

# **Computer Science Comprehensive Examinations 1981/82-1984/85**

edited by

Arthur M. Keller

**Department of Computer Science**

Stanford University  
Stanford, CA 94305





**Computer Science  
Comprehensive Examinations  
1981/82–1984/85**

by  
the faculty and students  
of the Computer Science Department  
of Stanford University

edited by  
Arthur M. Keller

**Abstract**

This report is a collection of the eight comprehensive examinations from Winter 1982 through Spring 1985 prepared by the faculty and students of Stanford's Computer Science Department, together with solutions to the problems posed.



## Preface

In November 1978, Frank Liang published the first collection of Computer Science Department Comprehensive Examinations, STAN-CS-78-677, and the document proved to be a tremendous success. Consequently, in August 1981, Carolyn Tajnai published the second collection, STAN-CS-81-869.

The examinations in this booklet have undergone only a little editing. The sections of each exam have been placed in a standard order, with all the questions preceding all the answers.

The comprehensive examination serves several purposes in the department. For the Ph.D. student it serves as a "Rite of Passage"; the exam must be passed at the Ph.D. level by the end of six quarters of full-time study (excluding summers) for the student to continue in the program. Passing the *Comp* at the Ph.D. level is the breadth requirement for the degree. The Ph.D. Minor student must pass at the Master's level to be eligible for the degree. Master's students admitted starting Autumn 1984 or electing the new requirements no longer have the **Comp** as a requirement of their degree. Master's students admitted before Autumn 1984 and electing the old requirements must pass the **Comp** at the Master's level.

The written portion is a six-hour examination given winter and spring quarters. Until Spring 1982, the written exam was "open book and notes" and was a single six-hour exam (with a break in the middle for lunch). Students were allowed to allocate the six hours as they saw fit. Beginning Winter 1983, the six parts of the exam were given as separate one-hour "closed book" exams. By having "closed book" exams, more fundamental questions could be asked. Grading remained unchanged: For a Ph.D. pass, both a minimum score in each area and a sufficiently high overall score were needed; for a Master's pass, only a passing overall score was needed. The *Comprehensive Examination Reading List* is the syllabus for the written exam.

I would like to acknowledge the efforts of Kathy Berg, Victoria Cheadle, Mary Drake, Don Knuth, Betty Scott, Carolyn Tajnai, Marilynn Walker, Phyllis Winkler, and the Comprehensive Examination Committees in producing this report. This publication has been supported in part by NSF grant DCR 83-00984.

Arthur M. Keller  
August 1985



**to the** students  
who take and **prepare**  
**the** Comprehensive Exam



## Table of Contents

<b>Preface</b> . . . . .	iii
<b>Table of Contents</b> . . . . .	vii
<b>Comprehensive Examination Reading List</b> . . . . .	xi
 <b>Winter 1982</b>	
Written Examination	
Analysis of Algorithms . . . . .	1
Artificial Intelligence . . . . .	4
Hardware Systems . . . . .	6
Numerical Analysis . . . . .	9
Software Systems . . . . .	11
Theory of Computation . . . . .	13
Solution	
Analysis of Algorithms . . . . .	14
Artificial Intelligence . . . . .	17
Hardware Systems . . . . .	20
Numerical Analysis . . . . .	25
Software Systems . . . . .	31
Theory of Computation . . . . .	34
 <b>Spring 1982</b>	
Written Examination	
Analysis of Algorithms . . . . .	37
Artificial Intelligence . . . . .	40
Hardware Systems . . . . .	42
Numerical Analysis . . . . .	46
Software Systems . . . . .	49
Theory of Computation . . . . .	52
Solution	
Analysis of Algorithms . . . . .	54
Artificial Intelligence . . . . .	57
Hardware Systems . . . . .	62
Numerical Analysis . . . . .	67
Software Systems . . . . .	76
Theory of Computation . . . . .	82

## Winter 1983

Written Examination	
Analysis of Algorithms . . . . .	85
Artificial Intelligence . . . . .	87
Hardware Systems . . . . .	88
Numerical Analysis . . . . .	91
Software Systems . . . . .	93
Theory of Computation . . . . .	97
Solution	
Analysis of Algorithms . . . . .	99
Artificial Intelligence . . . . .	102
Hardware Systems . . . . .	105
Numerical Analysis . . . . .	108
Software Systems . . . . .	111
Theory of Computation . . . . .	114

## Spring 1983

Written Examination	
Analysis of Algorithms . . . . .	117
Artificial Intelligence . . . . .	120
Hardware Systems . . . . .	122
Numerical Analysis . . . . .	124
Software Systems . . . . .	126
Theory of Computation . . . . .	129
Solution	
Analysis of Algorithms . . . . .	130
Artificial Intelligence . . . . .	132
Hardware Systems . . . . .	134
Numerical Analysis . . . . .	137
Software Systems . . . . .	139
Theory of Computation . . . . .	142

## Winter 1984

Written Examination	
Analysis of Algorithms . . . . .	145
Artificial Intelligence . . . . .	147
Hardware Systems . . . . .	149
Numerical Analysis . . . . .	153
Software Systems . . . . .	155
Theory of Computation . . . . .	158
Solution	
Analysis of Algorithms . . . . .	159
Artificial Intelligence . . . . .	160
Hardware Systems . . . . .	164
Numerical Analysis . . . . .	171
Software Systems . . . . .	174
Theory of Computation . . . . .	181

## Spring 1984

Written Examination	
Analysis of Algorithms . . . . .	185
Artificial Intelligence . . . . .	188
Hardware Systems . . . . .	190
Numerical Analysis . . . . .	194
Software Systems . . . . .	196
Theory of Computation . . . . .	200
Solution	
Analysis of Algorithms . . . . .	202
Artificial Intelligence . . . . .	204
<b>Hardware Systems</b> . . . . .	206
Numerical Analysis . . . . .	211
Software Systems . . . . .	215
Theory of Computation . . . . .	221

**Winter 1985**

Written Examination	
Analysis of Algorithms . . . . .	<b>225</b>
Artificial Intelligence . . . . .	<b>228</b>
Hardware Systems . . . . .	<b>230</b>
Numerical Analysis . . . . .	<b>233</b>
Software Systems . . . . .	<b>235</b>
Theory of Computation . . . . .	<b>238</b>
Solution	
Analysis of Algorithms . . . . .	<b>240</b>
Artificial Intelligence . . . . .	<b>243</b>
Hardware Systems . . . . .	<b>248</b>
Numerical Analysis . . . . .	<b>251</b>
Software Systems . . . . .	<b>253</b>
Theory of Computation . . . . .	<b>257</b>

**Spring 1985**

Written Examination	
Analysis of Algorithms . . . . .	<b>261</b>
Artificial Intelligence . . . . .	<b>264</b>
Hardware Systems . . . . .	<b>266</b>
Numerical Analysis . . . . .	<b>268</b>
Software Systems . . . . .	<b>270</b>
Theory of Computation . . . . .	<b>273</b>
Solution	
Analysis of Algorithms . . . . .	<b>275</b>
Artificial Intelligence . . . . .	<b>279</b>
Hardware Systems . . . . .	<b>283</b>
Numerical Analysis . . . . .	<b>287</b>
Software Systems . . . . .	<b>289</b>
Theory of Computation . . . . .	<b>293</b>

**Comprehensive Examination Reading List**  
(Revised July 12, 1985)

**ANALYSIS OF ALGORITHMS**

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

Michael R. Garey and David S. Johnson, *Computers and Intractability*, Freeman, 1979, Chapters 1-3.

Donald E. Knuth, *The Art of Computer Programming*, Volume 1, Second Edition, Addison-Wesley, 1973, Section 1.2 (except for Subsections 1.2.9, 1.2.10, 1.2.11.2, and 1.2.11.3).

**ARTIFICIAL INTELLIGENCE**

Elaine Rich, *Artificial Intelligence*, McGraw-Hill, 1983.

**HARDWARE SYSTEMS**

M. Morris Mano, *Computer System Architecture*, Second Edition, Prentice-Hall, 1982.

John F. Wakerly, *Microcomputer Architecture and Programming*, Wiley, 1981. Chapters 10, 11, 13, 14. Memorization of chapters 13 and 14 is not required.

**NUMERICAL ANALYSIS**

Kendall E. Atkinson, *An Introduction to Numerical Analysis*, Wiley, 1978, Chapters 1-3, 5, 7, 8 (except Sections 2.8, 2.10, 5.4).

**SOFTWARE SYSTEMS**

Alfred V. Aho, and Jeffrey D. Ullman, *Principles of Compiler Design*, Addison-Wesley, 1977. All except Chapters 12-14.

Terrence W. Pratt, *Programming Languages: Design and Implementation*, Second edition, Prentice-Hall, 1984.

James L. Peterson and Abraham Silberschatz, *Operating System Concepts*, Addison-Wesley, 1983, Chapters 1-12.

## **THEORY OF COMPUTATION**

John E. Hopcroft and Jeffrey D. Ullman, Introduction *to* Automata, ***Theory, Languages, and*** Computation, Addison-Wesley, 1979, Chapters 1-3, 4.1-4.6, 5-7, 8.1-8.5.

Michael R. Garey and David S. Johnson, Computers and Intractability, Freeman, 1979, Chapter 7.

Zohar Manna, Introduction to Mathematical *Theory* of Computation, McGraw-Hill, 1973, Chapters 1-3.

John McCarthy and Carolyn Talcott, LISP: Programming and Proving, (available from McCarthy's secretary) 1980, Chapters 1-3.

Herbert B. Enderton, A Mathematical Introduction *to Logic*, Academic Press, 1972, Chapters 1-2.

## **PREPARATION**

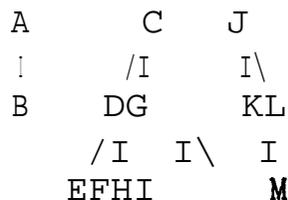
The document "Charge to the Comprehensive Exam Committee," available from the department office, describes general policy. Previous exams, also available from the department office, provide a good source from which to study.

The comprehensive exam is meant, generally to cover the material from the following courses: 260, 261, 262 (algorithms and data structures); 223 [520 is recommended in addition] (artificial intelligence); 108, 112, 212 (hardware systems); 237A (numerical analysis); 242, 243, 246 [346 is recommended in addition] (software systems); and 254, 257A [306 is recommended in addition] (theory of computation). Since the precise content of these courses varies, the actual scope of the exam will be determined by the references above.

The exam will also assume a certain mathematical sophistication and a knowledge of programming. Proofs of correctness for simple programs may be required. The programming knowledge required will be Pascal, LISP, and an assembly language. Nonsmoking and smoking examination rooms will be scheduled.

**Problem 1. (30 points)**

Consider a forest that is represented with conventional 'left child' and 'right sibling' pointers. For example, the forest



would be represented in two arrays *left* and *right* as follows:

$$\begin{array}{rcccccccccccccc}
 x & = & A & B & C & D & E & F & G & H & I & J & K & L & M \\
 left[x] & = & B & \Lambda & D & E & \Lambda & \Lambda & H & \Lambda & \Lambda & K & \Lambda & M & \Lambda \\
 right[x] & = & C & \Lambda & J & G & F & \Lambda & \Lambda & I & \Lambda & \Lambda & L & \Lambda & \Lambda
 \end{array}$$

(Upper case letters stand for pointers to nodes in the forest.) Everything is accessible when the root of the leftmost tree, in this case A, is given.

Now consider a similar representation in which we have 'right cousin' pointers instead. This means that  $right[x]$  is the next node to the right and on the same level as  $x$ , and  $right[x] = \Lambda$  if and only if  $x$  is the rightmost node on its level. In the above example, we would have  $right[B] = D$ ,  $right[G] = K$ ,  $right[F] = H$ ,  $right[I] = M$ , and all other values of *left* and *right* would be the same as before.

Your problem is to design an algorithm that goes from the new (left-child, right-cousin) representation of a given forest to the old (left-child, right-sibling) one, i.e., it should blank out the cousin links that aren't siblings. Furthermore, your algorithm should have the following properties:

- A pointer to the leftmost root is given. No other memory is used besides the *left* and *right* arrays and a fixed number of additional pointer registers, independent of the size of the forest.
- The contents of the *left* and *right* arrays and the pointer registers may contain only pointers to the nodes of the forest or the null value  $\Lambda$ . You aren't allowed to "pack in" more information by using signs or tag bits or other such tricks.
- The algorithm should run in linear time, i.e., at worst proportional to the total number of nodes in the forest.

Explain your algorithm to whatever level of detail is necessary to convince the grader that it is correct. If you can't satisfy constraint (c), do the best you can without violating (a) and (b). Partial credit will be given for good ideas that you write down coherently but do not develop completely.

**Problem 2.** (15 points)

Let  $G$  be a finite undirected graph; thus, for every pair of distinct vertices  $u$  and  $v$ , we either have an edge connecting  $u$  to  $v$  or we don't. A simple path of length  $k$  from  $v_0$  to  $v_k$  is a sequence of distinct vertices  $(v_0, v_1, \dots, v_k)$  such that an edge exists between  $v_j$  and  $v_{j+1}$  for  $0 \leq j < k$ .

For each of the following two problems, either prove that the problem is NP-complete, or give an algorithm to solve the problem in polynomial time, where the running time is a polynomial in the number of vertices and edges in  $G$ .

**Problem 2a.** Given a graph  $G$ , an integer  $k$ , and two designated vertices  $x$  and  $y$ , decide if there exists a simple path of length  $\leq k$  from  $x$  to  $y$ .

**Problem 2b.** Given a graph  $G$ , an integer  $k$ , and two designated vertices  $x$  and  $y$ , decide if there exists a simple path of length  $\geq k$  from  $x$  to  $y$ .

*Notes:* If you give an algorithm, an informal but convincing sketch is sufficient. For present purposes, any polynomial-time algorithm is as good as any other; you need not try to find an especially efficient one. In fact, the best algorithm is the one you can *explain* most efficiently, so that you can get on with other parts of this exam!

**Problem 3.** (15 points)

Imagine an array of  $n$  processors initially containing the respective numbers  $(x_1, x_2, \dots, x_n)$ . At each clock pulse, each processor simultaneously replaces its contents by either (a) the sum of the current contents of two processors, or (b) the current contents of some processor. For example, if  $n = 5$ , one of the possible configurations after one step would be

$$(x_2 + x_3, x_5, x_3 + x_5, x_4, 2x_1).$$

(Processor 5 has computed  $x_1 + x_1$ .) After another step, we might have

$$(x_4, 2x_2 + 2x_3, x_2 + 2x_3 + x_5, x_4 + 2x_1, x_2 + x_3);$$

and so on. Note that each processor can, in general, compute different functions at different times.

Find the shortest possible sequence of such steps so that the processors will contain all of the partial sums of the original data, i.e.,

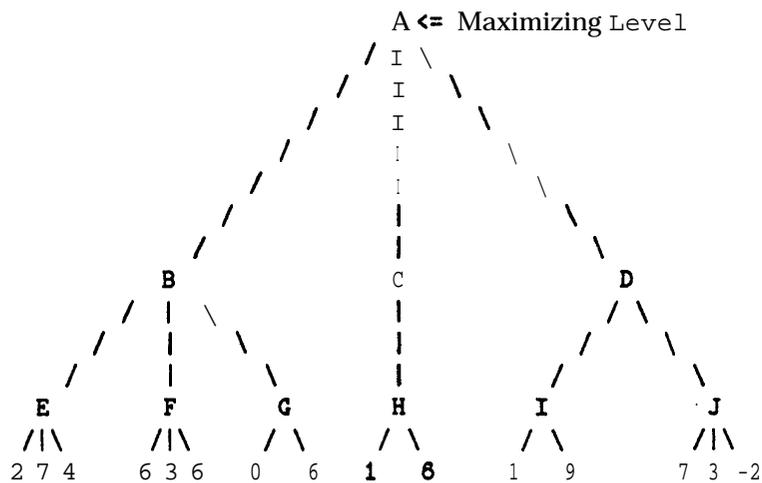
$$(x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots, x_1 + x_2 + x_3 + \dots + x_n)$$

[Hint: There is a two-step solution when  $n = 4$ .]

AI Questiona

1. (12 Points) Game Tree

Consider the following *MiniMax* tree (called an *And-Or* tree in [W&ton]):



- a) [3] What is the solution? That is, which move should be made next (indicate the node using its label); and what is the expected value of that move.
- b) [2] Using the alpha-beta pruning (and standard left-to-right evaluation of nodes), how many of the 15 leaves (bottom nodes) of the tree get evaluated?
- c) [3] Using alpha-beta pruning but *right-to-left* evaluation of nodes, how many of the leaves get evaluated?
- d) [4] What do the answers-to (b) and (c) imply about the worth of spending time choosing the order in which to expand nodes (when using alpha-beta search)? State a heuristic which can guide such guessing; it may either be domain-independent or specific to some task such as Chess .

2. (11 Points) Learning

Consider Winston's ARCH-learning program.

- a) [4] Was its performance (rate of learning) affected much by the order in which examples were presented to it? How, or why not?
- b) [7] Instead of passively waiting for the next example, the program might be modified to present an example to the human and ask him/her about its "archiness". Describe (using the language of MUST-/MAY/MUSTNOT lists) how this might be used effectively.

3. (10 Points) The black board model

- a) [5] The black board model works well when the task exhibits certain characteristics. What are these properties? (Note we're looking for the properties which the task exhibit, not the application domain in which the program is being designed.)
- b) [2] How many dimensions did IcarSayII's black board have; and what were they?
- c) [3] Give an example of a task which would require (well, which would work better with) a different number of dimensions, and with a different set of axes.

4. (15 Points) Representation, I.

Suppose we know that

Whenever a tall man has a wife, that wife likes all of her brothers.

Winter 1982 - Artificial Intelligence

Given this fact, we want to be able to deduce that Martha likes Fred, given that Fred is Martha's brother, and Martha is married to Harry, who is tall.

a) [5] Describe how this would be expressed in a semantic net (or frame) system. [Note: you do not need to write out the actual representation.] Indicate the difficulties involved.

b) [4] Use a frame system which handles defaults to express the fact that swans are white. From this, and the fact that Gertrude is swan, we can deduce that Gertrude is white.

Write the unit that represents Swans (or a typical member/exemplar/... of that set), and the unit for Gertrude. Indicate the GetValue call required to find Gertrude's color. [Note: GetValue(Unit Slot) returns the value of the Slot slot of the Unit unit. This need not be the value "physically" or "primitively" stored there - i.e. GetValue may need to perform some calculations first.]

c) [2] We now add the fact that

George is a black swan.

Write the unit for George, and indicate any changes necessary to the other units.

d) [4] Could this type of exception be handled in ordinary predicate calculus? Describe the difficulties encountered.

5. (12 Points) Representation, II.

The following is a set of first order axioms for the monkey-and-bananas problem. The variable  $a$  is used to represent situations and the constant  $s_0$  is an initial situation.  $result(e, a)$  is the situation that results when event  $e$  occurs in situation  $s$ . The events that concern us are actions of the form  $does(p, a)$  where  $p$  is an actor and  $a$  is an action. The only actor mentioned is *monkey* and the actions he can perform are moving the box and climbing it. The axioms describe the initial situation and give the effects of performing actions. The formula  $at(x, y, s)$  is used to assert that the actor or object  $x$  is at the place  $y$  in the situation  $a$ , and the formula  $in(x, y)$  asserts that  $x$  is permanently (as far as this axiomatization goes) in the place  $y$ .

1.  $in(under-bananas, room)$

2.  $in(corner, room)$

3.  $at(box, corner, s_0)$

3.  $at(monkey, corner, s_0)$

5.  $\forall s \ x \ y \ z. in(x, room) \wedge in(y, room) \wedge at(z, x, s) \wedge at(monkey, x, s)$

$\supset at(z, y, result(does(monkey, move(z, y)), s))$

$\wedge at(monkey, y, result(does(monkey, move(z, y)), s))$

6.  $\forall s \ x. at(box, x, s) \wedge at(monkey, x, s)$

$\supset at(monkey, top(box), result(does(monkey, climb(box)), s))$

7.  $\forall s. at(monkey, top(box), s) \wedge at(box, under-bananas, s)$

$\supset can-get(monkey, bananas, s)$

Questions:

a) [3] Why are these axioms inadequate for showing that the monkey can get the bananas by moving the box under the bananas and climbing the box?

b) [1] What is this difficulty called?

c) [4] Modify one of the axioms in a plausible way so that it can be shown that the monkey can get the bananas.

d) [2] What is unsatisfactory about such modified sets of axioms?

e) [2] Mention what approaches you know to solving the difficulty.

# Hardware Systems

## 1. Logic Design

[20] A mechanical mouse is being designed to follow a track laid out on a rectangular grid. The mouse is equipped with two sensors:

- SA detects the presence of a track to the front.
- SR detects the presence of a track to the right.

Both sensors are active low: a 0 output indicates the presence of a track.

The mouse has a motor with two control leads:

- MA causes the mouse to move ahead 1 grid unit.
- MR causes the mouse to pivot to the right 90 degrees.

Both the motor control lines are active high: a 1 output causes the motor to take the specified action.

After travelling 1 grid unit or making one 90 degree turn, the mouse will pause long enough for the sensors to stabilize, produce a clock pulse, then wait again long enough for any control circuitry outputs to stabilize. At that point, it turns right if MR is high, goes straight if MA is high, or turns 90 degrees to the left if neither is high (MR and MA should not both be high).

You are to design the control unit. It should produce the following mouse behavior: the mouse should turn right at a grid point if it can, turn left if it can't go straight or right, and reverse only if it has no other option. Your control unit may use D flip-flops plus AND, OR and NOT gates.

If you believe that the control unit will require state, please use the following states and state encodings:

State	01	02	Meaning
A	1	1	Move straight ahead 1 grid unit
B	1	0	Pivot (in place) 90 degrees to the right
C	0	1	Pivot (in place) 90 degrees to the left
D	0	0	Pivot (in place) 90 degrees to the left for the second time at an intersection.

You may assume that both the normal and the inverted form of the inputs and flip-flop outputs are available (without going through a NOT gate).

## 2. Memory Mapping

[14] Consider the following byte-addressable computer system:

- Main memory:  $2^{24}$  bytes, composed of  $2^{12}$ -byte **pages**.
- Virtual address space (per process):  $2^{32}$  bytes, composed of  $2^8$ -page **segments**.
- Process id: 4 bits of process id appears with every virtual address (as part of the 32-bit address)

### 2.a. Address Translation

[6] Assume:

- There is a single **fully mapped** page table; i.e., there is an entry in the page table for each **physical** page.
- Another table is used to map the process number and segment number into a page table entry.
- Segments are fully mapped; i.e., either all of the segment is in memory or none of it is.

Show the format of the process-id/segment table and the page table, being exact about the number of entries in the tables, their purpose, and size. Give the exact process for translating a virtual address to a real address.

### 2.b. Caching the Page Table

[8] To save on space, one might cache some portion of the page table. This could be done with an associative memory, but such memories are very expensive. Design an alternative caching scheme that requires only conventional memory and uses substantially less memory than the fully mapped approach. Show the page table and virtual address translation process exactly. Does your scheme perform particularly poorly under any situations? If so, what situations? How can this be improved?

### 3. Evaluation of Instruction Set Designs

[26] Many computer instruction sets include **condition codes** to store the results of conditional expressions. An alternative is to use *conditional branch* instructions. Assume that we have four machines called **I**, **V**, **T**, and **M**. All of them have the following operations:

<b>LOAD</b> <i>addr, r</i>	Load register <i>r</i> with contents of memory location <b>addr</b> .
<b>STORE</b> <i>r, addr</i>	Store contents of register <i>r</i> into memory location <b>addr</b> .
<b>JUMP</b> <i>addr</i>	Set PC to <b>addr</b> .
<b>arit</b> <i>r1, r2, r3</i>	Perform an arithmetic/logical operation ( <b>r3</b> := <i>r1 arith r2</i> ). The arithmetic/logical operations include <b>ADD, SUB, MULT, AND, OR, etc.</b>

These instructions each cost 1 instruction time.

On all machines, **false** is represented by 0, true by any non-zero quantity.

In addition:

- Machine **I** also has the operations:

<b>COMP</b> <i>r1, r2</i>	Compare register <i>r1</i> and <i>r2</i> (a register or 0) and set the condition code.
<b>Bcond</b> <i>addr</i>	Set PC to <b>addr</b> if the condition code satisfies <i>cond</i> . Possible conditions <i>cond</i> are the relational operators <b>EQ, NE, LT, LE, GT, and GE</b> ; i.e., there are instructions <b>BEQ, BGT, etc.</b>

The cost of a **COMP** instruction is 1 instruction time, and a branch instruction (conditional or unconditional) takes 4 instruction times. Arithmetic instructions set the condition code by comparing the result with zero.

