* is satisfied first and then *ON(B,C)* cannot be achieved. *Note:* You don't have to solve the problem using the other methods; just set the problem up so that they are ready to be used (e.g., set up the appropriate predicates for resolution, but don't actually convert them to clause form or carry out the resolution proof).

Compare the representations and problem-solving methodologies of the four techniques, using the above problem of stacking three blocks to discuss concrete advantages and disadvantages of each technique.

CS 375A Homework Set 7 20 November, 1981

Due Date: November *27,* 1981. (At the movies)

1) Problem 7.4 in Nilsson. (Replacing all constants by parameters is *not* the answer!)

2) Problem 7.5 in Nilsson.

3) Problem 7.8 in Nilsson.

1) The macro-rule:

```
PUT(x,y)

  PRECONDITIONS: ONTABLE(x), CLEAR(x), CLEAR(y), HANDEMPTY, x ≠ y
  ADD LIST: ON(x,y)
  DELETE LIST: ONTABLE(x), CLEAR(y)
```

General Procedure: (This is an approximation of the complete procedure—you weren't expected to work miracles.)

a) Draw the triangle table for the specific instance of the sequence of rules.

	0		

1	*CLEAR(A)
	*ONTABLE(A)
	*HANDEMPTY
		1
		PICKUP (A)	

2	*CLEAR(B)	*HOLDING(A)	2 STACK (A,B)

3			ON(A,B)
			CLEAR(A)
			HANDEMPTY

b) Generalize the triangle table: Replace every occurrence of a constant in the left-most column by a new parameter. (Multiple occurrences of the same constant are replaced by distinct parameters.) Fill in the rest of the table with appropriate add clauses assuming completely uninstantiated operators (i.e., as the add clauses appear in the operator descriptions).

	0		

1	*CLEAR(p1)
	*ONTABLE(p2)
	*HANDEMPTY
		1
		PICKUP (p4)	

2	*CLEAR(p3)	*HOLDING(p4)	2 STACK (p5,p6)

3			ON(p5,p6)
			CLEAR(p5)
			HANDEMPTY

This results in a table which is too general, so ...

c) Redo each operator's precondition proof using the precondition formulas from the operator descriptions as the theorems to be proved and the support clauses in the generalized table as axioms. The new proofs are done in exactly the same way as the original STRIPS proofs (using the same resolutions). This ensures that the original table is an instance of the generalized table.

```
PICKUP

Negation of Theorem: ¬ONTABLE(p4)∨¬CLEAR(p4)∨¬HANDEMPTY
Axiom: CLEAR(p1)    {p1/p4}
          ¬ONTABLE(p1)∨¬HANDEMPTY
Axiom: ONTABLE(p2)  {p1/p2}
          ¬HANDEMPTY
Axiom: HANDEMPTY
          NIL
```

Make the substitutions in the table.

	0		

1	*CLEAR(p1)
	*ONTABLE(p1)
	*HANDEMPTY
		1
		PICKUP (p1)	

2	*CLEAR(p3)	*HOLDING(p1)	2 STACK (p5,p6)

3			ON(p5,p6)
			CLEAR(p5)
			HANDEMPTY

Now consider STACK

```
Negation of Theorem: ¬HOLDING(p5)∨¬CLEAR(p6)
Axiom: CLEAR(p3)     {p3/p6}
          ¬HOLDING(p5)
Axiom: HOLDING(p1)  {p1/p5}
          NIL
```

Again, make the substitutions in the table.

	0		

1	*CLEAR(p1)
	*ONTABLE(p1)
	*HANDEMPTY
		1
		PICKUP (p1)	

2	*CLEAR(p3)	*HOLDING(p1)	2 STACK (p1,p3)