- Machine **V** is the same as machine **I**, except that the **LOAD** instruction also sets the condition code, by comparing its result (the value loaded) with 0.
- Machine **T** does not have the **COMP** instruction (nor any condition code) and its conditional branches are of the form:

<b>Bcond</b> <i>r1, r2, addr</i>	Set PC to <b>addr</b> if relation <i>cond</i> holds between <i>r1</i> and <i>r2</i> . <i>cond</i> is a relational operator (as for machine I). <i>r1</i> is a register, and <i>r2</i> is either a register or 0.
----------------------------------	--

This form of conditional branch takes 5 instruction times,

- Machine **M** is the same as machine **T**, except that it has an additional set of instructions:

**Scond** *r1, r2, r3*    Set **r3** := *r1 cond r2. cond* is a relational operator (as for machine **J**).

For example, the instruction **SNE R1, R2, R3** sets **R3** to *true* if the contents of **R1** and **R2** are not equal, *false* if they are equal. This cost of such an instruction is 1 instruction time.

### 3.a. Comparison Tests

[6] We want to compare two quantities **a** and **b**, and jump if the comparison test fails. Assume that the following data holds:

- 10% of the comparisons are between a variable and 0.
- 15% of the comparisons are between an arbitrary expression and 0.
- 75% of the comparisons are between two arbitrary expressions.

Ignoring the cost of **LOADs**, **STOREs**, and *ariths*, find the cost of the compare-and-jump on machines **I**, **V**, and **T**.

### 3.b. Boolean Expressions

[12] Consider the use of boolean expressions in an assignment, as in:

```
var a, b : boolean;
```

```
  a := (i < j) or b ;
```

**Early-out** evaluation of boolean expressions is a technique to reduce the amount of computation needed to evaluate an expression. Using early-out evaluation, the operators **and** and **or** are interpreted as follows:

**a and b**            if **a** then **b** else **false**

**a or b**            if **a** then **true** else **b**

1. [8] Show the minimal code sequence needed to evaluate

```
  a := (i < j) or b
```

(without early-out evaluation) using machines **Y** and **M**.

2. [4] Suppose that early-out evaluation is Allowed (i.e. only the portion of the expression affecting the final result need be evaluated). Show the code sequences for **V** and **M** which minimize the average **execution time**.

### 3.c. Comparison Tests and Boolean Expressions

[8] Suppose the following data is known about boolean expression evaluation where the result is to be stored into a variable.

- 10% of the expressions are a **single** comparison with 0.
- 20% involve a comparison of two variables.
- 70% involve two comparisons and a **boolean operation**.

ignoring the costs of **LOADs**, **STOREs**, and *ariths*, what is the **average** cost of evaluating an expression on **V** and **M** if early-out evaluation is allowed? Assume a comparison results in *true* half of the time.

Problem 1 [25 points]

For calculating the roots of the equation

$$(*) \quad x^3 - 3x^2 - 4x + 1 = 0$$

the following iteration formulas have been proposed

$$A. \quad x_{n+1} = \frac{3x_n^2 + 4x_n - 1}{x_n^2}$$

$$B. \quad x_{n+1} = \frac{x_n^3 - 3x_n^2 + 1}{4}$$

$$C. \quad x_{n+1} = \frac{3x_n^2 - 1}{x_n^2 - 4}$$

(a) [17 points] Which of these formulas, if any, gives a convergent iteration for finding the root of this equation

- (i) which is near **to 4**?
- (ii) which is near to 0?
- (iii) which is near to **-1**?

Give reasons for your answer. If there is more than one answer in any case, which formula would you prefer and why?

(b) [8 points] Use your preferred answer in (a) (ii); starting with  $x_0 = 0$ , to calculate  $x_1, x_2$ . Give bounds for the errors in  $x_2$  and  $x_5$ , but do not calculate  $x_5$ . Find the smallest value of  $n$  for which you can guarantee that  $x_n$  gives a root correct to 5 places of decimals.

Problem 2 [15 points]

The function  $f(x) = \sqrt{x}$  is tabulated at  $x=0, h, 2h, \dots, 100h$  correct to 4 places of decimals, where  $h$  is some small value like 0.01, 0.1 or 1.

(a) [11 points] In what portion of the interval  $[0, 100h]$  is this table well-suited for linear interpolation? (Your answer will depend on  $h$ .) Recall that a table is said to be well-suited for a method of interpolation if the error due to interpolation does not exceed the rounding error of the entries, i.e. one-half unit in the last place.

(b) [4 points] Estimate your answer in (a) for  $h=1, 0.1$  and  $0.01$ .

Problem 3 [20 points]

$A^{(1)}$  is an  $n \times n$  matrix partitioned as follows:

$$A^{(1)} = \left[ \begin{array}{c|c} a_{11} & b^T \\ \hline a & A_{22} \end{array} \right] \begin{array}{l} \left. \vphantom{\begin{array}{c|c} a_{11} & b^T \\ \hline a & A_{22} \end{array}} \right\} 1 \\ \left. \vphantom{\begin{array}{c|c} a_{11} & b^T \\ \hline a & A_{22} \end{array}} \right\} n-1 \end{array} \quad (a_{11} \neq 0).$$

After one step of Gaussian elimination on  $A^{(1)}x = c^{(1)}$  (i.e. eliminating  $x_1$ )  $A^{(1)}$  becomes

$$A^{(2)} = \left[ \begin{array}{c|c} a_{11} & b^T \\ \hline 0 & X \end{array} \right] \begin{array}{l} \left. \vphantom{\begin{array}{c|c} a_{11} & b^T \\ \hline 0 & X \end{array}} \right\} 1 \\ \left. \vphantom{\begin{array}{c|c} a_{11} & b^T \\ \hline 0 & X \end{array}} \right\} n-1 \end{array}.$$

(a) [5 points] Show that

$$X = A_{22} - \frac{ab^T}{a_{11}}.$$

DEFINITION A matrix  $A$  is positive definite if and only if  $A$  is symmetric and  $x^T A x > 0$  for all  $x \neq 0$ .

- (b) [5 points] Show from the definition that if  $A^{(1)}$  is positive definite-then  $a_{ii} > 0$  for all  $i$  and  $b^T = a^T$ . (Yes, this is what is really meant.)'
- (c) [10 points] Prove that if  $A^{(1)}$  is positive definite then  $X$  must also be positive definite. Notice that by continuing this argument it follows that the matrix of order  $n-r$  in the bottom right-hand corner after  $r$  steps of Gaussian elimination is positive definite. Hint: The argument should go as follows: Suppose  $X$  were not positive definite. Then there would exist a vector  $y \neq 0$  of order  $n-1$  such that

$$y^T X y \leq 0.$$

Show that if this is true there is a value of  $\beta$  such that for the vector  $x$  of order  $n$  given by

$$x = \left[ \begin{array}{c} \beta \\ y \end{array} \right] \begin{array}{l} \left. \vphantom{\begin{array}{c} \beta \\ y \end{array}} \right\} 1 \\ \left. \vphantom{\begin{array}{c} \beta \\ y \end{array}} \right\} n-1 \end{array}$$

we have

$$x^T A^{(1)} x = y^T X y \leq 0.$$

Since  $x \neq 0$  this is contrary to the assumption that  $A^{(1)}$  is positive definite.

# Software Systems

## 1. Synchronization and Message-passing

[9] Sketch a simulation of P and Y operations on counting semaphores using message-passing.

## 2. Programming Language Design

[10] Assume we have a programming language whose variables at run time can contain objects of either of the following types:

- character strings of arbitrary length
- integers of arbitrary size

There are no type declarations.

1. [5] What must be done at run time to check variable usage and to efficiently implement the available data types?
2. [5] It will often occur that in a given program certain variables will contain objects of only one of the available types: further, certain variables will often utilize objects of bounded size. Outline how a compiler, at compile time, could test for variables which satisfy one or both of the above restrictions. What run time optimization are possible once this information is available?

## 3. Memory Management

[10] Consider a two-level virtual memory system with  $M$  physical pages and  $N$  virtual pages. This virtual memory system is to use demand paging. We would like to tune the number of pages for each program so that it does not thrash, but also so that it does not require too many pages of marginal utility. Important to the decision of increasing or decreasing the number  $M$  of physical memory pages allocated to a program is the number of page faults that would occur if  $M + 1$  or  $M - 1$  physical pages were allocated. Quickly sketch a method by which a virtual memory system using an LRU page replacement algorithm would gather this additional information on such virtual page faults. (A small amount of additional hardware is not out of the question.)

## 4. Protection

[8] The *access matrix* is a general protection model which describes the access *domains* have to *objects*. A domain may be regarded as a human user, a program, a procedure, or the like. Objects include files and programs. Thus, a domain is itself an object and one domain  $A$  may have particular access rights to another domain  $B$ .

1. [4] Distinguish between capabilities and access control lists as implementations of this model.
2. [4] Consider a system where the subjects for protection purposes are procedures. Describe a method for associating a domain to a procedure. If one procedure wishes to call another procedure, how can the calling procedure pass its domain to the called procedure so that the latter can access those objects?

## 5. Deadlock

[8] Given:

- two processes,  $P$  and  $Q$
- five identical units of a resource  $A$
- two identical units of a resource  $B$

the following sequence of resource requests leads to deadlock:

1.  $P$  requests 3 units of  $A$
2.  $Q$  requests 1 unit of  $B$
3.  $P$  requests 2 units of  $B$
4.  $Q$  requests 4 units of  $A$

Describe briefly two policies an operating system might choose to enforce to avoid deadlock in regard to resource allocation and indicate how each of your policies would modify the sequence of requests that processes  $P$  and  $Q$  make to avoid deadlock.

## 6. Coroutines

1. [3] Give an example where use of coroutines is warranted, and explain why.
2. [3] What data structures in a PASCAL compiler environment will have to be changed to permit coroutines?

## 7. Code Generation

[9] Each of the following is one form of intermediate representation used by compilers. Briefly define each type of representation. Demonstrate how the statement

$$A := 12 * (Ex + Ell / Em)$$

can be represented in each form.

1. syntax tree
2. triples
3. quads (4-tuples)

## Theory of Computation

1. [25 points] Consider the following recursive program to compute the greatest common divisor of two natural numbers (non-negative integers).

$$\begin{aligned} \text{gcd}[m, n] \leftarrow & \text{if } m > n \text{ then } \text{gcd}[n, m] \\ & \text{else if } m = 0 \text{ then } n \\ & \text{else } \text{gcd}[n \bmod m, m] \end{aligned}$$

You may assume that the mod function is a primitive, i.e., it always returns the correct value when its second argument is non-zero.

- (a) [5 points] Prove that  $\text{gcd}[m, n]$  terminates for all natural numbers  $m$  and  $n$ .
- (b) [10 points] Prove that  $\text{gcd}[m, n]$  is a divisor of both  $m$  and  $n$ .
- (c) [10 points] Prove that if a natural number  $d$  divides both  $m$  and  $n$ , then  $d$  divides  $\text{gcd}[m, n]$ .

2. [10 points] Given a predicate of two arguments,  $P(x, y)$ , with the arguments in some domain  $D$ , we call a predicate  $Q(y)$  representable with respect to  $P$  if there is some constant  $a \in D$  such that

$$\forall y. Q(y) \equiv P(a, y).$$

Show that for any predicate  $P$ , there exists a predicate  $Q$  on  $D$  that is not representable with respect to  $P$ .

3. [25 points] Let  $L \subseteq 1(0+1)^*$ . In other words,  $L$  is a set of strings representing some subset of the positive integers in binary notation, without leading zeros. For a finite subset  $A \subseteq L$ , let  $F(A)$  be the string of 0's and 1's such that the  $i$ th symbol in  $F(A)$  is 1 if and only if the binary representation of the integer  $i$  is in  $A$ ; trailing zeros of  $F(A)$  are removed. For example, if

$$A = \{1, 100, 101, 1001\},$$

then

$$F(A) = 100110001.$$

(If  $A$  is the empty set, then  $F(A)$  is the null string.) Now define

$$L' = \{F(A) \mid A \text{ is a finite subset of } L\}.$$

- (a) [3 points] Consider the case where  $L$  is the set of strings representing odd integers. Compute  $L'$ , and show that both  $L$  and  $L'$  are regular.
- (b) [7 points] Find a regular set  $L$  such that  $L'$  is not regular. Justify your answer.
- (c) [5 points] Show that  $L'$  is recursive if and only if  $L$  is recursive.
- (d) [10 points] Show that  $L'$  is recursively enumerable if and only if  $L$  is recursively enumerable.

*Solution to Problem 1.*

The following solution needs only two additional pointer variables (not counting the pointer to the leftmost root, which is assumed given somehow), and it does not change the *left* links. Let  $l_k$  and  $r_k$  be the leftmost and rightmost nodes on level  $k$ , and let  $f_k$  be the parent of  $l_k$ . The idea is to first transform the forest by setting  $right[r_k] = f_k$  (thus removing all of the remaining  $\Lambda$  pointers in the *right* array instead of restoring the clobbered ones!). Then it turns out to be possible to fix everything, starting at the bottom level and working upward.

In the first phase of the algorithm,  $p$  will be the current node of interest on level  $k$ , and  $q = f_k$ . The integer variable  $k$  is included only for purposes of exposition and it can be eliminated. For convenience we may assume that  $left[\Lambda]$  is the leftmost root of the forest; the program can be modified to remove this convention by checking for  $x = \Lambda$  before evaluating  $left[x]$ .

- A1. [Initialize.] Set  $q \leftarrow \Lambda$ ,  $p \leftarrow left[q]$ ,  $k \leftarrow 0$ . If  $p = \Lambda$ , the forest was empty, so the algorithm terminates.
- A2. [Search for  $r_k$ .] (Now  $q = f_k$ , and  $p$  is a node on level  $k$ .) While  $right[p] \neq \Lambda$ , set  $p \leftarrow right[p]$  and repeat this step.
- A3. [Adjust the structure.] (At this point  $p = r_k$ ,  $q = f_k$ .) Set  $right[p] \leftarrow q$ , then set  $p \leftarrow left[q]$ . (Now  $p = l_k$ .)
- A4. [Search for  $f_{k+1}$ .] While  $left[p] = \Lambda$  and  $right[p] \neq q$ , set  $p \leftarrow right[p]$  and repeat this step.
- A5. [Bottom level?] If  $left[p] \neq \Lambda$  (this implies that  $p = f_{k+1}$  and  $left[p] = l_{k+1}$ ), set  $q \leftarrow p$ ,  $p \leftarrow left[q]$ ,  $k \leftarrow k + 1$ , and return to A2. Otherwise terminate phase one. (Level  $k$  is the bottom level, and we have  $q = f_k$ ,  $p = r_k$ .) ■

In the second phase we make  $q$  run through level  $k - 1$  as  $p$  runs through level  $k$ .

- B1. [Prepare to fix level  $k$ .] (At this point  $q = f_k$ ,  $p = r_k$ , and all levels  $> k$  are in the desired final state.) If  $q = \Lambda$ , we are done ( $k = 0$ ). Otherwise set  $right[p] \leftarrow \Lambda$ , then set  $p \leftarrow left[q]$ .
- B2. [Move  $q$  right.] (At this point  $p = left[q]$ .) Set  $q \leftarrow right[q]$  one or more times until  $left[q] \neq \Lambda$ . (In particular this will occur when  $q = right[r_{k-1}] = f_{k-1}$ , so the loop terminates.)
- B3. [Move  $p$  right.] (At this point either  $q = f_{k-1}$  and level  $k$  has been completely fixed up, or  $left[q]$  lies to the right of  $p$  on level  $k$ .) If  $right[p] = \Lambda$ , go to B4. Otherwise if  $right[p] = left[q]$ , set  $right[p] \leftarrow \Lambda$  and  $p \leftarrow left[q]$  and go back to step B2. Otherwise set  $p \leftarrow right[p]$  and repeat this step.
- B4. [Move up.] (Level  $k$  is done and  $q = f_{k-1}$ .) Set  $p \leftarrow left[q]$  (i.e.,  $l_{k-1}$ ), and then while  $right[p] \neq q$  set  $p \leftarrow right[p]$ . (Now  $p = r_{k-1}$ .) Decrease  $k$  by 1 and return to B1. ■

The algorithm runs in linear time, since each node is visited at most five times.

*Solution to Problem 2.*

Problem 2a can be solved in polynomial time. For example, consider the set  $S_d = \{z \mid \text{there exists a simple path of length } \leq d \text{ from } x \text{ to } z\}$ . We have  $S_0 = \{x\}$ , and  $S_{d+1} = S_d \cup \{z \mid \text{there is an edge from } z \text{ to an element of } S_d\}$ . A brute-force method will obviously compute  $S_{d+1}$  from  $S_d$  in time  $O(m)$ , where  $m$  is the number of edges, so we can compute  $S_k$  in  $O(km)$  steps. Note that we can assume that  $k$  is less than the number of vertices, since a simple path of length  $k$  involves  $k + 1$  different vertices.  $\square$

Problem 2b is NP-complete. It is clearly in NP, since we can find a simple path of length  $\geq k$  by making at most  $n - 1$  guesses, where  $n$  is the number of vertices. Conversely, we can reduce the NP-complete Hamiltonian cycle problem to it as follows: Given a graph  $H$  for which we want to determine the existence of a Hamiltonian cycle, let  $x$  be a vertex of  $H$ . Construct a new graph  $G$  consisting of  $H$  plus a new vertex  $y$ ; there is an edge between  $v$  and  $y$  in  $G$  if and only if there is an edge between  $v$  and  $x$  in  $H$ , for all vertices  $v$  of  $H$ . It is obvious that  $H$  has a Hamiltonian cycle if and only if there is a simple path of length  $\geq n$  from  $x$  to  $y$  in  $G$ , where  $n$  is the number of vertices of  $H$ .

*Solution to Problem 3.*

If we can compute  $(x_1, x_1 + x_2, \dots, x_1 + \dots + x_n)$  with  $n$  processors in  $t$  steps, we can use the same pattern to compute

$$(x_1, x_1 + x_2, \dots, x_1 + \dots + x_n, x_{n+1}, x_{n+1} + x_{n+2}, \dots, x_{n+1} + \dots + x_{2n})$$

with  $2n$  processors in  $t$  steps, so we can compute

$$x_1, x_1 + x_2, \dots, x_1 + \dots + x_n, x_1 + \dots + x_n + x_{n+1}, \dots, x_1 + \dots + x_n + x_{n+1} + \dots + x_{2n} \quad \bullet \quad (+2n)$$

with  $2n$  processors in  $t + 1$  steps. Thus we can solve the problem for  $2^t$  processors in  $t$  steps.

In this method no processor ever looks at any processor to its right, so the same construction works for any  $n \leq 2^t$  if we ignore all but the leftmost  $n$  processors.

We have constructed a  $t$ -step method if  $2^{t-1} < n \leq 2^t$ . This method is as short as possible, since no  $(t - 1)$ -step method can form a sum of more than  $2^{t-1}$  terms; the number of terms at most doubles at each step.

The solution just given is an example of the divide-and-conquer methodology. There is also another solution that has "bounded fanout": We can put the sum  $\sum_{k-2^t < j \leq k} x_j$  into processor  $k$  after  $t$  steps, where  $x_j$  is zero for  $j \leq 0$ .

Winter 1982 - Artificial Intelligence (Solutions)  
Solutions to the AI Questions

1. (12 Points) Game Tree

- a) [3] Of the three possible choices, the solution is *D*. Its value is 7.
- b) [2] All 15.
- c) [3] Only 8. (All three nodes under the *J* node, (with values 7, 3 and -2); the 9 node under *I*, the 6 and 1 under *II*, and the 6 and 0 under *G*.)
- d) [4] Yes, it's worth it. In the case of a tree with branching factor *b* and depth *d*, the savings can be as much as  $b^d - b^{(d/2)}$ . Here are two general ordering heuristics:

*Expand the tree to depth  $d - 1$ , and order the paths with those estimates.*

*If you begin to expand a subtree isomorphic to one you've expanded before, then you can guess at the value of it and decide where to put it in the ordering.*

A more specialized, domain-dependent heuristic is:

*Order the nodes so that if a mate is there you'll find it;  
Once you find a forced mate, you can stop looking.*

2. (13 Points) Learning

- a) [4] Yes. The program finds some random difference to update its model each time. If it is presented with "near misses" each time, it is bound to find the one, small, meaningful difference. But if the order of presentation were changed, one might show one of these "near-miss" cases too soon, and it would have MANY differences from the model, only one or two of which would be real. The chances of a valid learning event from such an example is therefore small.
- b) [7] Suppose the teacher types in a scene which the program says is an Arch, but the teacher tells it it's wrong. The program has many ways it can adjust its model so that this scene would not be considered to be an example of the Arch concept. One thing the program can do is to find those entries on its MAY list which are not satisfied in the scene, and then randomly pick one to transfer to its MUST list. Alternatively, the program can find all the relations in the scene which are not on its MUST or MAY list, and randomly picks one and adds it to its MUSTNOT list. The "near-miss" idea is just the careful presentation of examples so that the total number of alternatives is very low. But another approach would be for the program to immediately record all its options, and for each one make up a scene that was just like its arch model, except for that feature. The program then asks the teacher to look at each one in turn, and say whether or not it's an arch. Each one of these machine-generated scenes is guaranteed to be either an arch or a near-miss (precisely one feature different from its arch model).

3. (10 Points) The black board model

- a) [5] The black board model is most appropriate for tasks which can be decomposed into several almost independent subparts. It then permits a host of "sub-experts" to each work on a different part of the overall problem. (In HearSayII's case, the experts are called Knowledge Sources.) Each expert is only responsible for addressing its particular problem. It must therefore know what sort of information it needs, and where this on the black board, but it need not know how that information was derived.

All expert-to-expert communication is via this indirect, and therefore, rather expensive, black board. For this reason it is important that each experts be able to do most of its work independently.

- b) [2] HearSayII used axes: abstraction and time.

(Almost everyone answered a different question, giving the "levels" of its black board - which each corresponded to a particular Knowledge Source. These may be viewed as the points along the abstraction dimension.)

Winter 1982 - Artificial Intelligence (Solutions)

- c) [3] There are many possibilities, including
  - Vision • using x/y coordinates and abstraction
  - Planning • using abstraction, time, which actor, (and maybe meta-level)
  - Diagnosis • using location of problem/site of infection, time, suspected type of disease.

4. (15 Points) Representation, I.

a) [5] This would be VERY difficult to represent using a frame system. The various niceties of a frame system - to house facts about, for example, a typical man - are of no use in this convoluted case.

The most appropriate solution would be, essentially, to "simulate" the messy predicate calculus statement shown above, encoding each of the variables as a unit. Note we would need to show things like their skolem-dependencies, which come for nothing by the order in the PC statement. Another unit would be required to handle the various connectives - here the AND and IMPLICATION connectives.

The interpreter (or inferencing engine) associated with this frame system would need to "know" how to deal with such variables, etc, to perform the inferences needed to realize that Martha likes Fred.

b) [4]

TypicalSwan

TypicalExampleOf: Swan  
 Color: White  
 Aspect: by default  
 Description: This unit stores facts which pertain to swans, in general.

Gertrude

IsA: Swan  
 Description: This unit represents the facts about the swan, Gertrude.

Now (Get Value 'Gertrude 'Color) will return White.

c) [2]

George

IsA: Swan  
 Color: Black  
 Description: This unit represents the black swan, George.

The value of (Get Value 'George 'Color) is Black.

Given the system described in 4b), no changes need be made to the other units - in particular, TypicalSwan:Color is unaffected by George's arrival.

If we had claimed that Swans were White by definition, (ie if TypicalSwan's Color slot was White, with the aspect "by definition"), this aspect would have to be changed to "by default". Also, had the TypicalSwan unit above included a range of values for the color of swans, which did not include Black, that range would have to be augmented.

d) [4] This would be very difficult to handle in (vanilla) predicate calculus. Had we stated that

$$\forall x.(Swan\ x) \supset ((Color\ x) = White),$$

finding (SwanGeorge) and (ColorGeorgeBlack) [together with the "obvious" fact that (NOT(EQUAL White Black))] would have resulted in a contradiction (which, in turn, would have rendered this knowledge base inconsistent and theoretically useless). As we wanted to infer from Gertrude's swan-ness that she was white, simply enumerating the known swans, and giving each of them the color White is insufficient.

There has been much recent work on these issues, from both logicians and AI researchers. Special "modal" operators have been used to handle cases where the "unprovability" of some fact can be used when deriving some other proposition. Here, we could write

$$\forall x.(Swan\ x) \ \& \ \sim \Box(\exists y. Color\ x\ y) \supset (Color\ x\ White),$$

where  $\Box\phi$  means  $\phi$  was not derivable.

Many researchers are currently investigating such "default reasoning" schemes, (a branch of non-monotonic logic,) trying to formalize the process.

5. (12 Points) Representation II

- a) [3] It does not follow that when the monkey climbs the box, the box will still be under the bananas.
- b) [1] This is the "frame problem".
- c) [4] Axiom 6 should be replaced by

$$\begin{aligned} 6' . \forall s\ x. at(box, x, s) \wedge at(monkey, x, s) \\ \supset at(monkey, top(box), result(does(monkey, climb(box)), 3)) \\ \wedge at(box, x, result(does(monkey, climb(box)), s)) \end{aligned}$$

- d) [2] This approach requires that the axiom giving the effect of an event describe all the predicates that are unmodified by the event.
- e) [2] STRIPS, frames: microplanner, and non-monotonic reasoning for example.

# Hardware Systems

## 1. Logic Design

First derive the state transitions:

State Transition Table

SR	SA	I'	A	B	C
0	0		B	A	
0	1		<b>B</b>		
<b>1</b>	1		C	-	C
1	0		A		A

Note: If a mouse has just made a right-hand turn, it must be that there is now a path in front of it (formerly to the right) and a path to the right (which it traversed to get to the intersection). Hence, from state **B**, there are 3 don't care states. If a mouse has just made a left-hand turn, it cannot see a path to the left (this would have been straight ahead before the turn). So, from state **C**, there are 2 don't care states,

State Transition Table with States Expanded to Flip-Flop Contents:

SR	SA	D1	D2	D1'	D2'
0	0	0	0	-	-
0	0	0	1	-	-
0	0	1	1	1	0
0	0	1	0	1	1
0	1	1	0	-	-
0	1	1	1	1	0
0	1	0	1	-	-
0	1	0	0	-	-
1	1	0	0	-	-
1	1	0	1	0	1
1	1	1	1	0	1
1	1	1	0	-	-
1	0	1	0	-	-
1	0	1	1	1	1
1	0	0	1	1	1
1	0	0	0	-	-

Now, since the state is being stored in D flip-flops, the input needed to get a flip-flop to hold a particular value is the same as that value. So, minimum sum-of-products expressions for the new states can be derived from the above table using Karnaugh maps:

D1':

		SR			
		X	X	X	X
		X	X	0	1
D1		1	1	0	1
		1	X	X	X
		SA			

$$D1' = \sim SA + \sim SR$$

D2':

		SR			
		X	X	X	X
		X	X	1	1
D1		0	0	1	1
		1	X	X	X
		SA			

$$D2' = \sim D2 + SR$$

Motor control leads: Since the mouse waits at an intersection long enough for outputs derived from the new state to stabilize, the simplest method of getting the outputs is simply to take them from the flip-flop outputs:

$$\begin{aligned} MR &= D1 \cdot \sim D2 && \text{(ie, mouse in state 8)} \\ MA &= D1 \cdot D2 && \text{(ie, mouse in state A)} \end{aligned}$$

This method also ensures that the control leads will not fluctuate during a turn (the inputs from the sensors probably will).

Other solutions: A couple of other possible solutions can be obtained by using state D (on  $SR = SA = 1$ , go from state C to state D; always go from D to A) or by defining a new pair of states:

- R: just made/are making a right hand turn
- 0: any other action

From 0 go to R if  $SR = 0$ . From R always go back to 0. In 0, go forward if  $SA = 0$ , otherwise turn left

## 2. Memory Mapping

### 2.a. Address Translation

Given a 32-bit virtual address, with 4 bits for process id, S bits for page number, and 12 bits for byte offset, this leaves  $S$  bits for segment number; i.e. a process may have up to 256 segments. The resulting virtual address looks like (not to scale):



Winter 1982 - Hardware Systems (Solutions)

LOADS,	STOREs,	ariths for temporaries	cost
arith	R1, R2, R3	(final result)	0
Bcond	addr		4

- Comparison between two arbitrary expressions:

LOADS,	STOREs,	ariths for temporaries	cost
COMP	R1, R2		1
BNE	FAIL'		4

Final cost:

$$\text{Cost(I)} = .1*5 + .15*4 + .75*5 = 4.85$$

Similar analysis yields:

$$\text{Cost(V)} = .1*4 + .15*4 + .75*5 = 4.75$$

$$\text{Cost(T)} = .1*5 + .15*5 + .75*5 = 5.00$$

### 3.b. Boolean Expressions

#### 1. Machine V:

	LOAD	TRUE, R0			
	LOAD	i, R1			
	LCAD	j, R2			
	COMP	R1, R2			
	BLT	CONT			
	LOAD	FALSE, R0			
CONT	LOAD	b, R1			
	OR	R0, R1, R0	(or	BNE	DONE
				LOAD	FALSE, R0)
DONE	STORE	R0, a			

#### Machine M:

	LOAD	i, R1
	LOAD	j, R2
	SLT	R1, R2, R3
	LOAD	b, R4
	OR	R3, R4, R4
	STORE	R4, a

#### 2. Machine V:

	LOAD	TRUE, R0
	LOAD	i, R1
	LOAD	j, R2
	COMP	R1, R2
	BLT	DONE
	LOAD	b, R0
DONE	STORE	R0, a

Note that the code is identical except for the BLT branch; that branch reduces execution time!

Machine M: If we used early-out, we'd probably generate code like:

Winter 1982 - Hardware systems (Solutions)

```

LOAD TRUE, R0
LOAD i, R1
LOAD j, R2
BLT R1, R2, DONE
LOAD b, R0
DONE STORE R0, a
    
```

This requires 10 instruction times, whereas full evaluation requires only 6! A *Bcond* will always require more time than an *Scond*, to achieve the same effect, so stick with full evaluation!

3.c. Comparison Tests and Boolean Expressions

Machine *V*: We can ignore the store into the variable:

- Single comparison with 0:

		Cost	% of time evaluated
LOAD	var, R1	0	100%
Bcond	store 0	4	100%

weighted cost = 4 \* .1 = .4

- Comparison of two variables:

		cost	% of time evaluated
LOAD	a, R1	0	
LOAD	b, R2	0	
COMP	R1, R2	1	100%
Bcond	store 0	4	100%

weighted cost = 5 \* .2 = 1.0

- Two comparisons and a boolean: The OR and AND cases are identical except for the sign of the branch and the values stored into the variable.

		cost	% of time evaluated
LOAD	a, R1	0	
LOAD	b, R1	0	
COMP	R1, R2	1	100%
Bcond	store 1	4	100%
LOAD	c, R1	0	
LOAD	d, R2	0	
COMP	R1, R2	1	50%
Bcond	store 0	4	50%

weighted cost = .7 \* (5 + .5 \* 5) = 5.25

The total cost for machine *V* is 6.65.

Machine *M*: We need only count the *Sconds* since full evaluation is always faster.

- For compare with 0: a single *Scond* is needed. Cost = .1 \* 1 = .1.
- For comparing two variables, a single *Scond* is needed. Cost .2 \* 1 = .2.
- For comparing two expressions and combining the result, two *Sconds* are needed. Cost = .7 \* 2 = 1.4.

Total cost on machine *M* is 1.7.

Winter 1982

Numerical Analysis - Solutions

Problem 1

Basic tools are Theorem 2.3 and 2.4 of Atkinson, An Introduction to Numerical Analysis.

The general iteration formula is

$$x_{n+1} = g(x_n).$$

It suffices to examine the behavior of  $g'(x)$  in the neighborhood of a root  $\alpha$  of  $x = g(x)$ .

(a) A.  $g_1(x) = 3 + \frac{4}{x} - \frac{1}{x^2}$

$$g_1'(x) = -\frac{4}{x^2} + \frac{2}{x^3} = \frac{2-4x}{x^3}$$

(i) neighborhood of 4:  $g_1'(4) = \frac{-14}{64}$ ,  $|g_1'(x)| < \frac{1}{4}$   $\therefore$  convergent

(ii) neighborhood of 0:  $g_1'(x) \rightarrow \infty$ . divergent

(iii) neighborhood of -1:  $g_1'(-1) = -6$ . divergent.

3.  $g_2(x) = \frac{x^3 - 3x^2 + 1}{4}$

$$g_2'(x) = \frac{3x^2 - 6x}{4}$$

(i) neighborhood of 4:  $g_2'(4) = 6$ . divergent

(ii) neighborhood of 0:  $g_2'(0) = 0$ . convergent

(iii) neighborhood of -1:  $g_2'(-1) = \frac{9}{4}$ . divergent

c.  $g_3(x) = \frac{3x^2 - 1}{x^2 - 4}$

$$g_3'(x) = \frac{-22x}{(x^2 - 4)^2}$$

Winter 1982

Numerical Analysis -Solutions

Problem 1 (cont'd)

(i) neighborhood of 4:  $g'_3(4) = \frac{-88}{144}$  ,  $|g'_3(x)| < \frac{3}{4}$  . convergent

(ii) neighborhood of 0:  $g_3(0) = 0$  . convergent

(iii) neighborhood of -1:  $g'_3(-1) = \frac{22}{9}$  . divergent

For (i) I prefer A. as it has smaller  $|g'(x)|$

For (ii) I prefer B. as it is slightly easier to compute.

b) Use B: 
$$x_{n+1} = \frac{x_n^3 - 3x_n^2 + 1}{4}$$

$$x_0 = 0, x_1 = \frac{1}{4}, x_2 = \frac{53}{256} < \frac{1}{4}$$

Now in  $[0, \frac{1}{4}]$  ,  $g_3(x)$  decreases from 0 to  $-\frac{21}{64}$  and so for  $x \in [0, \frac{1}{4}]$  ,

$$g(x) \in [0, \frac{1}{4}]$$

Moreover in  $[0, \frac{1}{4}]$  ,  $|g'_3(x)| < \frac{1}{3}$  . Hence take  $\lambda = \frac{1}{3}$  .

Then 
$$|\alpha - x_n| \leq \lambda^n |\alpha - x_0|$$

Since  $\alpha$  in  $[0, \frac{1}{4}]$  ,  $|\alpha - x_0| < \frac{1}{4}$

$$\therefore |\alpha - x_n| \leq \frac{1}{4} \cdot 3^{-n}$$

$$\therefore |\alpha - x_2| \leq \frac{1}{4} 3^{-2} = \frac{1}{36} \quad , \quad |\alpha - x_5| \leq \frac{1}{4} 3^{-5} = \frac{1}{972}$$

$x_n$  correct to 5 places of decimals if  $\frac{1}{4} 3^{-n} \leq \frac{1}{2} 10^{-5}$  or  $3^n > \frac{2}{4} 10^5 = 50000$

i.e. if  $n \geq 10$ .

Winter 1982

Numerical Analysis - Solutions

Problem 2

The formula for linear interpolation between  $x_n$  and  $x_{n+1}$  may be written in the form

$$p(x) = f(x_n) + (x-x_n) \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n}$$

The error is given by

$$E = \frac{f''(\xi)}{2} (x-x_n)(x-x_{n+1})$$

If  $|f''(x)| \leq M$  in  $(x_n, x_{n+1})$  then

$$|E| \leq \frac{M}{2} \max |(x-x_n)(x-x_{n+1})| = \frac{M}{2} \frac{h^2}{4} = \frac{Mh^2}{8} \quad \text{where } h = x_{n+1} - x_n$$

$$\text{since max occurs at } x = \frac{x_n + x_{n+1}}{2}$$

Now for  $f(x) = \sqrt{x}$

$$f'(x) = \frac{1}{2} x^{-1/2}$$
$$f''(x) = -\frac{1}{4} x^{-3/2}$$

Let  $x_n = nh$ ; then in  $(x_n, x_{n+1})$  we have  $|f''(x)| \leq \frac{1}{4(nh)^{3/2}}$

$$\therefore |E| \leq \frac{1}{4(nh)^{3/2}} \cdot \frac{1}{8} h^2 = \frac{h^{1/2}}{32n^{3/2}} \quad \text{in } (nh, (n+1)h)$$

If the table is well-suited for linear interpolation then we must have

$$|E| \leq \frac{1}{2} 10^{-4}$$

i.e.  $\frac{h^{1/2}}{32n^{3/2}} < \frac{1}{2} 10^{-4} \quad \text{or} \quad n^{3/2} > \frac{10^4 h^{1/2}}{16} = 625h^{1/2}$

$$\text{or } \underline{n > (625)^{2/3} h^{1/3} = n_0}$$

$$\therefore \text{OK in } \underline{[(n_0)+1]h, 100h}$$

Numerical Analysis - Solutions

Problem 2 (cont'd)

(b)  $\underline{h=1}$   $n > (625)^{2/3} \approx 73.1$

$\therefore$  OK in [74, 100]

$h=0.1$   $n > (625)^{2/3} (.1)^{1/3} = 625^{2/3} \left(\frac{100}{1000}\right)^{1/3} \approx \frac{73.1 \times 4.64}{10}$   
 $\approx 33.9$

$\therefore$  OK in [3.4, 10]

$\underline{h=0.01}$   $n > (625)^{2/3} (.01)^{1/3} = 625^{2/3} \left(\frac{10}{1000}\right)^{1/3} = \frac{73.1 \times 2.15}{10} = 15.7$

OK in [.16, 1.00]

Problem 3

(a) Let

$$a = \begin{bmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{ni} \end{bmatrix}$$

$$b^T = [a_{12}, a_{13}, \dots, a_{1n}]$$

$$X = (x_{ij})$$

$$i, j = 2, \dots, n$$

Then using the usual Gaussian elimination scheme, we have after one step.

$$x_{ij} = a_{ij} - \frac{a_{i1} a_{1j}}{a_{11}}, \quad i, j = 2, \dots, n$$

Thus 
$$X = A_{22} - \frac{ab^T}{a_{11}}$$

(b) Suppose  $A^{(1)}$  is positive definite and suppose  $a_{ii} < 0$  for some;

Choose  $x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$   $--i^{\text{th}}$  component

Then  $x^T A^{(1)} x = a_{ii} < 0$ . Contradiction. Hence we must have  $a_{ii} > 0$

for all  $i$ .

$b^T = a^T$  is obvious from symmetry of  $A^{(1)}$

Winter 1982

Numerical Analysis - Solutions

Problem 3 (cont'd)

(c) We are assuming that there is a  $y \neq 0$  such that  $y^T X y \leq 0$ .

We have then

$$\begin{aligned} 0 \geq y^T X y &= y^T (A_{22} - \frac{aa^T}{a_{11}}) y \\ &= y^T A_{22} y - \frac{(y^T a)(a^T y)}{a_{11}} \\ &= y^T A_{22} y - \frac{(a^T y)^2}{a_{11}} \end{aligned} \quad (1)$$

This being so, we show that we can construct an  $x \neq 0$  of the form described such that  $x^T A(1)x \leq 0$ , thereby contradicting the hypothesis.

$$\begin{aligned} x^T A(1)x &= \begin{bmatrix} \beta & | & y^T \end{bmatrix} \begin{bmatrix} a_{11} & | & a^T \\ \hline a & & A_{22} \end{bmatrix} \begin{bmatrix} \beta \\ y \end{bmatrix} \\ &= \begin{bmatrix} \beta & | & y^T \end{bmatrix} \begin{bmatrix} \beta a_{11} + a^T y \\ \beta a + A_{22} y \end{bmatrix} \\ &= \beta^2 a_{11} + \beta a^T y + \beta y^T a + y^T A_{22} y \\ &= \beta^2 a_{11} + 2\beta(a^T y) + y^T A_{22} y. \end{aligned} \quad (2)$$

Choose  $\beta = -(a^T y)/a_{11}$ . Then from (2)

$$\begin{aligned} x^T A(1)x &= \frac{(a^T y)^2}{a_{11}} - \frac{2(a^T y)^2}{a_{11}} + y^T A_{22} y \\ &= -\frac{(a^T y)^2}{a_{11}} + y^T A_{22} y \\ &\leq 0 \text{ from (1)}. \end{aligned}$$

(c) Alternative solution in matrix terms is as follows. In the positive definite case we have

$$A^{(1)} = \left[ \begin{array}{c|c} a_{11} & a^T \\ \hline a & A_{22} \end{array} \right]$$

Let  $M = \left[ \begin{array}{c|c} 1 & 0 \\ \hline -\frac{a}{a_{11}} & I \end{array} \right] \begin{matrix} 1 \\ n-1 \end{matrix}$

$$MA^{(1)} = \left[ \begin{array}{c|c} a_{11} & a^T \\ \hline 0 & A_{22} - \frac{aa^T}{a_{11}} \end{array} \right] \begin{matrix} 1 \\ n-1 \end{matrix} = \left[ \begin{array}{c|c} a_{11} & a^T \\ \hline 0 & X \end{array} \right]$$

$$MA^{(1)}M^T = \left[ \begin{array}{c|c} a_{11} & 0 \\ \hline 0 & X \end{array} \right] \quad (MA^{(1)}M^T \text{ is obviously symmetric}) \quad (1)$$

$A^{(1)}$  positive definite  $\Leftrightarrow MA^{(1)}M^T$  positive definite (from definition of positive definiteness and non-singularity of  $M$ ).

If  $A^{(1)}$  is not positive definite, there exists  $y \neq 0$  such that  $y^T X y = 0$ .

i.e.  $\begin{bmatrix} 0 & y^T \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & X \end{bmatrix} \begin{bmatrix} 0 \\ y \end{bmatrix} \leq 0$  from (1)

i.e.  $\begin{bmatrix} 0 & y^T \end{bmatrix} M[A^{(1)}]M^T \begin{bmatrix} 0 \\ y \end{bmatrix} \leq 0$

i.e.  $z^T A^{(1)} z \leq 0$  (2)

where  $z = M^T \begin{bmatrix} 0 \\ y \end{bmatrix} = \begin{bmatrix} (-a^T y)/a_{11} \\ y \end{bmatrix} \neq 0$ .

(2) Not possible because  $A^{(1)}$  is positive definite.

# Software Systems

## 1. Synchronization and Message-passing

The semaphore(s) is (are) managed by a server process which accepts three types of message:

*CreateSemaphore* Create/initialize a semaphore variable, initialize it to the value based with the request, and return a *handle* or id for that semaphore.

*P* *P* on the indicated semaphore.

*V* *V* on the indicated semaphore.

When a set of clients wishes to instantiate a process for synchronization, one of them must send a *CreateSemaphore* message to the server and make the resulting id available to the others. Then, to synchronize, the cooperating process simply send *P* and *V* messages to the server, containing the semaphore id, and wait for a response. If a *P* causes the client to block, it is placed on a queue internal to the semaphore server until such time as another client's *V* wakes it up.

## 2. Programming Language Design

1. Each variable contains a pointer to a record of unknown size. To check variable usage, each record must have a tag saying whether it is an integer or a string. Also need length information. Should allocate values (and return them) to a heap. Probably should have "records" of only certain fixed sizes (Fibonacci or binary) and allocate the smallest size record that is large enough for the value.
2. By global data flow analysis in some cases one can determine that a variable is used only as an integer or as a string. By adding range analysis in some cases, one can bound the size of values. If we know a variable is always a specific type, the compiler doesn't need to check its type before each operation. We may also be able to eliminate the tag field describing the type. If a variable contains objects of bounded size, we can do a single allocation of space for it (the largest ever used). May be able to avoid the indirection.

## 3. Memory Management

A virtual memory system with  $M$  real pages using LRU can keep track of the number of page faults for  $M+1$  real pages by simply noticing on every page fault if the page being demanded is the same one that was most recently victimized. This requires no extra hardware, and only a negligible amount of overhead on a page fault.

To keep track of virtual page faults of memory size of  $M-1$ , we simply lie to the system and tell it that there are only  $M-1$  pages in real memory. Whenever we get a page fault, we see if the page demanded is our extra page. If it is, we can "page it in" without any I/O; if it isn't, we would have required an I/O transfer anyway. The important thing to notice is that our white lie will cause only a negligible increase in running time because by the LRU assumption, the extra page will be accessed only infrequently.

## 4. Protection

1. The difference lies in when the access rights are checked and where the access control information is stored. With access lists, the rights of an individual to manipulate an object are checked each time he attempts to do so. The access list storage is associated with the object. With capabilities, the mere fact

## Winter 1982 - Software Systems (Solutions)

that someone has a capability grants certain access rights to it. Operations on capabilities are restricted to ensure their continued validity. Storage of capabilities is associated with the user.

2. An executing domain could contain a capability for the domain object itself which could be passed as a parameter allowing objects in the caller's domain to be accessible through a path subject to rights along that path.

### 5. Deadlock

Many possible policies exist. A few of the more likely answers are listed below.

1. Lock All

Require only one process to execute on the machine at a time. Run it to completion. Under this policy, P runs first, using whatever resources it wants. Then Q runs after P is done.

2. Preclaim

Require processes to claim all the resources that they might ever want to use before they start to use any. So, P would claim 3 units of A and 2 of B immediately. Q would be forced to wait until P is done to be granted its claim of 4 and 1.

3. Preschedule commitments

Require processes to declare the maximum amount of all the resources they might want to use. Request resources (up to the max declared) at will. Use a method such as the Banker's algorithm (Brinch Hansen) to decide whether or not a process must be blocked for a time before each request can be granted. Here, Q's first request cannot be granted immediately since P might need the unit of B to complete.

4. Ordering

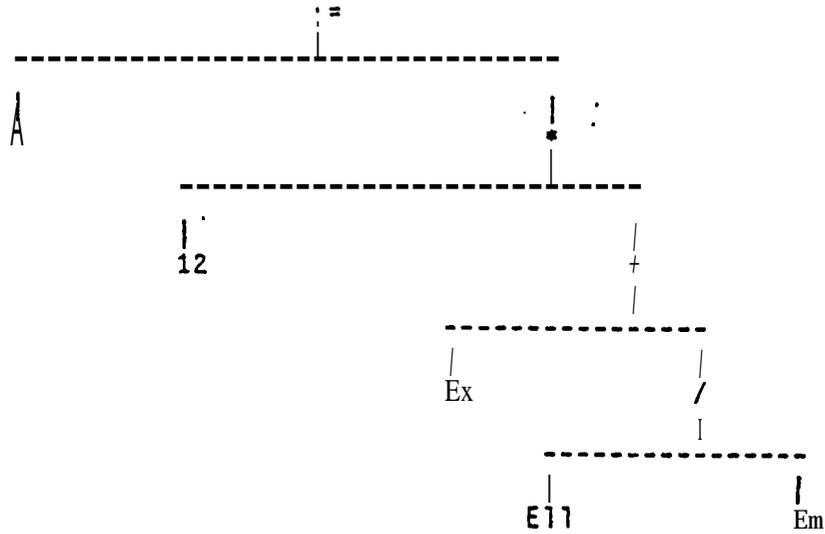
Assign each distinct resource type a priority. Allow only requests for resources of higher priority than the highest priority held. Q must request its 4 units of A before it asks for 1 of B. Hence, it will block until P is done.

### 6. Coroutines

1. Processing of a FORMAT list in FORTRAN or PL/1. The list elements are obtained by one coroutine, and the conversion specification elements by another. The match of list and conversions is only by count, i.e., the number of mutual invocations. The format specifications can include constant data elements to be output. The list elements may require arbitrary computations. The arbitrariness of these elements makes passing of the elements from one routine to the other by parameter-passing mechanisms well nigh impossible.
2. Multiple stacks have to be maintained, one for each coroutine. The heap storage has to be protected if heap deletion is performed by unstacking.

### 7. Code Generation

1. A tree whose interior nodes represent operations and leaves represent objects/constants.



2. A list of the operations, each one being specified as an operation and up to two operands. The operands may be objects (variables or constants) or references to the result of other operations.

	operation	arg 1	arg 2
1.	DIV	El 7	Em
2.	PLUS	Ex	(1)
3.	TIMES	12	(2)
4.	ASSIGN	A	(3)

3. A list of the operations, each being specified as an operation, up to two operands, and a result. No references to other quads are permitted as operands or results.

Op	Arg 1	Arg 2	Result
DIV	E11	Em	T1
PLUS	Ex	T1	T2
TIMES	12	T2	A

Theory of Computation Solutions

1. (a) Whenever  $gcd[m, n]$  calls  $gcd$  recursively, the first argument is strictly less than  $m$ . (The two cases are  $gcd[n, m]$  when  $n < m$ , or  $gcd[n \bmod m, m]$ , and  $n \bmod m$  is always less than  $m$ .) Therefore, the case  $m = 0$  must eventually be reached, and the recursive calls will end.
- (b) We will show by induction that

$$\Phi(m) \equiv \forall n. (gcd[m, n] \mid m) \wedge (gcd[m, n] \mid n)$$

holds for all natural numbers  $m$ . (The symbol " $\mid$ " indicates divisibility.) The base case is  $m = 0$ ; here we have  $gcd[m, n] = n$ , and  $\forall n. (gcd[m, n] \mid 0) \wedge (gcd[m, n] \mid n)$  is clearly true. Otherwise, assume (the inductive hypothesis) that  $\Phi(k)$  is true for natural numbers  $k < m$ . If  $n < m$ , then  $gcd[m, n] = gcd[n, m]$ , in which case the inductive hypothesis implies that  $gcd[m, n]$  divides both  $m$  and  $n$ . If  $m \leq n$ , then, since  $0 \leq n \bmod m < m$ , we have by the inductive hypothesis that

$$\forall N. (gcd[n \bmod m, N] \mid n \bmod m) \wedge (gcd[n \bmod m, N] \mid N).$$

In particular, for  $N = m$ , we find that  $gcd[m, n] = gcd[n \bmod m, m]$  divides both  $n \bmod m$  and  $m$ . But  $n = qm + (n \bmod m)$  for some natural number  $q$ ; therefore  $gcd[m, n]$  divides  $n$  also. We have shown that  $gcd[m, n]$  divides both  $m$  and  $n$ , for all  $n$ , so the inductive step is proved.