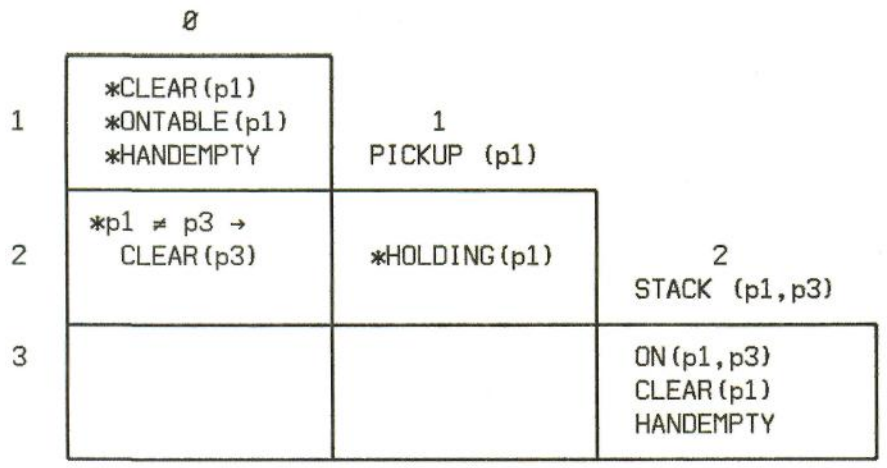
3			ON(p1,p3)
			CLEAR(p1)
			HANDEMPTY

d) Refinement 1: (Not needed here.) Recognize cases where two parameters are produced from a single occurrence of a constant in a single clause; if both such parameters do not appear as arguments of operators in the plan, then they can be bound together and one substituted for the other throughout the table without effectively inhibiting the generality of the plan. This avoids some overgeneralization that might lead to two clauses of the form INROOM(p1,p2) and INR00M(p1,p4) in column 0 (i.e., a plan whose preconditions allow object p1 to be in two distinct rooms at the same time.

e) Delete List Refinement: Deletions in the generalized table are not always the same as in the original table. For example, if p1=p3, you must delete the clause CLEAR(p3) when you pick up the block. To get around this problem, the generalized clause should be

`p1 ≠ p3 → CLEAR (p3)`

So the final table is



	0	1	2

1	*CLEAR(p1)
	*ONTABLE(p1)
	*HANDEMPTY
		PICKUP (p1)	

2	*p1 ≠ p3 → CLEAR(p3)	*HOLDING(p1)	STACK (p1,p3)

3			ON(p1,p3)
			CLEAR(p1)
			HANDEMPTY

2) For certain operators it is convenient to be able merely to specify the *form* of clauses to be deleted. We can do this with wild cards. For example, delete `ABOVE(x,$)` where `$` is matched to anything.

In general, however, it may not be possible to explicitly name all the atoms that should appear in the delete list. Some clauses may be *derived* from other clauses (e.g., `ABOVE`). One way to deal with the problem is to define a set of primitive predicates (e.g., `ON`) and relate all other predicates to this primitive set. In particular, require the delete list of an operator description to indicate all the atoms containing primitive predicates that should be deleted when the operator is applied. Also require that any nonprimitive clause in the world model have associated with it those primitive clauses on which its validity depends. (A primitive clause is one which contains only primitive predicates.) For example, the clause `ABOVE(B1,B2)` would have associated with it the clause `ON(B1,B2)`.

By using these conventions, we can be assured that primitive clauses will be correctly deleted during operator applications, and that the validity of nonprimitive clauses can be determined whenever they are used in a deduction by checking to see if all the primitive clauses on which the nonprimitive clause depends are still in the world model.

3) Monkeys and Bananas (revisited)

Initial Model:

```
¬ONBOX, AT(BOX,B), AT(MONKEY,A), ¬HAS-BANANAS
BANANAS hang out of reach above position C
```

Operators:

`GOTO(u)`

```
preconditions: ¬ONBOX
add-list: AT(MONKEY,u)
delete-list: AT(MONKEY,$)
```

```
PUSHBOX(v)

   preconditions: ¬ONBOX, (∃x)[AT(MONKEY,x)∧AT(BOX,x)]
   add-list: AT(MONKEY,v), AT(BOX,v)
   delete-list: AT(MONKEY,$), AT(BOX,$)

CLIMBBOX

   preconditions: ¬ONBOX, (∃x)[AT(MONKEY,x)∧AT(BOX,x)]
   add-list: ONBOX
   delete-list: ¬ONBOX

GRASP

   preconditions: ONBOX, AT(BOX,C)
   add-list: HAS-BANANAS
   delete-list: ¬HAS-BANANAS
```

Due Date: December 4, 1981. (In class)

1) Consider the following grammar:

```
S → NP VP
NP → DETERMINER NOUN
VP → VERB/TRANSITIVE NP
VP → VERB/INTRANSITIVE
```

Modify the grammar to allow for adjectives, adverbs, and prepositional phrases. Write either a finite transition net or a flow chart for accepting the grammar you have defined. Using your grammar, draw the parse tree for the sentence, “A small brown dog quickly ate the well marinated meat beside the kitchen stove.”