- (c) This can also be proved by induction over the natural numbers. In this case,

$$\Phi(m) \equiv \forall n. (d \mid m) \wedge (d \mid n) \supset d \mid gcd[m, n]$$

will be shown to hold for all  $m$ . For  $m = 0$ ,  $d \mid m$  is true,  $gcd[m, n] = n$ , and therefore  $\Phi(m)$  is equivalent to  $\forall n. d \mid n \supset d \mid n$ , which is true. For the inductive step, we assume  $\Phi(k)$  for 311 natural numbers  $k < m$ . If  $n < m$ , then  $\Phi(n)$  states the desired result. Otherwise,

$$\Phi(n \bmod m) \equiv \forall N. (d \mid n \bmod m) \wedge (d \mid N) \supset d \mid gcd[n \bmod m, N].$$

Letting  $N = m$ , we have  $(d \mid n \bmod m) \wedge (d \mid m) \supset d \mid gcd[n \bmod m, m]$ . By the definition of  $gcd$ ,  $gcd[n \bmod m, m] = gcd(m, n)$ , and, again writing  $n = qm + (n \bmod m)$  we see that  $(d \mid m) \wedge (d \mid n) \supset (d \mid n \bmod m) \wedge (d \mid m)$ . Thus,  $d \mid gcd[m, n]$ .

2. For any predicate  $P$ , the predicate  $Q(y) \equiv \neg P(y, y)$  is not representable. For if it were, we would have  $\forall y. \neg P(y, y) \equiv P(a, y)$ , and therefore  $\neg P(a, a) \equiv P(a, a)$ , a contradiction.

3. (a)  $L = 1 + 1(0 + 1)^*1$ , and  $L' = \epsilon + (10 + 00)^*1$ , since it contains the empty string and the set of strings that end in 1 and have 1 only in odd-numbered positions. Since we can denote  $L$  and  $L'$  by regular expressions, they are both regular sets.

- (b) One solution is  $L = 10^*$ . Then  $L'$  is the set of strings ending in 1 which may have 1 only in positions  $2^m$ ,  $m \geq 0$ . Suppose  $L'$  were regular. By the pumping lemma, there is a constant  $n$  such that if  $z \in L'$ , and  $|z| \geq n$ , we may write  $z = uvw$  in such a way that  $|uv| \leq n$ ,  $|v| \geq 1$ , and for all  $i \geq 0$ ,  $uv^i w \in L'$ . But given  $n$ , suppose we choose  $z = 0^{m-1}1$ , where  $m$  is a power of 2 greater than  $n$ . Then, if  $z = uvw$ ,  $|uv| \leq n$ , and  $|v| \geq 1$ , we must have  $v = 0^k$  for some  $1 \leq k \leq n$ , and  $uv^2w = 0^{m+k-1}1$ . But  $0^{m+k-1}1 \in L'$  if and only if  $m+k$  is a power of 2, which contradicts the fact that  $m$  is a power of 2 and  $1 \leq k \leq n < m$ .

(c) If  $L'$  is recursive, then we can decide whether or not the string representing some positive integer  $i$  is in  $L$  by checking whether or not the string  $0^{i-1}1$  is in  $L'$ . If  $L$  is recursive, then we can decide whether a string is in  $L'$  by checking whether each member of the set that it represents is in  $L$ .

(d) We can observe that the method used in (c) for recognizing strings in language  $L$  or  $L'$ , given a machine to recognize the other, may sometimes not halt, but in those cases the string being tested is not in the language in question.

Another way to answer (d) is to construct a machine that enumerates  $L$  or  $L'$ , given such a machine for the other. If  $L'$  is r.e., then given a string representing the positive integer  $i$ , we can enumerate  $L'$  and stop if we see the string  $0^{i-1}1$ . If  $i$  is in  $L$ , then  $0^{i-1}1$  is in  $L'$ , so this process will eventually halt. If  $L$  is r.e., then construct a machine that generates the members of  $L$  in some order, say  $a_0, a_1, a_2, \dots$ , and after each one, say  $a_i$ , outputs the strings  $F(A)$  for all the sets  $A$  that contain  $a_i$  and all combinations of strings from  $\{a_0, a_1, \dots, a_{i-1}\}$ . Any string  $s \in L'$  is  $F(A)$  for some finite set  $A \subseteq L$ , and there must be some time  $3t$  which all the members of  $A$  have been generated; then the machine just described will output the string  $F(A)$ .

⋮



**Problem 1. (20 points total)**

Let  $G$  be a graph with vertex set  $V$  and edge set  $E$ . Consider the graph  $G^* = (V^*, E^*)$ , where  $V^*$  is the set of ordered pairs  $V \times V$ , and where  $\{(v, w), (v', w')\}$  is in  $E^*$  if and only if  $v = v'$  or  $\{v, v'\}$  is in  $E$ .

Part a. (5 points) If the largest clique in  $G$  has  $k$  vertices, what is the size of the largest clique in  $G^*$ ?

Part b. (15 points) Assuming that  $P \neq NP$ , prove that there is no deterministic polynomial-time algorithm with the following property: Given a graph  $G$  whose largest clique contains  $k$  vertices, the algorithm finds a clique of  $G$  having more than  $k - \sqrt{k}$  vertices.

**Problem 2. (40 points total)**

This problem deals with the scheduling of unit-time tasks on identical processors, subject to precedence constraints. We are given a set  $T$  of tasks and a relation  $P \subseteq T \times T$ . If  $(t, t') \in P$  we say that  $t$  precedes  $t'$  and write  $t \prec t'$ . The precedence relation  $P$  need not be transitive.

A task  $t$  is said to be at level  $l$  if  $l$  is the length of the longest precedence sequence of the form  $t = t_0 \prec t_1 \prec \dots \prec t_l$ . In particular, a *sink* (which precedes no other tasks) is at level 0. We assume that  $P$  contains no cycles of the form  $t_0 \prec t_1 \prec \dots \prec t_l \prec t_0$ ; otherwise the notion of level would not be defined. A task is at level 1 if it precedes only sinks and if it precedes at least one sink. In general, a task is at level  $l$  if it becomes a sink when all tasks at levels  $< l$  are removed from  $T$  and  $P$ .

Part a. (10 points) Sketch an implementation of an algorithm that determines the level of each task in  $T$ , assuming that  $T = \{1, 2, \dots, n\}$  and that  $P$  is given as a list of  $m$  pairs  $(t, t')$  of integers. Your solution should use  $O(m + n)$  time and space. Describe the data structures used by your implementation.

Part b. (10 points) A schedule for  $T$  and  $P$  on  $p$  processors is a partitioning  $T = T_1 \cup \dots \cup T_s$  into disjoint sets  $T_i$  such that  $\|T_i\| \leq p$  for all  $i$ ; furthermore if  $t_i \in T_i$  and  $t_j \in T_j$  and if  $t_i \prec t_j$ , then  $i < j$ .

A task  $t$  is said to be a *source* if no other task precedes it. The *greedy algorithm* for scheduling is defined as follows:

```

r := 0;
while T ≠ ∅ do
  begin r := r + 1;
    k := min(p, the number of source tasks in T);
    T_r := any set of k source tasks, having the maximum
           possible level among all current source tasks;
    remove all elements of T_r × T from P;
    remove all elements of T_r from T;
  end.

```

In other words, at each unit of time we schedule as many as possible of the tasks that are not prohibited from being scheduled by the remaining precedence constraints. If more than  $p$  such tasks are available, we choose them arbitrarily but in order of decreasing level. The idea is that, all other things being equal, it seems best to schedule high-level tasks first.

Several conditions on  $P$  are known to be sufficient to guarantee that the greedy algorithm produces an optimum schedule, i.e., a schedule with minimum length  $s$ . However, greediness does not always pay: Find an example of a precedence relation when there are  $p = 3$  processors, where an optimum schedule would not be found by any implementation of the greedy algorithm.

Part c. (**20** points). One of the **cases** for which the greedy algorithm is known to be optimum occurs when  $\mathcal{P}$  defines a forest, namely when each task precedes at most one other task. (You need not prove this.)

Sketch the details of an implementation that performs such a greedy algorithm, assuming that  $\mathcal{P}$  defines a **forest** in this sense. Define and use appropriate data structures such that your implementation is  $O(n)$  in time and space.

Spring 1982  
ARTIFICIAL INTELLIGENCE

**1. [45]** Expert System, and related issues

This question centers around the task of writing an Expert System to (help) plan the courses an incoming MS/AI student should take during his stay at Stanford.

The listing below approximates the requirements for a MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE degree from Stanford. Assume all courses are 3 units, except for CS 293 and 390. An asterisk ( "\*" ) means "take for a letter grade, end pass."

-----  
I. AI pert:

- cs 222
- CS 223
- one of CS 275, CS 276, CS 226, or CS 227.

II. Classical hardware and software.

- CS 142

III. Theoretical computer science.

- one of CS 156 or CS 206.

IV. Practicum.

- CS 293 (27 units) or CS 390 (27 units).
- 

A) **[8]** Representation

(i) **[4]** Draw an AND-OR tree which represents the CS/MS requirements I-IV. (shown above).

Note that you will be expected to use this representation when answering the following parts of this question.

(ii) **[3]** What does each leaf represent? each non-terminal node? each arc?

(iii) **[1]** What does it mean (in real-world terms) to achieve the top node of the tree?

6) **[4]** Multiple Parents

(i) **[3]** Write a (small) change to the MS/AI requirements (I-IV above) which would cause a node to have more than one parent.

(ii) **[1]** What problems now arise?

C) **[11]** Complications

You now realize there are other requirements, viz. you must:

-----  
V. graduate in not more than 2 years.

VI. pass at least 54 units.

VII. maintain an overall GPA of at least 3.00 in these CS Dept courses.

("GPA" means "grade-point average": it is the average of all graded units.)  
-----

Below we consider how you would represent these requirements as well.

(1) **[3]** How easy or hard is this to do using the AND-OR trees? Why?

(ii) **[8]** Outline how you would represent all of these constraints (I-VII) using an ATN (augmented transition network).

D) **[2]** Searching

Suppose we assign a number to each leaf in the tree, representing its estimated worth (expected grade in that course). Will alpha-beta

## Spring 1982 - Artificial Intelligence

pruning help us search that tree? How/why? Estimate the **benefits**.

### E) [9] Heuristics

There are still many options, many possible programs that satisfy the MS/AI requirements (I-VII above). To **help you** decide on a programme of study, you devise an "objective function" that helps guide you in your search. This function will contain informal (heuristic) knowledge. Suppose you decide to represent this knowledge as condition-action rules. List 3 such pieces of knowledge. For each, indicate the (rough) percentage of the nodes in the AND-OR tree to which it applies.

### F) [5] Blackboard Model

We have, of course, simplified or eliminated many of the issues **which complicate** scheduling. We also assumed that **all** of the information is available at once. Consider now the issue of planning your schedule, one term at a time, based on incremental data. For this we will use the blackboard model. The y-axis represents levels of abstraction. What are some **of** these? What does the x-axis represent? Name 3 appropriate **KSs** which should be used.

### G) [6] Other representations

If someone asks you to guess, "**Is** CS223 being taught by someone in the department?", and you didn't even know what CS223 was, you still might guess the default answer "Yes."

(i) [4] Choose two of the following representation schemes, and (for each one) represent enough of the world to show how such default reasoning would automatically take place:

- production rules,
- predicate calculus
- units

(ii) [2] Suppose that, over the next few years, we hire outside people to teach most of our classes. For each of the two representations you chose above, make whatever modifications are necessary to reflect that change.

2. [15] This question **concerns** the scientific motivations of various "AI researchers" and the methods they **might** use.

A. [5] Roughly speaking, there appear **to** be among AI researchers two classes: the "artifact builders" and the "psychologists" (i.e. those concerned with modeling human behavior).

Briefly, what leads the "psychologists" to believe that AI research has something to offer them in pursuit **of** their objectives? What is their most fundamental modeling assumption? Why **is** a computer appropriate for **their** work?

B. [3] Human thought "**runs**" in brains; machine thought "**runs**" in computers. To what extent is progress in the "psychology" part **of** AI attributable to the **correct** computer modeling **of** brain **function** and structure? Cite examples.

C. [4] Consider the DENDRAL program **for** inferring molecular structure hypotheses from mass spectral data. What was its main contribution to the art of "artifact building"? To the science of human behavior?

D. [3] If the DENDRAL project had sought to emphasize the latter rather than the former (or vice versa), how would the chosen research method have differed?

**Problem 1. [20 points.]**

The balanced ternary representation of an integer has the form:

$$\sum_i d_i 3^i, \quad \text{where } d_i \in \{0, 1, -1\}.$$

Call each  $d_i$  a **trit**.

(8) **[12 points.]**

Design a half-adder for a number in this representation. (A half-adder takes two trits as input and produces the resulting sum and carry trits. You may use AND, OR, and NOT gates. Give (i) your encoding of a trit; (ii) a table showing sum and carry as a function of input trits; and (iii) the final circuit.

(b) **[3 points.]**

Solve (a) using at most 8 AND and OR gates total, with any number of NOT gates (inverters). The AND and OR gates may have any number of inputs. (An 8-gate solution to (a) will automatically earn the 3 points for this question.)

(c) **[5 points.]**

Using half-adders, construct a full adder. This is a three-input two-output device that takes two trits and a carry trit and produces a sum and a carry trit. Do not assume any particular representation for trits in part (c).

**Problem 2. [10 points.]**

The figures below show timing diagrams for the DEC Unibus (tm). The first diagram is that for transferring data from slave to master. The second shows the protocol for transferring control.

Figure 1:

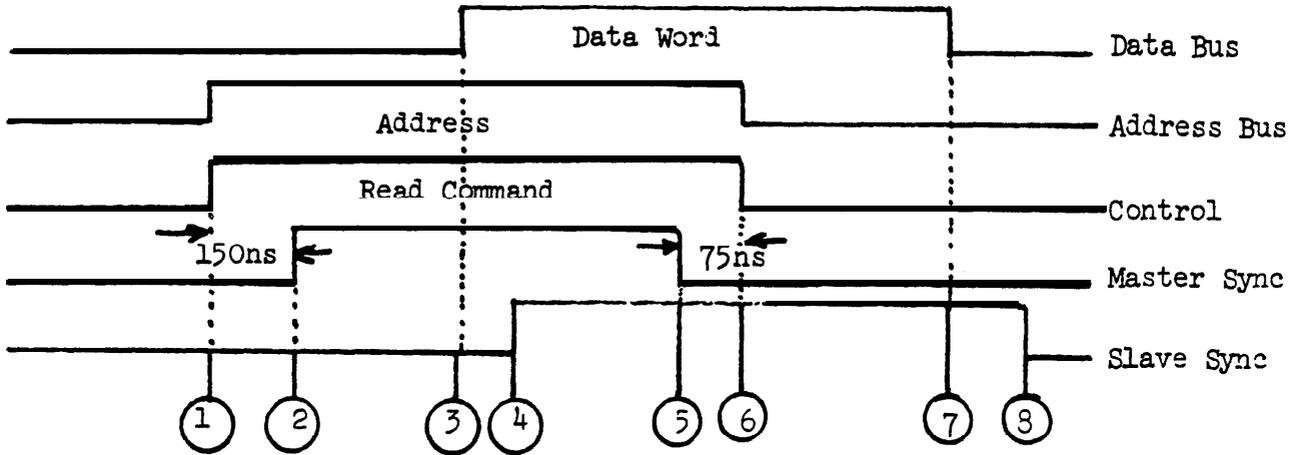
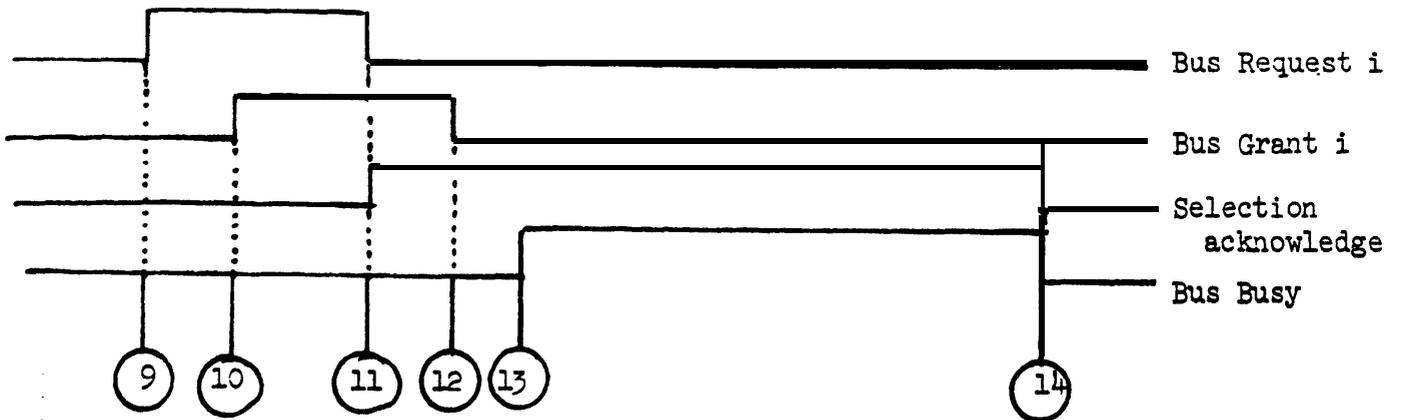


Figure 2:



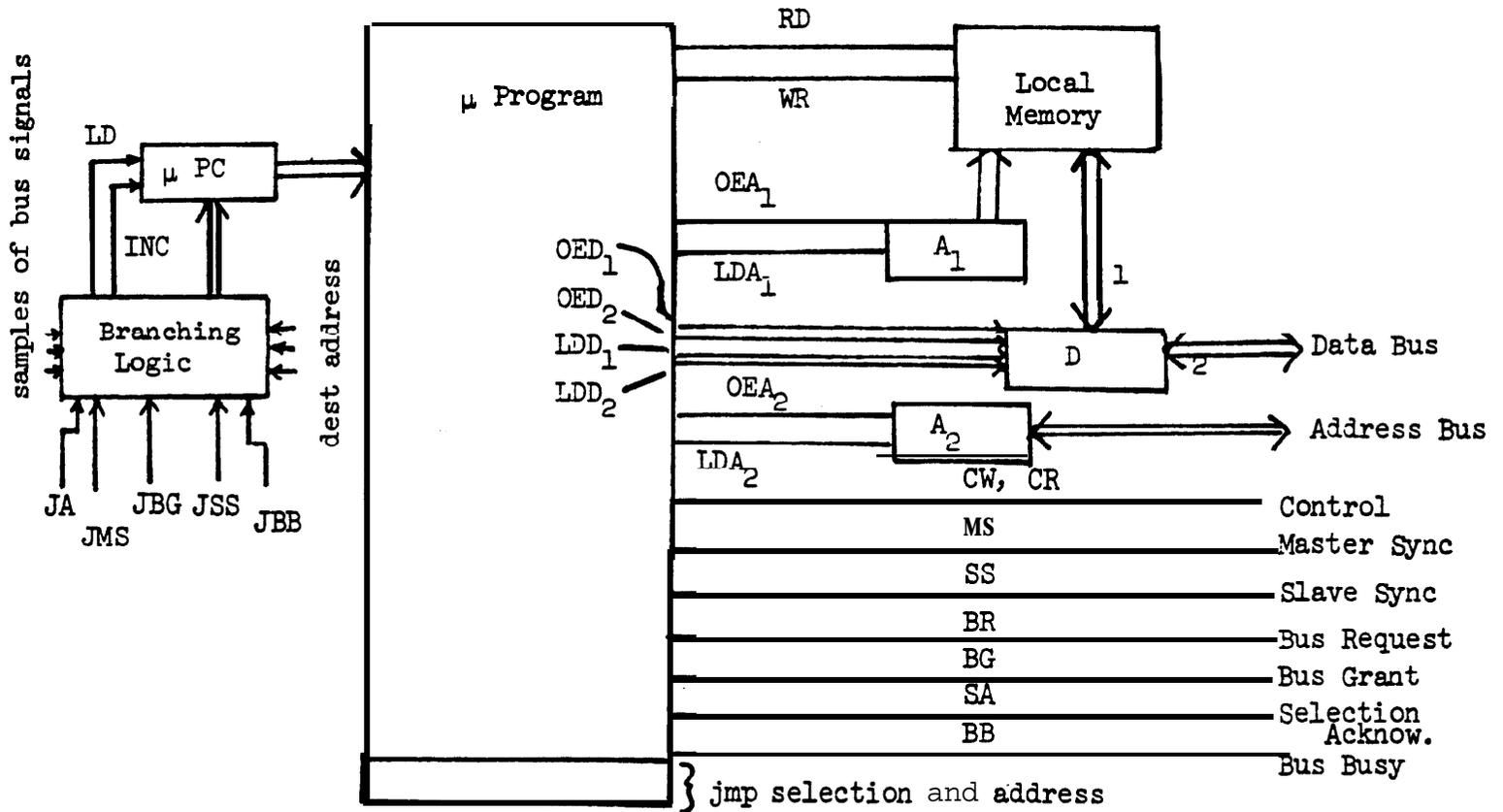
Match the statements below with the appropriate points on the timing diagrams:

- a. Ok, the bus **is** all yours.
- b. **Master**, your data is waiting.
- c. Quick, I need the bus.
- d. Ok, I got the bus, thanks.
- e. **Attention** slaves! **All the** details on a job for **one** of you are **ready**.

**Problem 3. [20 points.]**

The diagram below shows some of the details of a microprogrammable device designed as a DMA controller (see problem 2). It communicates over a bus using the Unibus protocol.

The microcode is fully horizontal. Each microinstruction is represented by listing all active control lines and jump destination (if any). The clock for the  $\mu PC$  has a cycle of 75ns.



Note:

- JSS: jump if Slave Sync is active
- JMS: jump if Master Sync is active
- JBG: jump if Bus Grant is active
- JBB: jump if Bus Busy is active
- JA: jump always
- RD: read local memory word
- WR: write local memory word
- OE $x$ : enable output from register  $x$
- LD $x$ : load register  $x$

**Control Word Codes .**

CW: transfer data from master to slave

CR: transfer data from slave to master

**Simplifying Assumptions**

(i) Accessing local memory (1 read cycle) can be assumed to complete in 2 clock cycles. All set-up times are at most 75ns.

(ii) A signal stays active for exactly 1 full clock-cycle if **activated** by a p-instruction. There are no glitches during renewals in subsequent instructions.

(iii). The device will not be asked to perform another activity while busy.

(a) [5 points.] A segment of microprogram is listed below. Briefly describe what it does.

```
OEA1
OEA1; RD
OEA1; RD
OEA1; RD; LDD1
OED2
LOOP: OED2; SS; JMS LOOP
```

(b) [15 points.] This DMA controller has received a command to transfer 1 word of data from local memory over the bus. Register  $A_1$  holds the address of the word in the local memory. Register  $A_2$  holds the bus address of the intended recipient. Write a sequence of microcode commands to perform the transfer.

Hint: The steps in the transfer are:

- get the data into register  $D$
- get control of the bus
- transfer the data word
- release the bus

**Problem 4. [10 points.]**

Given a disk with the following characteristics:

access time (seek time) 60 ms average (uniformly distributed seek distances)

410 tracks

48 sectors per track

256 bytes per sector

3600 RPM

(a) [3 points.]

What is the average rotational latency?

(b) [3 points.]

Assuming that the choice of sector is uniformly distributed independent of past choices, what is the average amount of time needed to read a sector?

(c) [4 points.]

What is the average time if 70% of the seeks are to the same track as the last operation? (The remaining 30% of the time the requests are as per (b). The sector within a track is always uniformly distributed independent of past behavior. Assume that the controller is fast enough to read only every third sector (1:3 interleaving).

Numerical Analysis

1) Iteration Methods [20 points]

a) [3 points]

If  $a > 0$ ,  $k > 1$  then  $\sqrt[k]{a}$  can be calculated by solving  $x^k - a = 0$ . Give the iteration formula produced by applying Newton's Method to this problem.

b) [14 points]

Prove that this iteration formula converges to  $\sqrt[k]{a}$  for all  $x_0 > 0$ .

(Hint: Consider  $0 < x_1 < \sqrt[k]{a}$  and  $x_1 > \sqrt[k]{a}$  as separate cases in analyzing  $\sqrt[k]{a} - x_1$ ).

c) [3 points]

What order of convergence does this method display near  $\sqrt[k]{a}$ ? Explain why.

2) Interpolation [20 points]

Consider producing a **6-decimal** place table of values for  $f(x) = \log_{10} x$ , and **assume** cubic interpolation is to be used. The interpolation error should be less than the rounding error in 6 decimal place numbers.

a) [12 points]

Suppose the values of  $f(x)$  are to be printed for  $x = 1.0, 1+h, 1+2h, \dots, 10.0$ . What is an appropriate choice **of h**?  $h$  should be the largest possible number which would be convenient to use.

b) (8 points]

Show that a larger value of  $h$  could be chosen if the table were printed for  $x = 1.0, 1+(1/2)h, 1+h, 1+(3/2)h, 1+2h, 1+3h, \dots, 10.0$ . What is this larger value of  $h$ ? Note that we increase  $h$ , thus decreasing the total number of table values and add just 2 extra values at the beginning of the table.