2) Give a grammar for recognizing simple questions such as “Where are we?” You only need worry about simple questions involving “where”, “who”, or “what.” Note that this can be an open-ended problem so don’t try too much. Don’t worry about prepositional phrases or other complications unless you feel ambitious.

3) Here are a vocabulary and grammar used in a mythical speech understanding system. The vocabulary consists of the following set of words:

```
{monkeys, programs, termites, climb, eat, manipulate, search, bananas, bits, trees}
```

The grammar consists of the following productions:

```
<sentence> → <subject> <verb> <object>
<subject> → monkeys | programs | termites
<verb> → climb | eat | manipulate | search
<object> → bananas | bits | trees
```

Assume that for small vocabularies (≤ 25 words) the probability of recognizing a particular word incorrectly, p_e, varies linearly with the size of the vocabulary, $|V|$. (Note that the probability of recognizing a particular word correctly, p_c, is then just $1 - p_e$.) In particular, we shall assume that the following equation holds:

$$p_e = 0.0375(|V| - 1), \qquad 1 < |V| < 25$$

Also assume for this problem that the lexical segmentation scheme of the speech understanding system works perfectly, so that the only source of error lies in the word recognition process.

a) Without the use of syntax (i.e., the grammar above), what is the probability of correct sentence recognition (all words recognized correctly) on three-word sentences?

b) Now, using the grammar, but without any semantics, what is the probability of correct sentence recognition?

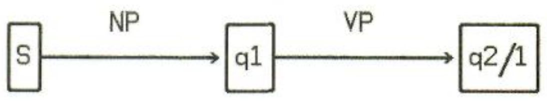
c) Now specify which sentences generated by the grammar you consider to be semantically meaningful. (Any reasonable assumption is fine.) Discuss why the sentence recognition rate should be better using both syntax and semantics, and make a rough estimate of the expected probability of correct sentence recognition in this case. (Precise calculations aren’t necessary.)

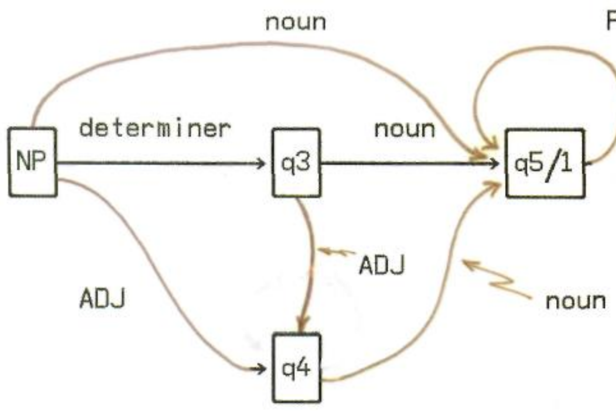
Problem 1

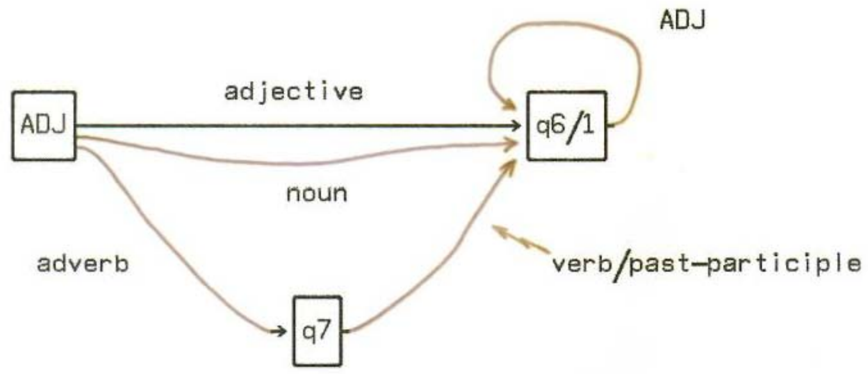
a) The (context free) grammar: to handle adjectives (including *describers* [in this case adverbs modifying past participles] and *classifiers* [in this case nouns]), adverbs, and prepositional phrases.