3) Linear Algebra Quickies [20 points]

In solving any of the following problems, you can use the results from preceding parts whether or not you have proved them.

Given:  $A, B$  are invertible  $n$  by  $n$  matrices

$b, x, \bar{x}, y$  are  $n$ -vectors

$$\|A\| \equiv \max_{\|x\| \neq 0} \frac{\|Ax\|}{\|x\|} \text{ for any vector norm } \|\cdot\|$$

i) [4 points]

a) Calculate  $\|I\|$  where  $I$  is the matrix with 1's on the diagonal and 0's elsewhere.

Show that

b)  $\|Ax\| \leq \|A\| \|x\|$

c)  $\|AB\| \leq \|A\| \|B\|$

d)  $K(A) = \|A\| \|A^{-1}\| \geq 1$

ii) [5 points] If  $\|A\| < 1$ , then prove

a)  $(I+A)$  is invertible

b)  $\|(I+A)^{-1}\| \leq \frac{1}{1-\|A\|}$

Hint for b): Since  $\max_{\|x\| \neq 0} \frac{\|Bx\|}{\|x\|} = \max_{\|x\|=1} \|Bx\|$  and

since the set  $\{x \mid \|x\| = 1\}$  is compact, then there exists

$y, \|y\| = 1$  such that  $\|(I+A)^{-1}y\| = \|(I+A)^{-1}\|$ . Look at  $x$  where  $x = (I+A)^{-1}y$ .

iii) [2 points] If  $\|B\| \leq \frac{1}{\|A^{-1}\|}$  then show

a)  $(A+B)$  is invertible

b)  $\|(A+B)^{-1}\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}B\|}$

iv) [2 points] If  $Ax=b$  then give upper and lower bounds for  $\|x\|$  in terms of  $\|A\|$ ,  $\|A^{-1}\|$ , and/or  $\|b\|$ .

v) [7 points] Assume  $(A+\delta A)\bar{x} = b$  where  $\|\delta A\| = \epsilon \|A\|$  and  $\epsilon K(A) < 1$

a) Give an upper bound for  $\|b-A\bar{x}\|$  in terms of  $\epsilon$ ,  $\|A\|$ , and  $\|\bar{x}\|$ .

b) If  $\|\bar{x}\|$  achieves the lower bound implied by the result in part iv), show that

$$\|b-A\bar{x}\| \leq \frac{\epsilon}{1-\epsilon} \|b\|$$

c) If  $\|\bar{x}\|$  achieves the upper bound implied by the result in part iv), show that

$$\|b-A\bar{x}\| \leq \frac{\epsilon \|b\| K(A)}{1-\epsilon K(A)}$$

d) Which of the upper bounds on the residual  $\|b-A\bar{x}\|$  in b) and c) is smaller?

e) What is an upper bound for the scaled residual  $\frac{\|b-A\bar{x}\|}{\|\bar{x}\|}$ ?

- (7) 1: Suppose that you are writing a compiler for PASCAL. You are at the stage of designing the symbol table routines.
- (2) a. What kinds of symbols would you represent in the symbol table?
- (5) b. For each kind mentioned above, what information would you store about a symbol of that kind? Present your answer in the form of commented record type-definitions (syntax ala PASCAL).
- (12) 2. (2) a. How do PASCAL and FORTRAN differ in the way each handles function/subroutine calls?
- (10) b. Suppose the linker/loader on the machine you use behaves as described below.

Relocatable Segment:

Definition Table	
<b>DName</b>	<b>DVal</b>
symbolic name	addr. within segment
Use Table	
<b>UName</b>	<b>UVal</b>
symbolic name	(absolute addr. filled in by linker)
Code and/or Data	

The Definition Table lists all symbols that might be referenced by external routines. The Use Table lists all externally defined symbols used within this segment. Whenever such a symbol is used, it is referenced indirectly through the Uval field.

The linker/loader (invoked prior to execution) does the following:

- i. Allocates space for each segment.
- ii. Adjusts all addresses in the segment by a relocation offset.
- iii. Fills in the addresses of external objects in the Use Table.

- (5) 1. Suppose you have just compiled a set of FORTRAN subroutines. You now wish to assign the data and code to relocatable segment(s) in preparation for linking/loading. How would you divide code and data among segments? What symbols would you define for the linker/loader?
- (5) ii. How would you do it if the language were PASCAL?
- (8) 3. A time-sharing system is to run a mix of interactive and batch jobs. The powers-that-be have asked your opinion on some potential scheduling policies. For each policy, list classes of jobs that would receive either especially favorable or unfavorable treatment and why.
- (2) 1. Round-robin for all runnable jobs. Load jobs on a first-come, first-serve basis.
- (2) ii. Run a job until it completes or blocks. Choose the smallest (main memory used) runnable jobs to run or load next. Never preempt, even if a shorter job becomes runnable.
- (2) iii. Run the highest, priority runnable job until it completes or blocks. **Higher** priority jobs **may** preempt lower priority jobs. Priority decreases with the amount of CPU time used.
- (2) iv. Like iii, but the priority decreases with the amount of CPU time used since the last **I/O** request (after an I/O request, a job gets maximum priority).
- (14) 4. A simple form of interprocess communication is event signalling using the following two primitives.

sleep (event) - suspend the calling process until the event occurs.

wakeup(event) - resume (unsuspend) each process sleeping on this event.

These primitives have been used in a least one operating system to control access to shared data structures as follows:

To gain exclusive access  $p^{\wedge}$

```
while  $p^{\wedge}$ .locked do sleep (p);  
 $p^{\wedge}$ .locked := true;
```

To release  $p^{\wedge}$

```
 $p^{\wedge}$ .locked := false;  
wakeup (p)
```

- (2) a. Why is there a "while" loop for executing "sleep" rather than a simple "if" test?
- (4) b. What is a logical problem with the use of these two constructs without further assumptions about the underlying implementation, i.e. describe how they can fail. Describe how to fix this problem considering both uniprocessor and multiprocessor machines.

(4) **c.** Synchronization control tends to perturb the effects of scheduling decisions. How? Compare the effect of the above constructs with that on semaphores with respect to scheduling in a mixed batch and timesharing system.

(4) **d.** Suppose we add an operation "Destroy (process)" that instantaneously removes the process from the system. What logical problem does that raise with the use of these constructs? How is this handled or solved by semaphores and monitors?

(6) **5.** A message-based system might provide two message primitives

Send (process, message area)

process := Receive (message area)

where Send transmits the message and returns **immediately**, and Receive blocks until a message is available and then returns with the oldest queued message copied into the specified message area. In particular, messages are assumed to be copied out of the message area after Send returns and copied in **when Receive** returns.

(3) **a.** Assuming infinite space for internal message buffering, can a set of processes deadlock using these two primitives? Justify your answer.

(3) **b.** Assume a finite space for internal messages buffering and that Send blocks until a message buffer is available. Describe a set of processes that use these primitives and do something useful (e.g. producer-consumer) and don't deadlock with an-infinite buffer pool but can deadlock with **a** finite buffer pool. Describe how they can deadlock.

(13) **6.** Consider the grammar G,

$S \rightarrow SUS \mid SVS \mid W$

(2) **a.** Show that this is an ambiguous **grammar**.

(4) **b.** Give an unambiguous grammar for the same language.

(1) **c.** For what **K** is the original grammar LR(K)? Justify your answer.

(3) **d.** Describe how assigning precedence to the terminals and productions of G can help to resolve the ambiguity of G. I.e. describe a precedence and what ambiguity is resolved by giving the parse tree for wuwvw.

(3) **e.** Describe an ambiguity of G that cannot be resolved with precedence and show how to resolve it with associativity.

12 point value

The following program **computes the** sum modulo  $b^n$  of two **numbers represented** as  $n$ -element arrays  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$  of digits in some **integer** base  $b \geq 2$ . Only natural numbers are **used** in the computation. Give a formal statement of an appropriate loop invariant and a formal statement of the input and output specifications.

```

i ← 1; carry ← 0;
while i ≤ n do
    temp ← carry + xi + yi;
    zi ← temp mod b;
    carry ← temp ÷ b;      (integer division)
    i ← i + 1
end
    
```

10 point value

Outline two algorithms demonstrating **different** techniques for establishing **whether** or not a propositional formula is a tautology. Try to pick methods that are as distinct as possible. Show how both work for the formula

$$\neg(P \supset Q) \supset (P \wedge \neg Q)$$

10 point value

Give **five** statements that are equivalent to " $L$  is a **regular** set." For full credit, the substance of your statements should be significantly **different** from **one** another **and** from the statement " $L$  is a **regular** set."

13 point value

If  $L$  is a language **over** some alphabet  $\Sigma$ , and  $a$  is in  $\Sigma$ , define  $\frac{dL}{da} = \{w \mid aw \text{ is in } L\}$ .

1. 3 point value - Compute  $\frac{dL}{da}$  and  $\frac{dL}{d1}$ , where  $L = \{0^n 1^n \mid n \geq 0\}$ .
2. 10 point value - Show that if some language  $L$  is regular, so is  $\frac{dL}{da}$ ; if  $L$  is context-free, so is  $\frac{dL}{da}$ . Informal but **clear** arguments are acceptable.

15 point value

Let  $L_1$  and  $L_2$  be languages contained in  $(0+1)^*$ . A reduction from  $L_1$  to  $L_2$  is a total recursive function from  $(0+1)^*$  to  $(0+1)^*$  such that  $f(w)$  is in  $L_2$  if and only if  $w$  is in  $L_1$ .

Let  $M_1, M_2, \dots$  be an **enumeration** of Turing **machines** with input **alphabet**  $\{0, 1\}$  that is "standard," **in the sense** that **there** are algorithms to translate from **the integer**  $i$  (**expressed** in binary) to **the usual** "next state, next symbol, next head **move**" notation for Turing machines and back. Let  $L(M_i)$  be the language accepted by  $M_i$ . **Define**

$$P = \{ \text{binary integers } i \mid L(M_i) \text{ contains at least one member} \}$$

$$Q = \{ \text{binary integers } i \mid L(M_i) \text{ contains at least two members} \}$$

Give a reduction from  $P$  to  $Q$  and **one** from  $Q$  to  $P$ .

. Spring 1982 - Theory of Computation

**HINT:** To reduce  $P$  to  $Q$ , **assume** that given  $i$ , we can construct **the** usual Turing machine **notation** for  $M_i$ . Informally **describe** a construction on this **machine** to produce another **machine** that is  $M_j$  for some\*  $j$ . Arrange that  $M_j$  accepts two or more strings if and only if  $M_i$  accepts one or more.

**Solution to Problem 1.** Let  $G$  have  $n = \|V\|$  vertices.

(a) For every clique  $C$  in  $G$ ,  $C \times V$  is obviously a clique in  $G^*$ . Furthermore every maximal clique  $C^*$  in  $G^*$  has the form  $C \times V$  for some clique  $C$  in  $G$ ; for if  $(v, w)$  is in  $C^*$ , all pairs  $v \times V$  are adjacent to all elements of  $C^*$ . Hence the maximum clique in  $G^*$  has  $kn$  vertices.

(b) Suppose such an algorithm  $A$  exists; we will then solve the,  $\mathcal{NP}$ -complete maximum clique problem in polynomial time, proving that  $P = \mathcal{NP}$ .

Given a graph  $G$  and an integer  $k$ , we wish to determine in deterministic polynomial time whether  $G$  contains a clique of size  $k$ . Apply algorithm  $A$  to  $G^*$ , obtaining a clique  $C^*$  of size  $k^*$ . Note that  $G^*$  is polynomial in the size of  $G$ . We now take the projection of  $C^*$  on its first components; this is a clique  $C$  in  $G$ , and it contains at least  $k^*/n$  vertices. We claim that  $G$  has a clique of size  $k$  if and only if  $\|C\| \geq k$ , so we have constructed an algorithm that answers the desired question.

To-prove the claim, it is obvious that  $\|C\| < k$  if  $G$  has no clique of size  $k$ . Otherwise  $G^*$  has a clique of size  $\geq kn$ , by part (a); hence we have  $k^* \geq kn - \sqrt{kn}$  by assumption. Thus  $\|C\| \geq k^*/n \geq k - \sqrt{k/n} > k - 1$ . Q.E.D.

**Solution to Problem 2.**

(a) We shall use the ideas of topological sorting (Knuth §2.2.3) in reverse.

For each task  $t$  we initialize  $\text{count}[t] =$  the number of tasks it precedes, and build the set  $\text{pred}[t]$  of its predecessors. Then we proceed as follows:

```

 $l := -1; Z :=$  set of all  $t$  with  $\text{count}[t] = 0;$ 
while  $Z \neq \emptyset$  do
  begin  $Y := \emptyset; l := l + 1;$ 
  for all  $t \in Z$  do
    begin  $\text{level}[t] := l;$ 
    for all  $s \in \text{pred}[t]$  do
      begin  $\text{count}[s] := \text{count}[s] - 1;$ 
      if  $\text{count}[s] = 0$  then include  $s$  in  $Y;$ 
      end;
    end;
   $Z := Y;$ 
end.

```

The sets  $\text{pred}[t]$ ,  $Y$ , and  $Z$  are conveniently represented as linear lists.

(b) For example, let  $1, 2, 3 \prec 4 \prec 5, 6, 7$  and let  $8 \prec 9, 10, 11, 12$ . Then there is a way to complete the schedule in four units of time, starting with  $\{1, 2, 8\}$ . But the greedy algorithm must start with  $\{1, 2, 3\}$  (since these are the only elements at level 2), and that leaves only two usable tasks for the next step in the schedule.

(c) After calculating levels as in (a), we can use topological sorting in the forward direction, with  $\text{count}[t]$  now representing the number of tasks that precede  $t$ , and with  $\text{succ}[t]$  the unique successor of  $t$  (or zero if  $t$  precedes no other task). To initialize for this algorithm, we construct for each level  $l$  the set  $Z[l]$  of all tasks at that level whose count is zero, and the set  $Q$  of all levels  $l$  such that  $Z[l]$  is not empty. The trick to keep within linear time is to avoid sorting by level, and to avoid searching for nonempty levels. We use the fact that the successor of a task at level  $l$  is at level  $l - 1$ . Tasks that become sources at the next unit of scheduled time are put into sets  $Y[l]$  analogous to  $Z[l]$ ; and  $X$  is to  $Y$  as  $Q$  is to  $Z$ :

```

 $r := 0;$ 
while  $Q \neq \emptyset$  do
  begin  $r := r + 1; k := 0; T_r := \emptyset; X := \emptyset;$ 
  while  $k < p$  and  $Q \neq \emptyset$  do
    begin  $l :=$  largest element of  $Q;$ 
     $Y[l] := \emptyset;$ 
    while  $k < p$  and  $Z[l] \neq \emptyset$  do
      begin  $t :=$  any element of  $Z[l];$ 
      delete  $t$  from  $Z[l];$  include  $t$  in  $T_r; s := \text{succ}[t];$ 
      if  $s \neq 0$  then

```

```
    begin count[s] := count[s] - 1;
    if count [s] = 0 then include s in Y[l];
    end;
    if Y[l] ≠ ∅ then include l in X;
    if Z[l] = ∅ then delete l from Q;
    end;
end;
for all l ∈ X do
    begin include l - 1 in Q;
    Z[l - 1] := Z[l - 1] ∪ Y[l];
    end;
end.
```

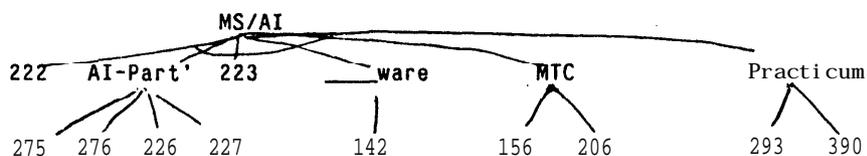
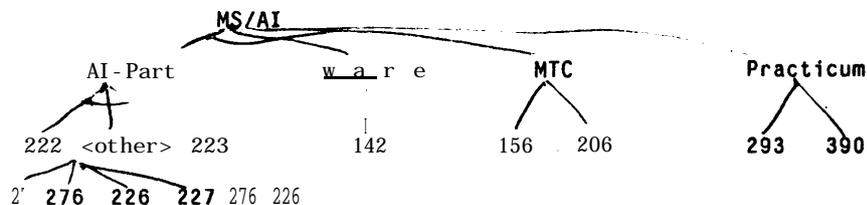
Note that  $Q$  can be maintained in sorted order without violating the linear time constraint, because the elements of  $X$  are in sorted order, and all elements of  $X$  are  $\geq$  all elements of  $Q$ .

AI Answers

1. [45] Expert System, and related issues

A) [8] Representation

(i) [4] [Both of the trees below were acceptable.  
The first was more commonly used, **the** latter slightly better for (1d)]



(ii) [3] Each leaf node refers to the chore of (taking and passing) a particular course.

The internal nodes refer to area-wide requirements.

The links represent the subgoal relation.

(iii) [1] It means **you** have satisfied the requirements needed to graduate.

B) [4] Multiple Parents

(i) [3] Let some course satisfy more than one requirement.

For example, make CS 206 a requirement for the AI part.

(That is, change "I." to now include

- CS 206.)

(ii) [1] We can no longer call this a "tree".

There are no problems representing this.

Problems might arise in using it, of course: e.g., counting CS206 twice when figuring total units.

C) [15] Complications

(i) [3] It is very difficult:

First, we'd have to include other classes, to **make the** 64 units.

Next, consider all the ways of attaining 64 units.

Each of these combinatorially large cases would have to be represented in the **tree!**

A similar **problem** would arise when worrying about V.

**Multiple** copies of each course would have to be included, one for each term the course is offered.

Only the ones which occurred within two years of the student's entering date would be linked into this (by now gargantuan) tree.

The final problem is how to represent the GPA requirement.

We would first have to encode the grade attained for each course in the tree.

Each term the student could only take that set of courses which maintains his GPA. This would involve having nodes which each represented a set of courses offered in the same term.

Clearly this AND-OR tree structure, so nicely suited for I-IV, is grossly inappropriate for other types of requirements.

(ii) [8]

This question was asking for a sketch of how to design this ATN.

The description below refers to an ATN which ACCEPTS

a proposed schedule. We also gave full credit to those who

## Spring'1982 - Artificial Intelligence (Solutions)

described ATNS which GENERATED acceptable schedules.

Begin by generating a regular Transition Network for the core I-IV requirements

-- this will be similar to the AND/OR tree shown above.

That is, have four nodes in series,

which each (sequentially) examine the proposed schedule.

The j-th node is satisfied if the j-th requirement passes.

(Each of these, in turn, would be a little network,

in basically the obvious fashion.

However, for reasons mentioned below, each OR-junction would have to accept ALL **disjuncts** found, not just the first.)

Add on to this another node, OTHER, which could accept any other course.

Now augment this with a few registers.

The CS-Course-Units register would be a counter,

incremented (by the course's units) each time any CS course is passed.

This update is part of the action taken at the end of any

"course pass" node, embedded in any of the first four

top level nodes mentioned above.

Another similar counter, CS-Grade-Points,

would record the cumulative grade points for these same nodes.

(Like the CS-Course-Units register,

it is updated at the completion of each "primitive" course node.

Its value is incremented by the grade received for this course

times the number of units.)

A third Non-CS-Course-Units register would behave like CS-Course-Units,

but only be incremented **by the** completion of nodes found in OTHER.

A final register, Start-Time, records the time at which you began the program.

(Note these registers are all initiated during the transition leading to

the I node -- to 0 in the first three cases.)

' The Start-Time register is needed to handle the "two years requirement", **#V:**

Associated with each course would be the time at which it was taken:

and only those courses which would be completed

within two years of that Start-Time register value would only be accepted.

The OTHER node, which accepts any course NOT included anywhere in I-IV,

is needed to deal with the "54 units requirement", **#VI.**

(Note you have to be a bit careful to insure that each course is counted once and only once.

This is why each of the first four high level nodes must deal with EVERY course mentioned in that requirement

-- in particular, not just the first member of an OR-junction which succeeds.

Suppose someone takes both CS 275 and CS 226, for example.

If the Ith node used only the CS275 course,

**the** 3 units associated with CS226 would never be counted,

as the I node and the OTHER node would both ignore it.)

All that remains is the final transition, linking that OTHER node with SUCCESS.

Its associated transition test would

i) add together the values of the CS-Course-Units register with the

Non-ES-Course-Units register, and check that the sum was at least 54, and

ii) verify that the ratio of

CS-Course-Units to CS-Grade-Points (which is the final GPA,) is at least 3.00.

### D) [2] Searching

No, **ab** pruning would not be useful for this example, for two main reasons.

First, it is not defined for this situation.

The "GPA value" at a non-leaf node is NOT simply the

GPA value of a bottom node, which has been propagated up.

Second, the tree is too shallow.

Notes: Many people (incorrectly) argued that ab pruning was

inapplicable because this wasn't an adversarial situation.

One certainly could think of scheduling as trying to maximize your reward.

Second, the fact that the "value" of each leaf node was an ESTIMATE is also irrelevant,

## Spring 1982 - Artificial Intelligence (Solutions)

### E) [9] Heuristics

- \* Find nodes which satisfy more' than one requirement, and achieve those whenever possible.
  - This is general, referring to the overall structure of the graph. (Inapplicable to the TREE structure here, but a good idea in general.)
- Take classes which you expect to do well in, especially if your current GPA is precariously near the boundary.
  - This is fairly general, referring to a whole class of primitive nodes. (each corresponding to a class)
- \* If there is no penalty for trying to fulfill some requirement and failing, then attempt to pass this requirement as often as possible.
  - This is relevant only to a small set of nodes.
- \* Avoid any classes which require competency in mathematics.
  - Applies to all leaf nodes, potentially pruning away a small percentage of them.
- Try to take as many classes taught by MIT profs as possible.
  - Again, applies to some leaf nodes.

### F) [5] Blackboard Model

NOTE: Many people seemed to equate KS with data. Wrong.  
KS should be viewed as a procedure' one which is triggered by certain types of input, and uses a particular corpus of facts to generate another hypothesis.  
NOTE: Quick overview of the **BlackBoard**:  
Its basic purpose is to store hypotheses.  
For this application,  
it could house hypothesized schedules, and schedule fragments.  
Each fragment stores a set of "What course at what time" assignments.

The abstraction levels roughly parallel the levels of the AND-OR tree.  
The bottom "primitive" level are particular time assignments to the courses themselves:  
the higher levels refer to classes of courses  
-- e.g., specifying that some requirement has been' satisfied.

One obvious x-axis would represent time in the year  
-- when the class is taken (and passed).  
This is NOT the same as the time in the week the' course is offered:  
that is indeed an important consideration' but not appropriate for the x-axis.

One KS could be devoted to detecting schedule conflicts '  
(it would have access to periodic time schedules).  
Another would note when various basic requirements (I-IV) were satisfied.  
Another would maintain the GPA (watching especially for times when it begins to fall around 3.00)'  
Other **Ks** could help argue for or against a proposed course,  
on the basis of work load for a given term,  
how good the teacher is, your personal interest in  
the material (esp. for additional courses), misc. constraints such as the  
fact that the course is only offered every other year, etc.

### G) [6] Other representations

NOTE: Curiously no one used the "obvious' representation for dealing with default  
-- units (frames).  
NOTE: Almost everyone wrote out universal statements for those default claims --  
what we specifically requested you NOT to do.

#### i) [4] Default Specification

Production Rules:

```
IF (AND (Course crs)
        (= (First-Two-Characters crs) CS))
THEN (Department crs CS)
```

<<above rule is optional -- the important thing is to derive that

Spring 1982 - Artificial Intelligence (Solutions)

(Department CS223 CS) >>

IF (Department crs CS)  
THEN (TeacherInDept crs CS)

<<Notice that, in most Production Systems, it is not a "contradiction," to  
assert that (TeacherInDept CS223 Physics).>>

\*\*\*\*\*

Predicate Calculus:

V crs. ((Course crs) & (= (First-Two-Characters crs) CS))  
    ⊃ (Department crs CS)

<<above sentence is optional -- the important thing is to derive that  
(Department CS223 CS) >>

V crs, dept. (}[](TeacherInDept crs dept) & (Department crs CS))  
    ⊃ (TeacherInDept crs CS)

<<This box, [], is the modal PROVABILITY operator.>>

\*\*\*\*\*

Units:

CS223  
    Isa: CS-Course

CS-Course  
    TeacherInDept: CS

<<Here any individual instance of CS-Course, such as CS223, could still override  
this specification -- if its TeachInDept value was, say, Physics.>>

ii) New defaults

\*\*\*\*\*  
Production Rules:

IF (Department crs CS)  
THEN (Note (TeacherInDept crs CS))

<<Again, this is not "contradicted" by assertions like  
(TeacherInDept CS223 CS).>>

\*\*\*\*\*

Predicate Calculus:

V crs, dept. (}[](TeacherInDept crs dept) & (Department crs CS))  
    ⊃ (NOT (TeacherInDept crs CS)) .