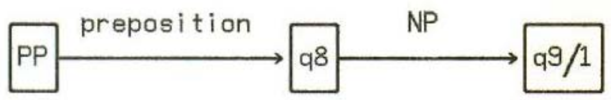
```
S → NP VP
NP → noun
NP → determiner NP1
NP1 → noun
NP1 → ADJ* noun        (kleene "*" – repetition)
NP1 → ADJ* noun PP*
NP1 → noun PP*

PP → preposition NP

ADJ → adjective
ADJ → adverb verb/past-participle
ADJ → noun

VP → verb/intransitive
VP → adverb* verb/intransitive
VP → verb/intransitive adverb*
VP → verb/intransitive PP*
VP → adverb* verb/intransitive PP*
VP → verb/intransitive adverb* PP*
VP → verb/intransitive PP* adverb*
VP → verb/transitive
VP → adverb* verb/transitive
VP → verb/transitive adverb*
VP → verb/transitive NP
VP → adverb* verb/transitive NP
VP → verb/transitive adverb* NP
VP → verb/transitive NP adverb*
```

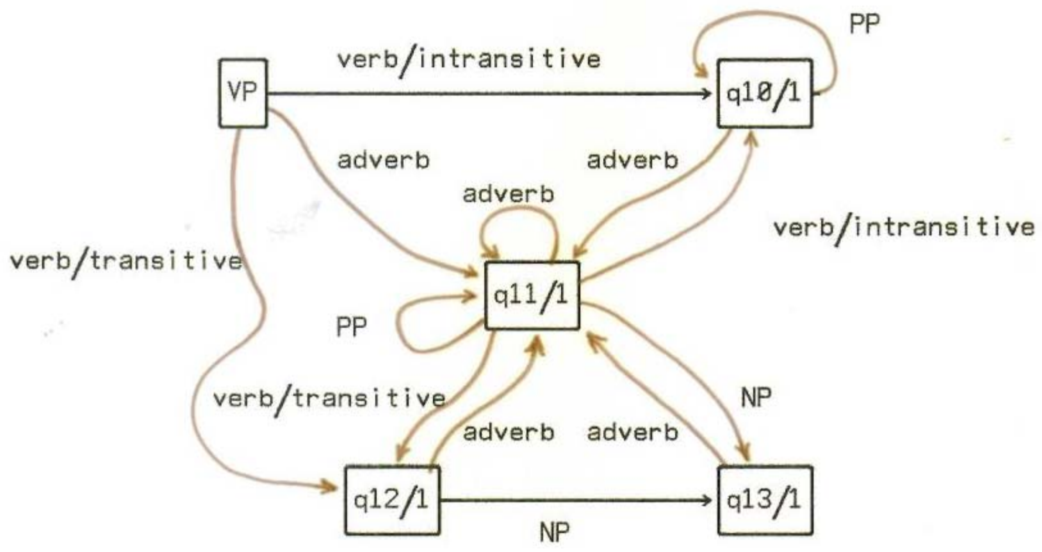
b) Transition net: A call to a network constitutes an implicit PUSH. Nodes are labeled with names inside. Names of the form "q/1" are used for terminal nodes—nodes at which a POP occurs.