\*\*\*\*\*

Units:

CS223  
    Isa: CS-Course

CS-Course  
    TeacherInDept: x  
    DefaultCondition: (NOT (EQ x CS))

<<There are many other equally-terse solutions using units.>>

2. [15] AI Philosophy/Methods

A) [5] AI's Contribution to Psychology

AI research offers a language (information processing language of computer processes) and a set of concepts applicable to the study of human information processing. The most fundamental modeling assumption is that human thought can be reduced (in the scientific

usage of that term) to a set of elementary information processing operations. A **computer is appropriate** because: since a computer is a general symbol processing system, any symbol-system model that can be conceived can be given an operational instantiation and rigorous test. AI has also lead to rich source of metaphors for describing people.

B) [3] Psychology's Contribution to AI

Essentially no progress is attributable to such brain modeling. The only example that may be at all relevant is in vision, where much of the low-level processing (at the sensor level, or slightly above) parallels the human visual system.

NOTE: Many people answered a different question here -- addressing the issue of how people have been described, using the metaphor of computers. This was not what we asked about.

C) [4] Dendral's Contribution to AI

Its main contribution

(in addition to its value as an **existence** proof that expert systems were possible) was the discovery that what was primarily responsible for the high levels of competence in performance was the knowledge that the program contained rather than the power of its inference methods. DENDRAL had little to say to the science of human behavior other than the result mentioned in the last sentence (which in itself is an important result for psychologists and educational psychologists).

D) [3] Dendral's Design Goals

If DENDRAL had sought to be a contribution to psychology, then a much more careful method of studying exactly how each expert chemist solved each problem would have to have been employed; with careful methods of acquiring and analyzing data invented or adapted. Also, there would have to have been a detailed match of the behavior of program and human.

HARDWARE SOLUTIONS

Problem 1. (12 points.) (a) Encoding of a Trit.

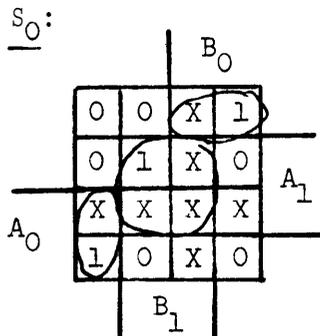
(2 pts)

Trit	D <sub>0</sub>	D <sub>1</sub>
-1	1	0
0	0	0
1	0	1

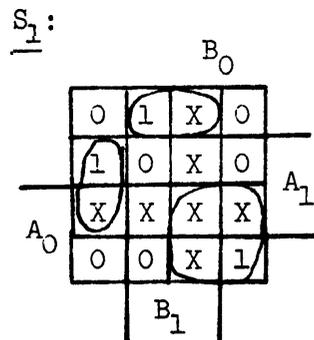
(5 pts) Addition Table

a	A <sub>0</sub>	A <sub>1</sub>	b	B <sub>0</sub>	B <sub>1</sub>	sum	S <sub>0</sub>	S <sub>1</sub>	carry	C <sub>0</sub>	C <sub>1</sub>
-1	1	0	-1	1	0	1	0	1	-1	1	0
-1	1	0	0	0	0	-1	1	0	0		00
-1	1	0	1	0	1	0		00	0		00
0	0	0	-1	1	0	-1	1	0	0		00
0	0	0	0	0	0	0		00	0		00
0	0	0	1	0	1	1	0	1	0		00
1	0	1	-1	1	0	0		00	0		00
1	0	1	0	0	0	1	0	1	0		00
1	0	1	1	0	1	-1	1	0	1	0	1
other						don't care					

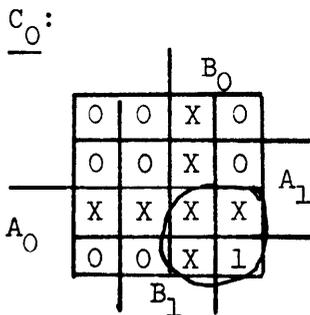
Karnaugh Maps



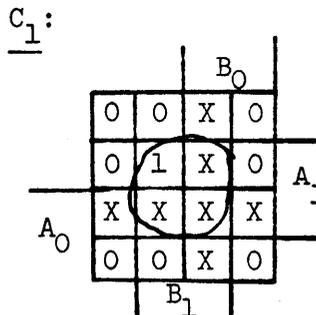
$$S_0 = A_1 B_1 + A_0 \bar{B}_0 \bar{B}_1 + \bar{A}_0 \bar{A}_1 B_0$$



$$S_1 = A_0 B_0 + \bar{A}_0 \bar{A}_1 B_1 + A_1 \bar{B}_0 \bar{B}_1$$

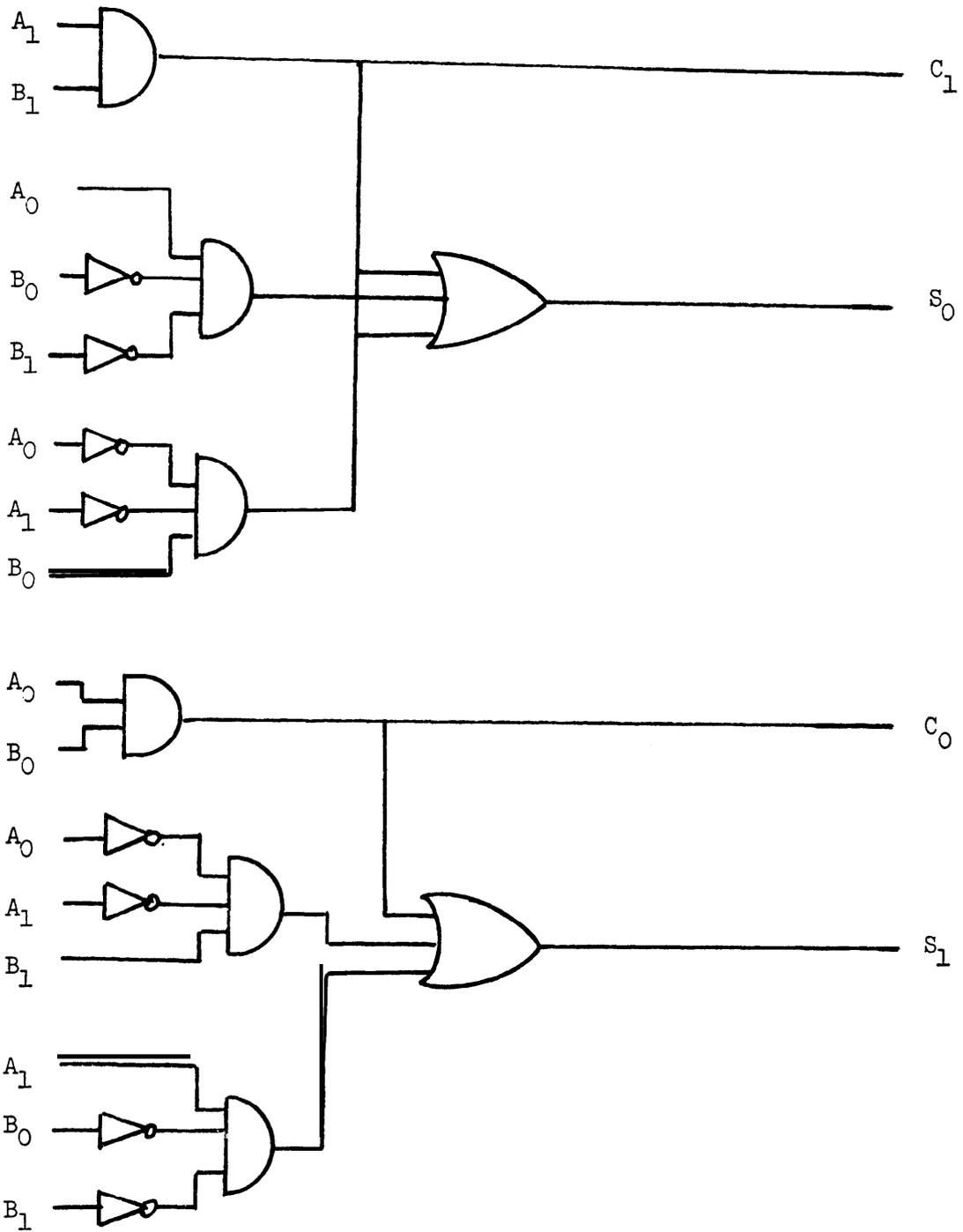


$$C_0 = A_0 B_0$$



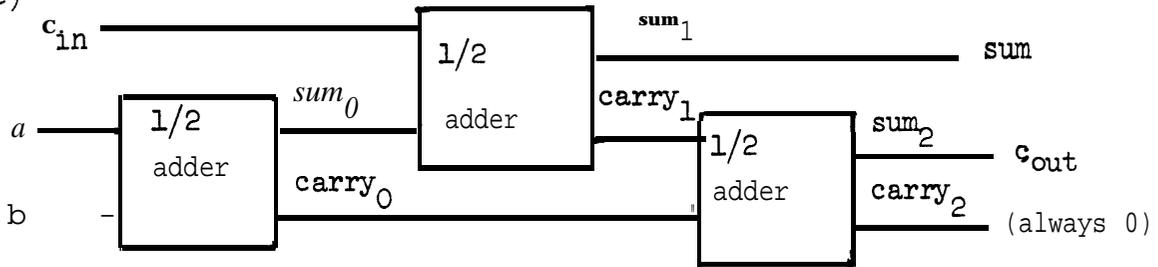
$$C_1 = A_1 B_1$$

(5 pts) Circuit (8 AND and OR gates)



(3 pts) (b) see circuit of (a)

(5 pts) (c)



Rationale:

$$\begin{aligned}
 (a + b) + c_{in} &= (s_0 + c_0) + c_{in} && \text{(1/2-adder)} \\
 &= (s_0 + c_{in}) + c_0 \\
 &= (s_1 + c_1) + c_0 \\
 &= s_1 + (c_1 + c_0) \\
 &= s_1 + s_2 + c_2
 \end{aligned}$$

If  $c_0 = 0$  then  $c_1 + c_0 = c_1 \therefore s_2 = c_1, c_2 = 0$

Only cases when  $c_0 \neq 0$  are:

(i)  $a = b = -1 \quad c_0 = -1 \quad s_0 = 1$

(ii)  $a = b = 1 \quad c_0 = 1 \quad s_0 = -1$

• If (i) holds, either  $c_1$  is 0 (hence  $c_2 = 0, s_2 = c_0$ ) or

$$c_{in} = 1 \text{ so } s_1 = -1 \quad c_1 = 1 \quad \therefore \quad c_0 + c_1 = 0 \quad \therefore \quad c_2 = 0$$

If (ii) holds, either  $c_1$  is 0 or

$$c_{in} = -1 \text{ so } s_1 = 1, \quad c_1 = -1 \quad \therefore \quad c_0 + c_1 = 0 \quad \therefore \quad c_2 = 0.$$

Problem 2. (10 points) (each part 2 pts)

- (a) 10
- (b) 4
- (c) 9
- (d) 11
- (e) 2

Problem 3. (20 points)

(a) (5 points) This code segments fetches the word at the address in A1 in the local memory, storing it in register D . It then puts the word on the bus. It continues to offer to word and assert slave sync (word ready) until it sees that master sync turns off.

(b) (15 points)

```

3 { /* get the word into register D */
    OEA1
    OEA1; Rd
    OEA1; Rd;
    OEA1; Rd; LDD1
}
4 { /* get control of the bus */
    Loop1: BR; JBG Got Bus
           BR; JA Loop1
    Got Bus SA; JBG Got Bus
}
5 { /* put request on bus until ack'ed */
    SA; BB; OED2; OEA2; CW;
    SA; BB; OED2; OEA2; CW;
    Loop2: SA; BB; OED2; OEA2; CW; MS; JSS Saw Cmd
           SA; BB; OED2; OEA2; CW; MS; JA Loop2
    Saw Cmd: SA; BB; OED2; OEA2; CS
}
3 { /* wait until slave is done, then release bus */
    Loop3: SA; BB; JSS Loop3
           (all inactive)
}

```

Problem 4. (10 points.)

(a) (3 points.)

$$2 \text{ rot} \cdot \frac{1 \text{ min}}{3600 \text{ rot}} \cdot \frac{60 \text{ sec}}{\text{min}} = \frac{1}{120} \text{ sec} = 8 \frac{1}{3} \text{ ms}$$

(b) (3 points.) read time = sector seek + rotational latency + scan of 1 sector

$$60 \text{ ms} + 8 \frac{1}{3} \text{ ms} + \frac{1 \text{ rot}}{48 \text{ sect}} \cdot \frac{1 \text{ min}}{3600 \text{ rot}} \cdot \frac{60 \text{ sec}}{\text{min}}$$

$$60 \text{ ms} + 8 \frac{1}{3} \text{ ms} + \underbrace{\frac{1}{48 \cdot 60} \text{ sec}}_{0.35 \text{ ms}}$$

68.7 ms

(c) (4 points.) 30%: 68.7 ms (as per (b))

70%: scan of 1 sector 0.35 ms  
 + rotational latency

([average of 2+ ... 49] x  $\frac{60}{3600}$  sec)

$$\frac{1}{48} \cdot \frac{51}{2} \cdot \frac{60}{3600} \quad \frac{8.85 \text{ ms}}{\hline} \quad \frac{9.20 \text{ ms}}{\hline}$$

$$(.3)(68.7) + (.7)(9.2) = 27.1 \text{ ms}$$

NUMERICAL ANALYSISSolutions

## 1) Iteration Methods

a)  $f(x) = x^n - a$

Newton's formula is  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

$$= x_n - \frac{x_n^k - a}{kx_n^{k-1}}$$

$$= x_n - \frac{1}{k} x_n + \frac{a}{kx_n^{k-1}}$$

$$= x_n \left(1 - \frac{1}{k}\right) + \frac{1}{k} \cdot \frac{a}{x_n^{k-1}}$$

b)  $g(x) = x \left(1 - \frac{1}{k}\right) + \frac{1}{k} \cdot \frac{a}{x^{k-1}}$

$$g'(x) = \frac{k-1}{k} - \frac{k-1}{k} \cdot \frac{1}{x^k}$$

$$= \frac{k-1}{k} \left(1 - \frac{1}{x^k}\right)$$

if  $0 < x_0 < \sqrt[k]{a}$  then  $x_1 - \sqrt[k]{a} = g(x_0) - g(\sqrt[k]{a})$

$$= g'(q) \left(x_0 - \sqrt[k]{a}\right) \text{ where } q \in \left(x_0, \sqrt[k]{a}\right)$$

and  $g'(q) = \frac{k-1}{k} \left(1 - \frac{1}{q^k}\right) < \frac{k-1}{k} \left(1 - \frac{1}{a}\right) = 0$

therefore  $x_1 - \sqrt[k]{a} > 0$  since

$$g'(q) < 0 \text{ and } x_0 - \sqrt[k]{a} < 0$$

$$\text{if } x_i > \sqrt[k]{a} \text{ then } x_{i+1} - \sqrt[k]{a} = g(x_i) - g(\sqrt[k]{a})$$

$$= g'(q) (x_i - \sqrt[k]{a}) \text{ where } q \in (\sqrt[k]{a}, x_i)$$

$$\text{and } g'(q) = \frac{k-1}{k} \left(1 - \frac{a}{q^k}\right) > \frac{k-1}{k} \left(1 - \frac{a}{a}\right) = 0$$

$$\text{therefore } x_i - \sqrt[k]{a} > 0 \text{ since}$$

$$g'(q) > 0 \text{ and } x_i - \sqrt[k]{a} > 0$$

$$\boxed{\text{so } x_i \geq \sqrt[k]{a} \text{ for all } i > 0}$$

$$\text{also for } x_i > \sqrt[k]{a} \text{ then}$$

$$\begin{aligned} |x_{i+1} - \sqrt[k]{a}| &= |g'(q)| |x_i - \sqrt[k]{a}| \text{ where } q \in (\sqrt[k]{a}, x_i) \\ &= \frac{k-1}{k} \left(1 - \frac{a}{q^k}\right) |x_i - \sqrt[k]{a}| \end{aligned}$$

$$\leq \frac{k-1}{k} |x_i - \sqrt[k]{a}|$$

$$\text{since } x_i \geq \sqrt[k]{a} \quad \forall \quad i \geq 1 \text{ then}$$

$$\boxed{|x_n - \sqrt[k]{a}| \leq \left(\frac{k-1}{k}\right)^{n-1} |x_1 - \sqrt[k]{a}|}$$

$$\text{so } x_i \xrightarrow{i \rightarrow \infty} \sqrt[k]{a} \text{ for all } x_0 > 0$$

c) 2 approaches:

$$1) f\left(\sqrt[k]{a}\right) = 0$$

$$f'\left(\sqrt[k]{a}\right) = k\left(\sqrt[k]{a}\right)^{n-1} \neq 0$$

therefore, by a theorem on Newton's method, the iteration converges quadratically locally.

$$\text{ii) } g\left(\sqrt[k]{a}\right) = \sqrt[k]{a} \left(1 - \frac{1}{k} + \frac{1}{k} \cdot \frac{a}{\left(\sqrt[k]{a}\right)^{n-1}}\right)$$

$$= \sqrt[k]{a} \left(1 - \frac{1}{k}\right) + \frac{1}{k} \sqrt[k]{a}$$

$$= \sqrt[k]{a}$$

$$g'\left(\sqrt[k]{a}\right) = \frac{k-1}{k} \left(1 - \frac{a}{a}\right)$$

$$= 0$$

$$g''\left(\sqrt[k]{a}\right) = (k-1) a^{1-\frac{k+1}{k}}$$

$$= (k-1) a^{-1/k} \neq 0$$

therefore, by a theorem in fixed point iteration, the iteration converges quadratically locally.

2) Interpolation

The necessary formulas for the error in cubic interpolation are found in Atkinson, pp. 132-134. Let  $x_0, x_1, x_2, x_3$  be equally spaced points with  $x_{i+1} - x_i = h$ . Then

$$\max_{x \in I} |f(x) - p_3(x)| \leq \frac{1}{4!} \max_{t \in I} |f^{(4)}(t)| \cdot \max_{t \in I} |\Psi_3(x)|$$

$$\text{where } \Psi_3(x) = (x-x_1)(x-x_2)(x-x_3)$$

$$\text{But } \max_{x_1 \leq x \leq x_2} |\Psi_3(x)| = \frac{9}{16}h^4$$

$$\text{and } \max_{x_0 \leq x \leq x_3} |\Psi_3(x)| = h^4$$

Hence

$$\max_{x_1 \leq x \leq x_2} |f(x) - p_3(x)| \leq \frac{3h^4}{128} \max_{x_0 \leq t \leq x_3} |f^{(4)}(t)| \quad (1)$$

$$\max_{x_0 \leq x \leq x_3} |f(x) - p_3(x)| < \frac{h^4}{24} \max_{x_0 \leq t \leq x_3} |f^{(4)}(t)| \quad (2)$$

a) Except for the intervals  $[1.0, 1.0+h]$  and  $[10.0-h, 10.0]$ , we can choose sample points to use (1). But since we do have to handle these intervals, we have to use (2). We want the estimate (2) to be less than  $5 \times 10^{-7}$ . Since  $f(x) = \log_{10} x = M \ln x$  where  $M = \log_{10} e \approx 0.4343$ , we have

$$f''(x) = -\frac{M}{x^2}, f'''(x) = \frac{2M}{x^3}, f^{(4)}(x) = -\frac{6M}{x^4}$$

Hence  $\max_{1 \leq t \leq 10} |f^{(4)}(t)| = \frac{6M}{1} = 6M$

So we need  $\frac{h^4}{24}(6M) \leq 5 \times 10^{-7}$

$$\text{or } h^4 \leq \frac{4 \times 5 \times 10^{-7}}{M} = 460.51 \times 10^{-8}$$

$$\text{or } h \leq 4.63 \times 10^{-2}$$

So we choose  $h = 0.04$ .

$$\text{Then } \frac{h^4}{24}(6M) \approx .000000278 < 3 \times 10^{-7}$$

But if  $h = 0.05$ ,  $\frac{h^4}{24}(6M) \approx .00000068 > 5 \times 10^{-7}$

b) With the two extra points  $1 + \frac{h}{2}$ ,  $1 + \frac{3h}{2}$ , we can use estimate (1) to get  $h$  because in first special interval  $h$  is halved.

$$\text{So now we need } \frac{3h^4}{128}(6M) \leq 5 \times 10^{-7}$$

$$\text{or } h^4 \leq \frac{64 \times 5 \times 10^{-7}}{9M} = 818.69 \times 10^{-8}$$

$$\text{or } h \leq 5.35 \times 10^{-2}$$

So we choose  $h = 0.05$ .

$$\text{Then } \frac{3h^4}{128} (6M) \approx .00000038 < 4 \times 10^{-7}$$

and in  $[1.0, 1.0+h]$ , we have extra point,  $1.0 + \frac{h}{2}$  so we use (2) and

$$\text{get errors } \leq \frac{\left(\frac{h}{2}\right)^4}{24} (6M) = .00000004 < 4 \times 10^{-8} < 10^{-7}$$

There is no problem near  $x = 10.0$  because

$$f^{(4)}(t) \sim \frac{6M}{10^4} \sim 3 \times 10^{-4} \text{ which is very small.}$$

3) Linear Algebra Quickies

$$i) \ a) \quad ||I|| = \max_{x \neq 0} \frac{||Ix||}{||x||} = \max_{x \neq 0} \frac{||x||}{||x||} = 1$$

$$b) \quad \frac{||Ax||}{||x||} \leq \max_{y \neq 0} \frac{||Ay||}{||y||} \equiv ||A|| \text{ if } ||x|| \neq 0$$

$$\text{therefore } ||Ax|| \leq ||A|| \cdot ||x||$$

$$\text{and } ||Ax|| = 0 = ||A|| \cdot ||x|| \quad \text{if } ||x|| = 0$$

$$\text{therefore } ||Ax|| \leq ||A|| \cdot ||x|| \quad \text{for all } x$$

$$c) \quad ||ABx|| \leq ||A|| \cdot ||Bx|| \leq ||A|| \cdot ||B|| \cdot ||x||$$

$$\text{so } \frac{||ABx||}{||x||} < ||A|| \cdot ||B|| \quad \text{for } x \neq 0$$

$$\text{therefore } ||AB|| = \max_{x \neq 0} \frac{||ABx||}{||x||} \leq ||A|| \cdot ||B||$$

$$d) \quad 1 = ||I|| = ||AA^{-1}|| \leq ||A|| \cdot ||A^{-1}|| = K(A)$$

ii) a) Assume that  $I+A$  is not invertible.

$$\text{then } x \neq 0 \text{ and } (I+A)x = 0$$

$$\text{then } Ix = -Ax$$

$$\text{so } ||x|| \leq ||-Ax|| = ||A|| \cdot ||x|| < ||x||$$

$$\text{since } ||A|| < 1$$

- contradiction

therefore  $I+A$  is invertible.

b)  $x = (I+A)^{-1}y$  and  $|| (I+A)^{-1}y || = || (I+A)^{-1} ||$

Therefore  $||x|| = || (I+A)^{-1} ||$

also  $(I+A)x = y$

and  $x = y - Ax$

$$||x|| \leq ||y|| + ||A|| \cdot ||x|| = 1 + ||A|| \cdot ||x||$$

so  $||x|| \leq \frac{1}{1-||A||}$

$$|| (I+A)^{-1} || \leq \frac{1}{1-||A||}$$

iii) a)  $(A+B) = A(I+A^{-1}B)$

and  $||A^{-1}B|| \leq ||A^{-1}|| \cdot ||B|| < 1$

so  $(I+A^{-1}B)^{-1}$  exists as does  $A^{-1}$

therefore  $(A+B)^{-1}$  exists

$$\begin{aligned} \text{b) } || (A+B)^{-1} || &= || (I+A^{-1}B)^{-1} A^{-1} || \\ &\leq || (I+A^{-1}B)^{-1} || \cdot ||A^{-1}|| \\ &\leq \frac{||A^{-1}||}{1-||A^{-1}B||} \end{aligned}$$

iv)  $Ax = b$  so  $||A|| \cdot ||x|| \geq ||b||$

and

$x = A^{-1}b$  so  $||x|| \leq ||A^{-1}|| \cdot ||b||$

giving  $\frac{||b||}{||A||} \leq ||x|| \leq ||A^{-1}|| \cdot ||b||$

v) a)  $b - A\bar{x} = \delta A\bar{x}$

$$\text{so } \|b - A\bar{x}\| < \|\delta A\| \cdot \|\bar{x}\| = \epsilon \|A\| \cdot \|\bar{x}\|$$

$$\text{b) } \|b - A\bar{x}\| \leq \epsilon \|A\| \frac{\|b\|}{\|A + \delta A\|}$$

$$< \epsilon \|A\| \cdot \|b\| \cdot \frac{1}{\|A\| - \epsilon \|A\|}$$

$$= \frac{\epsilon \|b\|}{1 - \epsilon}$$

$$\text{c) } \|b - A\bar{x}\| \leq \epsilon \|A\| \cdot \|(A + \delta A)^{-1}\| \cdot \|b\|$$

$$\leq \epsilon \|A\| \frac{\|A^{-1}\|}{1 - \|A^{-1}\delta A\|} \|b\|$$

$$\frac{\epsilon \|A\| \cdot \|A^{-1}\| \cdot \|b\|}{1 - \|A^{-1}\| \cdot \|A\|}$$

$$\frac{\epsilon K(A) \|b\|}{1 - \epsilon K(A)}$$

d) since  $K(A) > 1$ , the bound in b) is smaller.

$$\text{e) } \frac{\|b - A\bar{x}\|}{\|\bar{x}\|} < \epsilon \|A\| \quad \text{for all } \bar{x} \neq 0$$

SOFTWARE SYSTEMS  
SOLUTIONS TO THE COMPREHENSIVE

7po'ints #1

- (a) procedure & function names  
2 pts labels  
constants  
**types**  
variables  
formal parameters  
enumerated type value names  
record field names
- (b) for efficient manipulation, additional information  
on block number, lexical level, size of variables  
and types, etc is needed

for procedures/functions:

```

proc__entry = RECORD
    name: string;
    formal-list: Aformal_entry;
    locals: Avar_entry
    outer__proc: Aproc__entry;
    nested__proc: Aproc__entry;
    next,proc: Aproc__entry;
    return__type: Atype__entry;
    type-list: Atype__entry;
    const__list: Aconst__entry;
    label-list: Alabel__entry;
end;

```

for labels:

```

labels-entry = RECORD
    number: integer;
    next__label: Alabel__entry;
end;

```

for constants:

```

const__entry = RECORD
    name: string
    next__const: Aconst__entry;
    CASE kind: const__kind OF
        int: (int__value: integer);
        real-no: (real-value: real);
        character: (char-value: char);
        enumerated: (value: Aenum__entry);
    END;

```

[note: const,kind = (int, **real\_no**, character, enumerated);3

```

for types:
  type_kind = (array-type, enum_type, set_type,
              ptr_type, file-type, subrange_type,
              int_type, real_type, char-type,
              record-type, renaming);
  type-entry = RECORD
    name: string;
    next_type: ^type_entry;
    CASE kind: type_kind OF
      array-type: (index_list: ^index_entry;
                  elem_type: ^type_entry);
      enum_type: (count: integer;
                 first-val: ^enum_entry);
      set-type: (elem_type: ^type_entry);
      renaming: (original_type: ^type_entry);
      ptr_type: (ref_type: ^type_entry);
      file-type: (file, elem: ^type_entry);
      subrange-type: (base_type: ^type_entry;
                     first, elem,
                     last_elem: (*like const_entry*));
      record-type: (size: integer;
                   first-field: ^field_entry;
                   variant-part: ^variant_entry);
    END;

for variables:
  var_entry = RECORD
    name: string;
    type_def: ^type_entry;
    next_var: ^var_entry;
  END;

for formal parameters:
  parm_type = (byvalue, byref);
  formal-entry = RECORD
    name: string;
    kind: parm_type;
    type_def: ^type_entry;
    next-formal: ^formal_entry;
  END;

for enumerated type values:
  enum_entry = RECORD
    name: string;
    kind: ^type_entry;
    code-value: integer;
    prior,
    follower: ^enum_entry;
  END;

```

```

for record field names:
  field-entry = RECORD
    name: string;
    kind: ^type_entry;
    next-field: ^field_entry;
  END;

```

```

for variant of record:
  variant-entry = RECORD
    x tag_value: (*constant value*) \
    field_list: Afield-entry;
    next-tag: ^variant_entry;
  END;

```

```

for arrays:
  index-entry = RECORD
    index-type: type entry;
    next index: index entry;
  END;

```

(NOTE: Full credit was given for a good subset of this.)