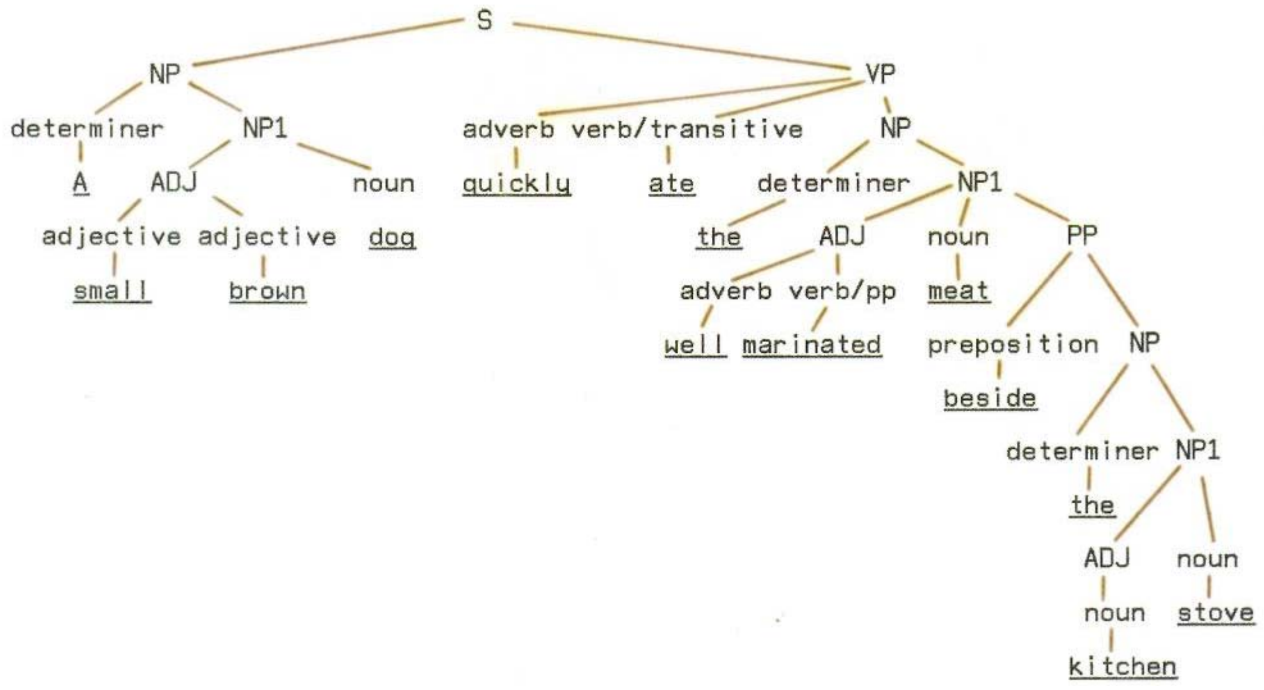








c) The parse tree for “A small brown dog quickly ate the well marinated meat beside the kitchen stove.”



Problem 2

A grammar for simple "what, where, who" questions, including prepositional phrases, pronouns, and auxiliary verbs – adverbs have been omitted. The grammar will handle examples like "*Who is it?*", "*Who can see?*", "*Who can you see by the cliff?",* and "*What time is it?*".

```
S → VP NP0
S → Adverb NP VP        adverb → Where | what | who
S → VP

NP0 → Pronoun
NP0 → NP
NP → Noun
NP → Determiner NP1
NP1 → Noun
NP1 → ADJ* noun
NP1 → ADJ* noun PP*
NP1 → Noun PP*

PP → Preposition NP

ADJ → Adjective
ADJ → Adverb verb/past-participle
ADJ → Noun

VP → Verb/intransitive
VP → Verb/transitive NP
VP → Auxiliary verb/intransitive
VP → Auxiliary verb/transitive NP
VP → Verb/ intransitive PP*
VP → Auxiliary verb/intransitive PP*
VP → Auxiliary NP0 VP
```

It should be noted that in general, questions can be viewed as sentences flipped inside out with some slot in the sentence missing. The question is then interpreted as a request to fill the slot. Thus a full purpose meaningful grammar for questions is very complex and usually not very useful. Often, one would much prefer to view questions as this type of sentence with a slot missing since this tends to make the semantics of the question much clearer.

Problem 3

a) For each word,

$$p_c = 1 - p_e = 1 - 0.0375 \cdot (10 - 1) = 0.6625$$

So the probability of correctly recognizing all words in a three word sentence is,

$$p_{c_3} = (p_c)^3 = 0.2908$$

b) Using syntax only,

1) The probability of correctly recognizing the <u>subject</u> is,

$$p_{c_s} = 1 - 0.375 \cdot (3 - 1) = 0.925$$

2) The probability of correctly recognizing the <u>verb</u> is,

$$p_{c_v} = 1 - 0.375 \cdot (4 - 1) = 0.8875$$

3) The probability of correctly recognizing the <u>object</u> is,

$$p_{c_o} = 1 - 0.375 \cdot (3 - 1) = 0.925$$

So,
$p_{c_{overall}} = p_{c_s} \cdot p_{c_v} \cdot p_{c_o} = 0.7594$

c) To consider semantics in this small system, we can define some possibly meaningful sentences:

```
1. monkeys climb trees
2. monkeys eat bananas
3. programs manipulate bits
4. programs search trees
5. termites climb trees
6. termites eat trees
```

In the first place, there are fewer valid sentences than before, so the recognition scores will be higher. For example, once a subject has been recognized, it places constraints on the possible verb and object. Similarly, recognition of the verb further constrains the choice of object. Because there are fewer possibilities to consider at each decision point, the recognition probabilities increase.

There are three possible subjects, so $p_{c_s} = 0.925$ as before, but once a subject is chosen, there are only two possible verbs so $p_{c_v} = 1 - 0.0375 \cdot (2 - 1) = 0.9625.$ Once both subject and verb are recognized there is only one possible object. It serves as a check word.