12 points #2

(a) Differences between PASCAL and FORTRAN  
 -2 pts function/subroutine calls:

	<u>PASCAL</u>	<u>FORTRAN</u>
Recursive call allowed?	Yes	No
All procedures visible in any scope?	No	Yes
Static local variable space allocation?	No	Yes

-10 pts

(b) \*5pts i. One method would be to use 1 segment per subroutine, function, and common block. Local variables are stored in segment with code for subroutine/function. The symbols defined for the linker/loader would include:

- .name for segment
- .entry point for execution of code
- .any external names used

\*5pts ii. One method would include 1 segment per procedure and function. These hold code only. Defined symbols include:

- .name for segment
- .external symbols - names for procedures called, for stack segment

A stack/heap segment would also be defined: symbols for it would be the bottom of the stack and the beginning of the heap.

8 points #3  
(2 each part)

- (i) Gives poor treatment to jobs that block frequently (**eg, on I/O**). Also a few long, big jobs may hog the entire machine for a long time (if they are the ones that get to be the runnable jobs).
- (ii) Favors jobs that use little main memory, especially if they don't block often (eg, compute-bound). Long jobs especially if they block frequently, may suffer starvation.
- (iii) Favors jobs newly submitted, especially if they use little CPU time.
- (iv) Favors jobs which have used little CPU time since submission or last I/O operation: ie, little time use or I/O bound. **Long**, compute-bound jobs treated very unfavorably.

14 points #4

- (2 pts) a. "While" is used to ensure that only 1 of the group of awakened jobs will gain entry (ie, all will check the lock to see if someone else already got in).
- (4 pts) b. The root of the problem is that conceivably, 2 processes may see the lock open at the same time,  
(eg, A tests lock, is suspended,  
B tests lock, sets lock, enters,  
is suspended, A is resumed).  
On an uniprocessor machine, one solution is to do the scheduling such that a process is not suspended until it blocks (or completes). **On** a multiprocessor machine, it is necessary to synchronize which processors are reading/writing locks (so if lock **is** open, can close it before others read). A monitor-type solution will be needed.
- (4 pts) c. Synchronization control involves blocking some processes until conditions permit them to continue. The choice of processes to block and awaken may not coincide with the usual **schedulling** policy. (**Eg**, if A and B are blocked at entrance to a critical section: A low priority but blocked first, B higher priority. Synch control may release A first, then B, even though B has higher schedule priority.) The constructs suggested have the advantage that they let the schedule decide which of the newly unblocked processes should go first (ie, scheduling policy still has influence).. Semaphores usually treat blocked processes on a first in, first out basis.

- (4 pts) d. **"Destroy(process)"** may leave the lock closed, permanently blocking matters. If the lock is opened by the system, trouble can arise because the data structures may be in an inconsistent state. Semaphores don't handle the matter at all. Monitors have **enough** structure that the **system** could simulate an exit from the monitor (at the risk of using corrupted data structures).

6 points #5  
(3 pts each part)

- (a) Yes, a set of processes can block:  
 process A does Receive  
 process B does Receive  
 (both expect a message from the other)
- (b) Suppose only 5 buffers are available. Consider the following set of processes:  
 A: reads a line of input, sends the line to B.  
 (indefinite repetition)  
 B: accepts lines of input until a full command has been read, parses command, sends it to C. Then goes back to getting input.  
 c: executes commands

Events: B starts parsing a command  
 A sends 5 lines of an 8 line command  
 and blocks on send  
 B finishes parsing and blocks trying to  
 send the command to C

(If ~~6~~ buffers, B won't block and A won't block.)

13 points #6

- (2pts) a. Consider sentence wuwvw. Two derivations are:

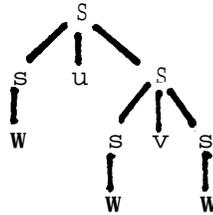


- (4pts) b.  $S \Rightarrow wu \ S \ | \ wv \ S \ | \ w$

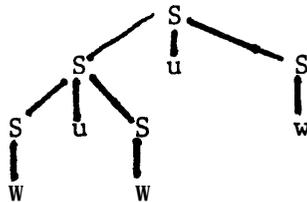
- (1pt) c. For no k: ambiguous grammars are never LR(k).

SOFTWARE (cont)

- (3pts) d. By assigning precedence to the productions  $S \rightarrow SuS$  and  $S \rightarrow SvS$  (so that they have different precedence), one can resolve how to parse statements of the form  $SuSvS$  (as  $Su(SvS)$  or  $(SuS)vS$ ). If  $v$  has higher precedence (in the same sense as  $* \gg +$  :ie,  $v$  binds more tightly)  $wuwvw$  has 1 derivation tree:



- (3pts) e. Ambiguities still arise with regard to parsing sentences involving only the same operator: eg,  $wuwuw$  : is this  $(wuw)uw$  or  $wu(wuw)$ ? Associativity (right to left, left to right, or non-associative) clarifies which. If associate left to right, the only derivation for  $wuwuw$  is





4: If  $L = \{0^n 1^n \mid n \geq 0\}$ , then  $\frac{dL}{d0} = \{0^{n-1} 1^n \mid n \geq 0\}$  and  $\frac{dL}{d1}$  is empty.

To show  $\frac{dL}{da}$  is regular when  $L$  is, let  $M = (Q, \Sigma, \delta, q_0, F)$  be a deterministic finite automaton accepting  $L$ . Then  $M' = (Q, \Sigma, \delta, \delta(q_0, a), F)$  accepts  $\frac{dL}{da}$ . That is,  $M'$  is  $M$  with a different start state, the state that  $M$  gets to from its start state on input  $a$ . In proof,  $\delta(q_0, aw) = \delta(\delta(q_0, a), w)$ , so  $M'$  accepts  $w$  if and only if  $M$  accepts  $aw$ ; that is,  $M'$  accepts  $\frac{dL}{da}$ .

For the context-free part, let  $L - \{\epsilon\}$  be generated by a Greibach Normal Form grammar

$$G = (V, T, P, S)$$

Let  $S'$  be a new variable, and  $P' = P \cup \{S' \rightarrow \alpha \mid S \rightarrow a\alpha \text{ is in } P\}$ . Let  $G' = (V \cup \{S'\}, T, P', S')$ . It is easy to show that each variable in  $V$  generates in  $G'$  exactly what it generates in  $G$ , and  $S'$  generates  $w$  if and only if  $S$  generates  $aw$ . Thus  $L(G') = \frac{dL}{da}$ .

5: To reduce  $P$  to  $Q$ , let  $M_i$  be a Turing machine. Construct another Turing machine  $M_j$  that does the following.

1. Delete the first symbol from the input (if the input is  $\epsilon$ , halt).
2. Simulate  $M_i$  on the remaining input, accepting if  $M_i$  accepts.  
If  $M_i$  accepts some input, say  $w$ , then  $M_j$  will accept  $0w$  and  $1w$ . If  $M_i$  accepts no input, neither will  $M_j$ . Thus, the function  $f(i) = j$  maps Turing machines accepting one or more inputs into machines accepting two or more, as desired.

For the opposite reduction, let  $M_i$  again be a Turing machine. We construct a nondeterministic, two-tape Turing machine  $M$  that works as follows.

1. Guess an input  $x$  on the second tape. Check that  $x$  is different from  $w$ , the input to  $M$  appearing on the first tape. If not, halt (in this nondeterministic branch).
2. Simulate  $M_i$  on  $w$ , using the first tape.
3. If  $M_i$  accepts  $w$ , simulate  $M_i$  on  $x$ , using the second tape.
4. If  $M_i$  accepts  $x$  too, then  $M$  accepts its input,  $w$ .

Now, convert  $M$  to a one-tape, deterministic Turing machine, using standard constructions, as in Chapter 7 of HU. Let the result be  $M_j$ . Then if  $M_i$  accepts two or more strings,  $M$ , given one of them, say  $w$ , will "guess" another,  $x$ , and accept  $w$ , so  $M_j$  will also accept  $w$ . However, if  $M_i$  accepts no string, or only one string, then  $M$  and  $M_j$  will never accept their input, because they cannot guess a different string  $x$ , that is also accepted by  $M_i$ . Thus the mapping  $f(i) = j$  is a reduction from  $Q$  to  $P$ .

...

**Computer Science Comprehensive Exam**

Winter 1983 (January 8-9, 1983)

Problem 1. [10 points]

- (a) Give the name and a short formal definition of three graph-theoretic problems other than the clique problem which are W-complete,
- (b) Give the name and a short formal definition of two NP-complete problems for sets of integers.

Problem 2. [5 points]

The external path length of a tree is the sum over **all** leaves of **the** lengths of the paths from the root to each leaf. What is the external path length of a complete binary tree with  $n$  levels?

Problem 3. [15 points]

Let  $G = (V, E)$  be a digraph, and  $u \in V$  some vertex such that every other vertex in  $V$  is reachable via a (directed) path from  $u$ . The following is a high-level description of a class of algorithms to number the vertices of  $G$  from 1 through  $|V|$ . Initially, **all** vertices are unnumbered.

```

i := 1; number[u] := 1;
while there is an unnumbered vertex in G do
begin
  i := i+1;
  v := an unnumbered node satisfying property P;
  number[v] := i
end;

```

Specify (at most two lines each)  $P$  such that the numbering produced by the algorithm gives

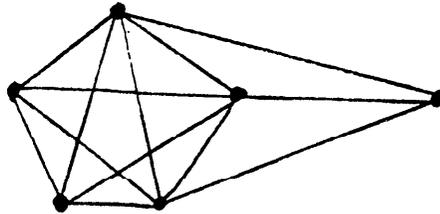
- (a) a depth-first numbering;
- (b) a breadth-first numbering;
- (c) node numberings in topological order.

State, where necessary, any additional condition which  $G$  has to satisfy for a corresponding numbering to exist. Note: Each property  $P$  should use only the unnumbered vertices, the numberings of the numbered vertices, and the edges, and should use no additional data structures.

Winter 1983

Problem 4. [15 points]

Let  $G = (V, E)$  be a (**general** undirected) graph. A clique of size  $k > 1$  of  $G$  is a subset  $V' \subset V$  with  $|V'| = k$  such that every two distinct vertices in  $V'$  are connected by an edge in  $E$ . The following 6-vertex graph contains a clique of size 5 :



- (a) Give a high-level description of a simple polynomial-time algorithm to determine if a planar graph  $G$  has a clique of size  $k$  or more, where  $(G, k)$  are the input data. Use the fact that no planar graph contains a clique of size 5 .
- (b) Briefly **state** why your algorithm is correct, and give an estimate of its running time in terms of  $|V|$  and  $k$  .

Problem 5. [15 points]

Suppose  $K$  is a totally ordered universe, and suppose you had a function middle $(k_1, k_2, k_3)$  which, when given any three keys  $k_1, k_2, k_3 \in K$ , produces the value 1, 2, or 3 depending on whether  $k_1, k_2$ , or  $k_3$  is the middle value of the keys (ties are broken arbitrarily). Also assume that  $K$  has a minimum element 0, We are interested in the complexity of sorting  $n$  different keys (all  $> 0$ ), using the function middle as the only comparison primitive.

- (a) Prove a worst-case lower bound of  $n \log_3 n +$  lower order terms (which needn't be given) for the number of calls of middle in order to sort  $n$  keys.
- (t) With insertion sort,  $n$  keys can be sorted using  $n \log_2 n +$  lower order terms (ordinary) comparisons (but possibly more time to move data around). Give a high-level description of a sorting algorithm using middle as the only comparison primitive such that its worst-case complexity in terms of **calls** of middle is  $n \log_3 n +$  lower order terms.

Winter 1983

These questions lie within the core of fundamentals of the AI part of CS, The answers should not be lengthy but they should penetrate to the basics.

Problem 1. [10 points]

- (a) What is a PROBLEM SPACE?
- (b) Describe the problem space for the problem of:
- (b1) making a move in chess
  - (b2) proving a theorem
- (If you use diagrams, be sure to use a lot of words on the diagram, so that we can understand what you are trying to say.)

Problem 2. [10 points]

"AI" uses heuristic methods to control combinatorial explosion,"

- (a) What is the so-called combinatorial explosion? (Answer by using an illustrative example.)
- (b) What is a heuristic method and/or heuristic'? (Answer with short discussion plus illustrative example.)

Problem 3. [13 points] KNOWLEDGE REPRESENTATION:

- (a) What is the "knowledge representation problem" of AI?
- (b) There are half a dozen (more or less) general classes of representational formalisms (or quasi-formalisms) that AI uses.
- What are they (name five)? (Answer with a very short description or illustrative example.)

Problem 4. [12 points]

AI programs use inference methods for finding lines-of-reasoning leading to solutions to problems (for purposes of this question you can consider "problem solving strategy" to be equivalent to "inference method"). There are half a dozen (more or less) general classes of such methods that AI uses. Name five such methods, with a sentence or two about each (how it works). You may not know the "official" name of the method, but you'll still get credit if your short description is clear and accurate.

Problem 5. [5 points]

The understanding of natural language by programs is one of AI's most intractable problems. There is now almost universal agreement among AI scientists as to the reason. What is it?

Problem 6. [10 points]

Some have said that AI ideas have had a revolutionary impact upon Cognitive Psychology. It seems paradoxical. AI is concerned with "machine intelligence" while Cognitive Psychology is concerned with human thought. Describe the methodology by which a Cognitive Psychologist makes use of "AI ideas" in advancing his/her science, Be brief (one paragraph should do it). (Hint: concentrate on the technical method by which psychologists take advantage of AI's computer science ideas.)

Winter 1983

In this part you will almost certainly improve your score by checking your working carefully!

· Problem 1. (10 points)

We often model a computer system as three boxes:



Suppose these units have the following long term average characteristics:

CPU      300ns per instruction. This time is NOT overlapped in any way with memory cycles. Assume 1.75 memory references per instruction average.

Cache    100ns access and cycle time for a "hit".  
 100ns access and cycle time for a "miss".  
 90% hit rate.

Memory   1000ns access and cycle time, not overlapped with cache,

(A) [5 points]

Assume there are no differences between memory cycles (that is, read cycles and write cycles take the same time). What is the average instruction rate of this system?

(B) [5 points]

Assume that  $1/3$  of all memory references are writes and that a write will cycle the cache and always miss the cache. What is the average instruction rate of this system?

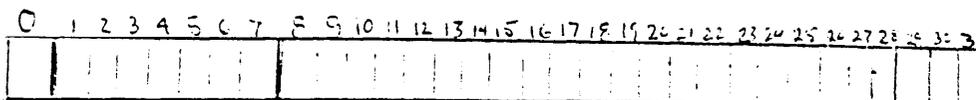
Winter 1983

Problem 2 (25 points)

These questions deal with Floating Point Arithmetic in Computers.

- A (2) What does "normalized" mean? That is, what characterizes a normalized number?
- B (2) Why is normalization necessary and/or desirable?
- C (1) Some computers use a "hidden bit" in their representation of a floating point number. What is a "hidden bit"?

The remaining questions in this section deal with the floating point format found in the IBM 360/370 and similar machines. A single-precision floating point number appears in 32 bits as follows



Fraction (Also called mantissa). The "hexadecimal point" appears to the left of bit 8.

An exponent of 16, in excess-64 notation.

Sign bit. 0 for positive; 1 for negative. This bit specifies the sign of the fraction. Negative numbers are in sign-magnitude notation.

Results and operands are always normalized. Zero is represented by 32 bits of zero.

- D (5) What is the smallest positive number representable in this format?
- E (5) What is the largest positive number representable?

In the 370, machine words are customarily written in hexadecimal notation. In hexadecimal notation, one character is written to represent 4 bits. 4 bits may take any of 16 possible values; in hexadecimal these sixteen values are written as the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

- F (5) Perform the following floating point addition. 41800000  
Assume truncation, not rounding + 42F90000
- G (5) Perform the following floating point addition. 451A6F3C  
Assume truncation, not rounding + C132185D

Winter 1983

Problem 3 (25 points)

In this problem you will be asked to design a clocked, finite-state automaton that will divide by three. A binary number is presented to the machine in a bit serial format with the most significant bit being presented first. The desired output is a series of bits representing the integer portion of the quotient of the input number divided by three.

You may assume the availability of

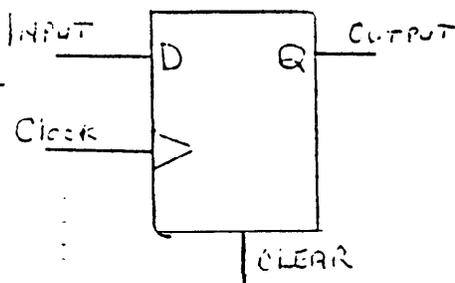
- the input data signal
- a clock that "ticks" an appropriate time after the input has become valid, and
- a reset signal which is asserted prior to any data being presented, and which should bring the machine to a known initial state.

A (3 points) In binary notation, show the long division of decimal 77 by decimal 3

B (10 points) Develop the state transition diagram for the automaton that divides by three. You should specify the behavior of the machine (that is, the new state and the output) for each state and condition of input.

C (10 points) Implement your state transition diagram using only multiple input NAND gates and type D, Clocked flip-flops with asynchronous clear. Provide a clearly labelled output signal and all gates and flip-flops needed to implement the prescribed machine.

A type D, Clocked flip-flop with a synchronous clear is depicted below:



While clear is asserted, Q is held at zero. In normal operation, the input at D is set up before the clock occurs. After the clock occurrence, the value that was present at the input D prior to the clock will appear at Q as output.

D (2 points) The combinational logic that you implemented with NAND gates in part C might, in more modern designs, be implemented in a rather different way. Briefly describe one such alternative implementation.

Winter 1983

Problem 1. [20 points]

(a) Suppose  $P(x)$  is a polynomial with real coefficients which has distinct real roots.

[5 points] Compare the rates of convergence of Newton's method, the secant method and the bisection method.

[5 points] State a possible disadvantage of each method.

(b) Suppose now that  $P(x)$  has a double root at  $x_0$ .

[5 points] How does this fact affect the performance of each of these methods in determining  $x_0$ .

[5 points] State how Newton's method can be modified to **retain** the same convergence properties near  $x_0$  as in the case when  $x_0$  is a distinct root.

Problem 2. [20 points]

(a) [4 points]

If the integration formula  $\sum_{i=0}^n w_i f(x_i)$  is exact for polynomials  $f$  of degree  $\leq n$ ,  $n \geq 1$ , in evaluating  $\int_a^b f(x)g(x) dx$  for fixed  $g(x) \geq 0$ , and  $\sum_{i=0}^m v_i f(x_i)$  is exact for polynomials of degree  $< m$ ,  $m > 1$ , show that

$$\sum_{i=0}^n w_i = \sum_{i=0}^m v_i .$$

(b) [8 points]

If  $I_n$  is the approximation to  $I = \int_a^b f(x) dx$  using the trapezoid rule with  $n$  equally spaced mesh points and  $f(x)$  has  $2r+2$  continuous derivatives then

$$I_n - I = \frac{h^2}{12} [f'(b) - f'(a)] - \frac{h^4}{720} [f'''(b) - f'''(a)] + \frac{h^6}{30,240} [f^{(5)}(b) - f^{(5)}(a)] + \dots + b_{2r} h^{2r} [f^{(2r-1)}(b) - f^{(2r-1)}(a)] = O(h^{2r+2})$$

Here  $h = (b-a)/n$ , and  $b_{2r}$  is a fixed constant. Using this fact

Winter 1983

show which values of the weights  $C_1$  and  $C_2$  will make

$$I^* = \frac{C_1 I_n + C_2 I_{2n}}{(C_1 + C_2)} \quad \text{a fourth order accurate approximation to } I .$$

(c) [8 points]

Using  $I_n$ ,  $I_{2n}$ , and  $I_{4n}$  derive a formula which is sixth order accurate.

Problem 3. [20 points]

Suppose we are solving the linear system of equations

$$Ax = b$$

where  $A$  is an invertible matrix, and we have an approximate solution  $\tilde{x}$  such that

$$A\tilde{x} = b + \tilde{r} .$$

The vector  $\tilde{r}$  is called the residual vector.

(a) [15 points]

Show that a large residual implies a large relative error in  $x$  by showing

$$\frac{\|x - \tilde{x}\|}{\|x\|} \geq \frac{\|\tilde{r}\|}{\|A\| \|A^{-1}\| \|b\|}$$

(b) [5 points]

Does a "small" residual vector (that is,  $\|b - A\tilde{x}\|$  is small) guarantee

that  $\tilde{x}$  is an accurate solution (that is,  $\|\tilde{x} - A^{-1}b\|$  is small)?

Why or why not?

Winter 1983

## SOFTWARE

- (2) 1. What is the basic difference between a process and a program, if any?
- (14) 2. Modify the following two Pascal procedures so that they are correctly synchronized when called by arbitrarily many processes running concurrently.

```

procedure QueueId(id:integer);
begin
    free:= free -1;
    next:= next +1;
    if next = 51 then next:= 1;
    Queue[next]:= id;
    used:= used +1;
end

```

```

procedure DequeueId(var id:integer);
begin
    used:= used -1;
    id:= Queue[last];
    last:= last +1;
    if last = 51 then last:= 1;
    free:= free +1;
end

```

QueueId queues the specified identifier, suspending the call if necessary until a queue space is available. DequeueId returns the first id in the queue, suspending the call if necessary until the queue is non-empty. There are at most 50 integer storage units for queued identifiers.

You are to use Wait(event) and Signal(event) for synchronization and disable and enable for indivisibility (if necessary).

Wait(event) - blocks calling process until another process calls Signal(event). This has the side-effect of enabling interrupts.

Signal(event) - unblocks all processes waiting for the specified event. This has no effect if no process is waiting on the event.

Enable/Disable - enable and disable all processor interrupts respectively.