So,

$p_{c_{overall}} = p_{c_s} \cdot p_{c_v} = 0.8903$

Department of Mathematics, Statistics, and Computing Science
Faculty of Arts and Science
Dalhousie University

Computer Science 375A

Final Examination — Tuesday, December 15, 1981
Dr. R. G. Smith — 3:30 - 6:30

BOOKS AND NOTES ARE ALLOWED

Instructions: Answer *all* questions. The point value of each question corresponds roughly with the time estimate associated with it. Think carefully through your answers *before* writing—*Be succinct,* make specific points and back them up with clear arguments and examples. In general, a concise, one page answer is worth more points than a rambling four page answer.

Good Luck!

1) A* and the Traveling Salesman problem. (20 minutes)

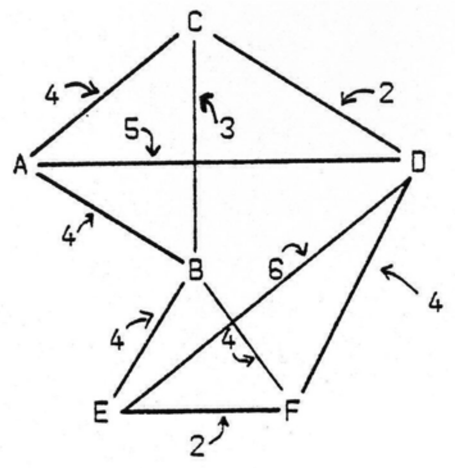
In the graph below, a salesman must plan a trip so that he starts at city A and visits each of the cities just once and returns to city A.

A. (5 minutes) Characterize a production system for solving the traveling salesman problem. Specifically, describe the database and production rules, along with the initial state and goal condition. The control strategy for this production system will be A*.

B. (10 minutes) Illustrate the application of A* to the traveling salesman problem. Draw the search tree corresponding to the first 4 node expansions in the search, assuming the map shown below. Label each node in the search tree with its h value, f value, and the step during which it is expanded. Use the following h function to generate the search. (Note that this function is not admissible.)

$$h = C_{min} \cdot (1 + N_u)$$

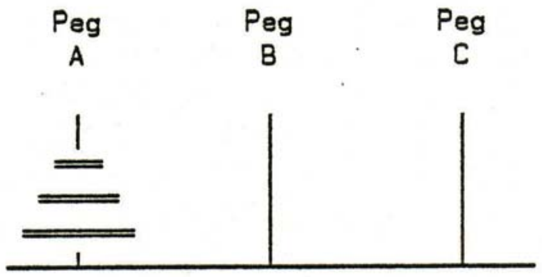
Where: C_{min} = the cost of the lowest cost edge leading from the current node to an unvisited node, and N_u = the number of unvisited nodes.



C. (5 minutes) Describe a better, admissible h function for applying A* to the traveling salesman problem. Explain why your h is admissible.

2) Means-Ends Analysis (25 minutes)

Consider the Towers of Hanoi problem: Three disks of different sizes begin on peg A as shown. The objective is to move all three to peg C under the following constraints: (i) only one disk can be moved at a time, (ii) only the top disk on any given peg can be moved, and (iii) no disk can ever be placed on top of a smaller disk.



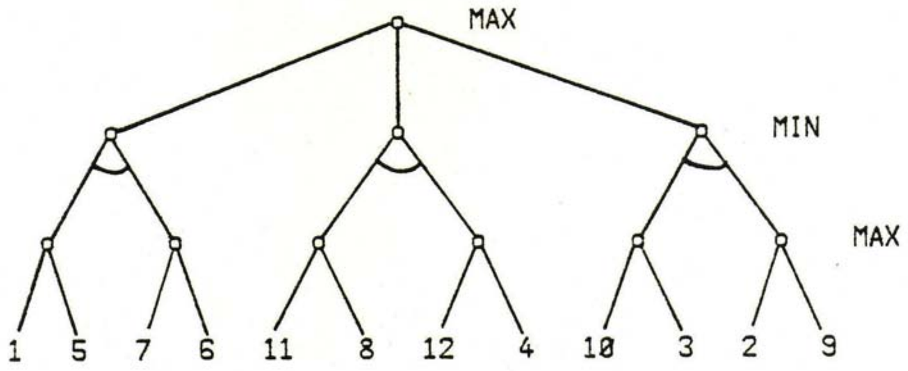
A. (5 minutes) Specify the database and production rules for a production system to solve the Towers of Hanoi problem. Use predicate calculus literals to specify the database and STRIPS-style production rules.

B. (13 minutes) Illustrate a means-ends (problem reduction) approach to solving this problem, by tracing the first several steps in the means-ends search for a solution. Discuss any difficulties that arise. Is means-ends analysis a good strategy for solving this problem? Is forward search a good strategy?

C. (7 minutes) Point out the advantages of using forward search and the advantages of using means-ends analysis for certain classes of problems that you can think of. Suggest a problem that is *not* discussed in our textbook or our homework sets, which is best handled by forward search, and one which is best handled by means-ends analysis.

3) Alpha/Beta Search of AND/OR Trees (25 minutes)

Consider the AND/OR game tree below, assuming that the root node corresponds to a game position in which player MAX is about to move. Assume that MAX would like to find the move which will lead to the highest-valued position three moves into the future.



A. (11 minutes) Circle *all* of the nodes that will *not* be examined by an Alpha/Beta search of this game tree, assuming the search is from left to right.

For B and C below, consider all search trees with exactly this structure, but where the values 1 through 12 can be reassigned to the tip nodes (provided each of the 12 values is used exactly once).

B. (7 minutes) What is the *maximum* number of nodes (i.e., worst case) that will be examined by an Alpha/Beta search for trees with exactly this structure, but with values possibly reassigned to the tip nodes? Give an assignment of the values 1 through 12 to the tip nodes, for which this worst case pruning will result. Is your solution unique?

C. (7 minutes) What is the *minimum* number of nodes (i.e., best case) that must be examined by Alpha/Beta for a tree with this structure? Give an assignment of the values 1 through 12 to tip nodes, for which this best case pruning will result. Is your solution unique?

4) Predicate Calculus Theorem Proving (35 minutes)

A. (6 minutes) Make up an appropriate set of predicates, functions, and constants, and express each of the following statements as one or more statements in first order predicate calculus.

1) Santa lives at the North Pole.
2) Everyone who attends Dalhousie lives in Nova Scotia.
3) Santa does not attend Dalhousie.
4) Santa's wife knows someone who attends Dalhousie.
5) People know everyone that their wife knows.
6) Everyone lives somewhere.

B. (4 minutes) Transform each of the above statements to clause form.

C. (15 minutes) Give a resolution proof that Santa knows someone who lives in Nova Scotia.

D. (10 minutes) Statement 3 cannot be proven from statements 1 and 2. What is the problem? Give one or more predicate calculus assertions which, together with assertions 1 and 2 allow a proof of statement 3. (Note that you don't have to give the proof—just the additional assertions.) Add general assertions, rather than very specific ones (i.e., don't add "Santa does not attend Dalhousie.").

5) Application Areas (15 minutes)

Choose one of the following areas that we covered in class: Natural Language Understanding (text), Speech Understanding, Machine Vision, and Machine Learning.

1) (6 minutes) Characterize briefly the progress that has been made in this area, and describe the main ideas/methods responsible for this progress.

2) (9 minutes) Describe the problems that you feel are the bottlenecks holding up

progress in this area. Be as specific as possible, giving examples of abilities that systems do not currently have, and problems for which current methods are not well suited.

6) AI Systems and Methods (15 minutes)

    1) Give a one or two sentence definition of each of the following:

        1) Satisfiability
        2) Validity
        3) Bi-directional search
        4) Semantic Network
        5) Frame
        6) Demon
        7) Heuristic

    2) For each of the following, give a one sentence description of what it does, and a one or two sentence characterization of the main idea/method involved in how it does it.

        1) ABSTRIPS
        2) AM
        3) SHRDLU
        4) HEARSAY II
        5) Winston's "Arch" Learning Program

7) AI terms. (10 minutes)

Match each item in column A with the one from column B that is most related to it.

A	B
(1) AND/OR Trees	(1) COBOL
(2) add and delete lists	(2) backward reasoning
(3) consequent theorems	(3) effects of rules
(4) cooperating knowledge sources	(4) forward reasoning
(5) symbol-mapping problem	(5) HEARSAY-II
(6) Skolem functions	(6) inheritance of properties
(7) unification	(7) matching
(8) set-of-support	(8) problem reduction
	(9) removal of existential quantifiers
	(10) resolution strategy