(2)

Winter 1983

- (6) 3. Discriminate between deadlock prevention, avoidance and detection, and describe how each approach incurs a significant cost in dealing with deadlock.
- (8) 4. In a demand paging system,
- a) State the optimal page replacement policy.
  - b) State why the optimal is impractical and give a commonly used page replacement policy.
  - c) Define "thrashing" in the context of demand paging.
  - d) Describe program behavior that would cause your policy in (b) to behave worse than random page replacement, and thus aggravate a thrashing problem.
- (6) 5. Parameter Passing

Consider the program fragment:

```

var
  A: array[1..5] of integer;
  I: integer;

procedure SWAP (X, Y : integer);

  var
    T : integer;

  begin
    T := X;
    X := Y;
    Y := T;

  end;

begin
  A [1] := 9;
  A [2] := 7;
  A [3] := 5;
  A [4] := 3;
  A [5] := 1;

  I := 5;

  SWAP (I, A[I]);

  WriteLn (I, A[1], A[2], A[3], A[4], A[5]);

end

```

(3)

Winter 1983

What will be printed out if the parameters to SWAP are passed

- a) by value?
- b) by reference?
- c) by name?

- (8) 6. a) Do top-down parsers usually produce a leftmost or a rightmost derivation? Why?
- b) Do bottom-up parsers usually produce a leftmost or a rightmost derivation? Why?

- (8) 7. Consider the following grammar fragment:

```
<statement> ::= <if statement> |
               <while statement> |
               <compound statement> |
               <assignment statement>
```

```
<if statement> ::= IF <condition> THEN <statement> |
                  IF <condition> THEN <statement> ELSE <statement>
```

```
<compound statement> ::= BEGIN <statement list> END
```

```
<statement list> :: <statement list> ; <statement> |
                   <statement>
```

```
·
·
```

(productions for <while statement> , <assignment statement> , <condition>...)

- a) What is a problem with this grammar that will be present with any method used to parse it? How can this problem be solved? (one means of solving it is sufficient).
  - b) List three things that make this grammar unsuitable for use by a recursive descent parser. How could the grammar be changed to solve these problems without changing the language generated by the grammar?
- (8) 8 a) Suppose you were writing a procedure that needed to retain some state between invocations. How would this best be done for a procedure written in
- i) FORTRAN?
  - ii) ALGOL?
  - iii) Pascal?

Winter 1983

(4)

- b) One issue a language implementor must deal with is the allocation of storage for the data a program uses. Why is this allocation more difficult in ALGOL than in FORTRAN? Why is it yet more difficult in Pascal?

Winter 1983

Problem 1. [10 points]

Assume  $\{f_1, f_2, \dots\}$  is a total recursive listing of some of the total recursive functions (i.e., the function  $F(n, m) = f_n(m)$  is **total** recursive). Show that there is a total recursive function  $g$  not occurring in this list.

Problem 2. [10 points]

For each of the following formulae, give a proof of it in **your** favorite system of predicate logic or else give a model in which it is **false**:

$$(a) \exists y. \forall x. B(x, y) \supset \forall x. \exists y. B(x, y)$$

$$(b) \forall x. \exists y. B(x, y) \supset \exists y. \forall x. B(x, y)$$

In proofs, indicate the **axioms** and inference rules from your "favorite system" that are used.

Problem 3. [20 points]

Define recursive functions `reverse` and `append` on lists as follows:

```
reverse(u) = if u = nil then nil
             else append(reverse(cdr u), cons(car u, nil))

append(u, v) = if u = nil then v
               else cons(car u, append((cdr u), v)) .
```

It follows from these definitions that the reverse of any list is a list and the append of any two lists is a list. One can also prove the following set of facts about `append` and `reverse` :

```
nilR = nil
(a.u)R = uR * (a.nil)
nil * v = v
(a.u) * v = a.(u * v)
u * (v * w) = (u * v) * w
```

Here we have switched to using different notation for `cons` , `append` and `reverse` :

```
a.v = cons(a, v)
uR = reverse(u)
u * v = append(u, v)
```

which you may wish to use in your answer.

Winter 1983

Using these facts, prove:

(a) For **all** lists  $u$ :  $\text{append}(u, \text{nil}) = u$

(b) **Using** (a) and the facts cited above, if **you wish**, (you don't have to prove (a) to get full credit for this part), prove that for all lists  $u, v$ :

$$\text{reverse}(\text{append}(u, v)) = \text{append}(\text{reverse}(v), \text{reverse}(u))$$

Problem 4. [20 points]

For each of the languages below, indicate whether it is (i) a regular set, (ii) context-free but not a regular set, or (iii) not context-free.

Prove your answer in each case.

(a)  $\{0^n 1^i 2^n \mid i \geq 1, n \geq 1\}$

(b)  $\{0^i 1^j 2^k \mid i \geq j \geq k\}$

(c)  $\{0^i 1^j \mid \text{exactly one of } i \text{ and } j \text{ is odd}\}$

⋮

Winter 1983 - Analysis of Algorithms (Solutions)

Problem 1.

- (a) (a1) vertex cover: given a graph  $G = (V, E)$  and some number  $k \in \mathbb{N}$ , is there a  $V' \subseteq V$  with  $|V'| \leq k$  such that every edge in  $E$  is incident to at least one elt. of  $V'$ ?
- (a2) independent set: give a graph  $G = (V, E)$  and some number  $k \in \mathbb{N}$ , is there a  $V' \subseteq V$  with  $|V'| \geq k$  such that no pair of distinct vertices in  $V'$  is connected by an edge in  $E$ ?
- (a3) Hamiltonian path: given a graph  $G = (V, E)$  does  $G$  contain a simple (= no node repetition) path of length  $|V| - 1$ ?
- (b) (b1) partition: given  $a_1, \dots, a_n \in \mathbb{N}$ , is there a subset  $I \subseteq \{1, \dots, n\}$  such that

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i \quad ?$$

- (b2) 3-partition: given a  $b \in \mathbb{N}$  and  $3n$  numbers  $a_1, \dots, a_{3n} \in \mathbb{N}$ , all strictly between  $b/4$  and  $b/2$ , can the  $3n$  numbers be partitioned such that for each class the sum of its elements is  $b$ ?

Problem 2.

A complete binary tree has  $2^{i-1}$  nodes on level  $i$  (root at level 1). Hence there are  $2^{n-1}$  leaves, each reachable from the root by a path of length  $n-1$ . The external path length hence is

$$(n-1)2^{n-1} . \quad \dots$$

Problem 3.

- (a) depth-first:  $v :=$  some unnumbered vertex reachable by an edge from a numbered vertex whose number is as high as possible;
- (b) breadth-first:  $v :=$  some unnumbered vertex reachable by an edge from a numbered vertex whose number is as low as possible;
- (c) For a topological numbering to exist  $G$  must not contain any directed cycles.
- topological numbering:  $v :=$  some unnumbered vertex all whose predecessors are already numbered.

Winter 1983 - Analysis of Algorithms (Solutions)

Problem 4.

(a) case k of

$k = 1$  : if  $|V| > 0$  then return Yes else return No;

$k = 2$  : if  $|E| > 0$  then return Yes else return No;

$k = 3$  : for all  $\{v_1, v_2\} \in E$  do  
     for all  $v_3 \in V - \{v_1, v_2\}$  do  
         if  $\{v_1, v_3\} \in E$  and  $\{v_2, v_3\} \in E$  then  
             return Yes;

return No;

$k = 4$  : for all  $\{v_1, v_2, v_3, v_4\} \subseteq V$  do  
     if  $\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\} \in E$  then  
         return Yes;

return No;

$k \geq 5$  : return No;

(b) Any k-clique for  $k \geq 5$  contains a 5-clique. Therefore, as given by the hint, no planar graph can contain a k-clique for  $k > 4$ .

The worst-case running time of the above algorithm is dominated by the case  $k = 4$ . If we assume that E is given as an adjacency matrix, the if statement uses time  $O(1)$ , and the overall time **complexity** is  $O(|V|^4)$ .

Problem 5.

(a) There are six possible total **orderings** for three different keys  $k_1, k_2, k_3$ .

If we know the middle **element**, say  $k_2$ , two total orderings are compatible:  $k_1 < k_2 < k_3$  and  $k_3 < k_2 < k_1$ . Hence, one **call** of middle provides  $\log_2 3$  bits of information. To **determine** the total order of n keys, at least  $\log_2 n!$  bits of information **are** needed.

Thus, there has to be

$$\geq \frac{\log_2 n!}{\log_2 3} = \log_3 n! = n \log_3 n + \text{lower order terms}$$

calls of middle.

(b) (b1) Two keys  $k_1, k_2$  can be compared using one call of middle as follows: middle(0,  $k_1, k_2$ ), if the result is 2 the  $k_1 < k_2$ , else the result must be 3, and we have  $k_2 < k_1$ .

(b2). To sort  $< 3$  keys use a binary sort with **comparisons** implemented as above.

Winter 1983 - Analysis of Algorithms (Solutions)

(b3) To sort  $n > 3$  keys  $k_1, \dots, k_n$ , do an insertion sort as follows:

```

sort  $k_1, k_2, k_3$ ;
for  $i = 4, \dots, n$  do
    insert  $k_i$  in the sorted sequence of  $k_1, \dots, k_{i-1}$ ;
    let  $k', k''$  be the  $\lfloor (i-1)/3 \rfloor$  resp.  $\lfloor (2(i-1))/3 \rfloor$ 
        largest key of  $k_1, \dots, k_{i-1}$ ;
    case middle( $k', k_i, k''$ ) of:
1: co  $k_i < k' < k''$  oc
    recursively insert  $k_i$  in the lower third of the
        sorted sequence of  $k_1, \dots, k_{i-1}$ .
2: co  $k' < k_i < k''$  oc
    recursively insert  $k_i$  in the middle third.
3: co  $k' < k'' < k_i$  oc
    recursively insert  $k_i$  in the upper third.

```

To insert  $k_i$  into an ordered sequence of  $< 2$  keys we simulate binary comparisons.

As in every level of the recursion the length of the subsequence into which we have to insert  $k_i$ , is divided by 3 there are about  $\log_3 i$  levels of recursion (and hence calls to middle) to insert  $k_i$ . This yields a total of

$$\begin{aligned}
 & \sum_{i=4}^n \log_3 i + \text{lower order terms} \\
 &= \log_3 n! + \text{lower order terms} \\
 &= n \log_3 n + \text{lower order terms}
 \end{aligned}$$

calls of middle.

Winter 1983 - Artificial Intelligence (Solutions)

AI Questions: These answers are intended to be exemplary, not mandatory. Reasonable **facsimilies** and elaboradons may be perfectly acceptable.

1. A PROBLEM SPACE is the set of all possible paths that could lead to solutions of a problem (paths that are legal, of course, in terms of the vocabulary and task structure).

For the problem of making a move in chess, the **problem** space is the space of all legal moves that can be made plus all of their legal continuations, plus all of the legal continuations of those, etc., until the natural **termini** (win, lose, draw) of that tree are reached.

For **proving** a theorem, the problem space is the space of all one-step subproblem reductions from the theorem to be provided, and all the one-step reductions of those, etc., until the natural termini are reached.

2. A. "Combinatorial explosion": problem spaces generally have a **combinatorial** nature; hence, the number of paths that might conceivable need to be explored to find a solution goes up exponentially with problem difficulty (e.g. a "hard" proof might be many orders of magnitude more difficult to find than a "simple" proof; more formally, average-branching-factor raised to the power of **average-number-of-levels** that need to be searched to find a solution).

B. "Heuristic" or "heuristic method": knowledge of the task domain or a solution-finding procedure that reduces the amount of search that must be done in combinatorially explosive problem spaces (albiet in a way that is "plausible" rather than **rigorous** or guaranteed). Illustrative examples:

In chess: "in the middle game, it is good to move rooks to open files"

In problem solving: order your consideration of next-paths-to-be-generated on the basis of some evaluation function (best-first search).

3. Knowledge representation: designing and choosing formalisms in which knowledge can be expressed; and designing and choosing data structures that allow the expressions to be accessed and manipulated efficiently by inference processes.

Some knowledge representations:

- A. Logical formulae, e.g. expressions in first order predicate calculus.
- B. Production Rules, of the form IF (condition) THEN (action or consequence), as commonly used in expert systems work.
- C. Objects, collections of descriptive information about some entity in, **say**, attribute-value form. Examples: frames in FRL, UNITS, or objects in **SMALLTALK**.
- D. Semantic nets, networks of linked objects where the links are labeled by relations between pairs of objects.
- E. Procedural representations, or programs whose execution is the **expression of** the knowledge (sometimes referred to as "knowing how").
- F. **Semantic primitives**, e.g. Schank's "conceptual dependency" formulation.
- G. Scripts, or abstractions of a sequence of events; also known as skeletal plans.
- H. **Straightforward** list-structures or graph structures, e.g. chemical molecules represented as atom-bond graphs.

4. Inference methods/problem solving strategies:
  - A. Theorem proving. State the goal as a theorem to be proved and use some method (e.g. resolution) to prove it, thereby discovering the sequence for achieving the goal.
  - B. Working backwards, start from goal and do successive **subproblem** reductions using knowledge of the problem, until "**knowns**" terminate the search.
  - C. Workings forwards, explore from current (or initial state), seeking some state that satisfies a set of criteria for a solution. Finding a good move from some position in a chess game is usually done by working forwards.
  - D. Incremental, opportunistic bidirectional search, as in the blackboard procedure. Using knowledge to make applicable steps, either goal-driven (working backwards, top-down) or data-driven (working forwards, bottom-up).
  - E. Means-end analysis. Compare initial problem state with final goal. If they are the same, then done. If not, then use differences to select next operator to apply to initial state to make a move to a new state, and **recurse** (oversimplified but adequate statement).
  
5. The reason that natural language understanding is very difficult **is** that knowledge of the domain of discourse is essential for effective performance. Since most natural language discourse can range widely over the things that humans know, and since the knowledge bases that computers have these days are small and specialized, there is a "knowledge gap" that makes N.L. understanding difficult.
  
6. The methods known to AI (e.g. all of the above, plus many other concepts, plus various programming techniques) are used as a **model-building** laboratory by **the** psychologist. The psychologist's information-processing theory of some aspect of human cognition is given precise expression in the form of a program that models the theory. Simulation runs give the predictions from the theory for particular experimental situations being studied. Some psychologists use the AI concepts only, and not the modeling/programming technique.

;

### Hardware 1

- A. Memory Time =  $.9 * 100ns + .1 * (100ns + 1000ns)$   
 = 200ns.
- Instruction Time = 300ns. + 1.75 \* Memory Time
- Instruction Rate = 1/instruction time = 1.538 Million Instructions/second
- B. Memory Time = Probability of Read \* Read Time +  
 Probability of write \* Write time  
 =  $\frac{2}{3} * 200ns + \frac{1}{3} * 1100ns = 500ns$
- Instruction Time = 300ns + 1.75 \* 500ns = **1175ns**
- Instruction Rate = 0.851 Million instructions/second

### Hardware 2

- A. A normalized number has significant information in the leftmost position of the fraction. In a binary radix, this is usually a "1" bit in the leftmost bit of the fraction.
- B. Normalization maximizes the significance of the number. That is, it preserves the most information about the number. Also, normalization facilitates comparisons.
- C. In a binary radix, a hidden bit is an unwritten (not stored) "1" bit that is assumed to be present in all numbers (except zero). Its purpose is to add one bit of fraction, by not storing the constant "1" bit,
- D. The smallest positive number, written in hex as 00100000 represents  $(0.1)_{16} * 16^{-64} = 16^{-65} = 5.3976 * 10^{-79}$
- E. The largest positive number written in hex as 7FFFFFFF  
 =  $(0.FFFFFFF)_{16} * 16^{63} = (1 - 2^{-24}) * 16^{63} = 7.237 * 10^{75}$
- F. 

41	800000	Adjust Exponent ->	42)	080000
+	42	F90000	42	F90000
--	w--	m-----	-----	
43	101000	<- Post Normalize	42	1010000
- G. 

451A6F3C	->	45	1A6F3C	45	1A6F3C
+	C1321850	->	41-32185D	Adjust Exponent->	45 - 000032
-----					-----
451A6F0A	<	-----	45	1A6F0A	

### Hardware 3

A. 
$$\begin{array}{r} 38 \\ 2 \overline{)77} \\ \underline{[1]} \end{array} \quad \begin{array}{r} 19 \\ 2 \overline{)38} \\ \underline{[0]} \end{array} \quad \begin{array}{r} 9 \\ 2 \overline{)19} \\ \underline{[1]} \end{array} \quad \begin{array}{r} 4 \\ 2 \overline{)9} \\ \underline{[1]} \end{array} \quad \begin{array}{r} 2 \\ 2 \overline{)4} \\ \underline{[0]} \end{array} \quad \begin{array}{r} 1 \\ 2 \overline{)2} \\ \underline{[0]} \end{array} \quad \begin{array}{r} 0 \\ 2 \overline{)1} \\ \underline{[1]} \end{array} \quad \text{c-remainders}$$

Read remainders from right to left  $77_{10} = 1001101_2$

$$\begin{array}{r} 0011001 \\ 11 \overline{)1001101} \\ \underline{011} \\ 11 \\ \underline{-11} \\ 0101 \\ \underline{-11} \\ 10 \end{array}$$

$11001_2 = 16 + 8 + 1 = 25_{10}$  (a sensible check)

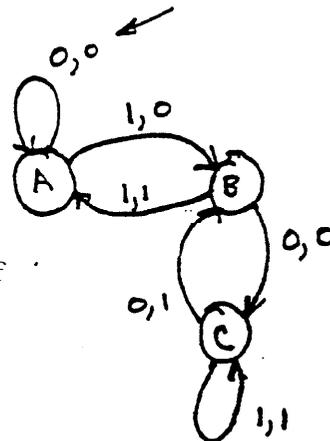
B. 

Old State	A	B	C
Input			
0	A,0	C,0	B,1
1	B,0	A,1	C,1

next state, output  
"A" is the initial state.

States A, B, C, Correspond to remainders of 0, 1, 2 respectively.

Input, Output



### Hardware 3C

3 states  $\rightarrow$  2 flipflops.  $Q_1, Q_2$  choose A=00 for easy initialization  
B=01 arbitrary choice  
C=10

$Q_1, Q_2$ Old State	Input	$D_1, D_2$ New State	output
00	0	00	0
00	1	01	0
01	0	10	0
01	1	00	1
10	0	01	1
10	1	10	1

$Q_1, Q_2$ In	00	01	11	10
0	0	1	X	0
1	0	0	X	1

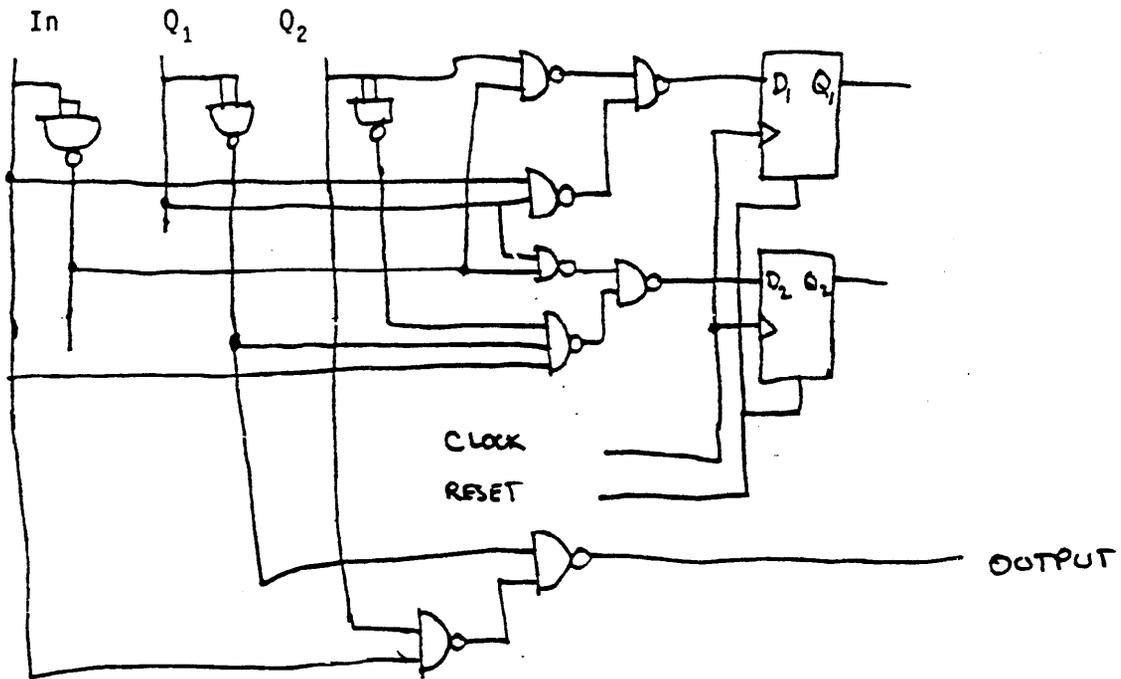
$D_1 = Q_2 \bar{I} + Q_1 I$

$Q_1, Q_2$ In	00	01	11	10
0	0	0	X	1
1	1	0	X	0

$D_2 = Q_1 \bar{I} + \bar{Q}_1 \bar{Q}_2 I$

$Q_1, Q_2$ In	00	01	11	10
0	0	0	X	1
1	0	1	X	1

Output =  $Q_1 + Q_2 I$



D. Use a ROM instead of all this logic. 3 address lines  $Q_1$ ,  $Q_2$ , Input. 3 outputs:  $D_1$ ,  $D_2$ , output. Data as specified in the state transitions table above.

## Winter 1983—Numerical Analysis (Solutions)

### Problem 1

- (a) Newton is second order locally and is not guaranteed to converge globally. Secant converges with order  $\frac{1+\sqrt{5}}{2}$  locally and is not guaranteed to converge globally. Bisection converges linearly if start with an interval for whose endpoints the function has differing signs.

Possible disadvantages are:

- |           |   |
|-----------|---|
| Newton    | (i) must know the derivative;   |
|           | (ii) two function evaluations per iterate;  |
|           | (iii) global convergence is not guaranteed and the root it does converge to cannot be specified unless a very good initial guess is provided. |
| Secant    | (i) can be unstable numerically near the root;  |
|           | (ii) global convergence is not guaranteed and the root it does converge to cannot be specified unless a very good initial guess is provided.  |
| Bisection | (i) need to find an interval for whose endpoints the function has differing signs;  |
|           | (ii) can converge relatively slowly.  |

- (b) With a double root: Newton converges linearly locally. Secant converges linearly locally. Bisection may not be applicable since an interval isolating this root for whose endpoints the function has differing signs may not exist.

For all three the error in the root is much larger for a given tolerance

$$|f(x_i) - f(x_{i+1})| < \epsilon .$$

To **fix** up Newton: use  $x_{i+1} = x_i - 2 \frac{f(x_i)}{f'(x_i)}$ , or else, work with  $f'(x)$ ;  $f'(x)$  will have a simple root at  $x = x_0$ .

**Winter 1983—Numerical Analysis (Solutions)**

**Problem 2**

(a) The equality follows by letting  $f(x) = 1$  in both quadrature formulas.

$$\int_a^b f(x)w(x) dx = \sum_{i=0}^n \omega_i f(x_i) = \sum_{i=0}^n \omega_i$$

and

$$\int_a^b f(x)w(x) dx = \sum_{i=0}^m v_i f(x_i) = \sum_{i=0}^m v_i,$$

so

$$\sum_{i=0}^m v_i = \sum_{i=0}^n \omega_i.$$

(b) The error formula given is of the form

(1)  $I_n = I + a_1 h^2 + a_2 h^4 + O(h^6)$  where  $a_1$  and  $a_2$  depend only on  $f$ .

By halving the mesh width, we obtain

(2)  $I_{2n} = I + a_1 \frac{h^2}{4} + \frac{a_2 h^4}{16} + O(h^6),$

and by halving it again, we obtain

(3)  $I_{4n} = I + \frac{a_1 h^2}{16} + \frac{a_2 h^4}{256} + O(h^6).$

To obtain a formula as stated, we eliminate the terms involving  $h^2$  in (1) and (2). In order to do this, we multiply (2) by 4 and subtract (1):

(4)  $4I_{2n} - I_n = 3I - \frac{3}{4}h^4 + O(h^6).$

Solving for I,

$$I = \frac{4I_{2n} - I_n}{3} + \frac{3}{4}h^4 + O(h^6),$$

and so  $C_1 = 4$  and  $C_2 = -1$ .

(c) To obtain a 6th order accurate formula, we first obtain a 4th order accurate by using (2) and (3).

Multiplying (3) by 4, we have

$$4I_{4n} = 4I + \frac{a_1 h^2}{4} + \frac{a_2 h^4}{64} = O(h^6),$$

and subtracting (2), we obtain

(5)  $4I_{4n} - I_{2n} = 3I - a_2 \frac{3}{64}h^4 + O(h^6).$

### Winter 1983—Numerical Analysis (Solutions)

Now we use (4) and (5) to obtain a formula without terms involving  $h^4$ .

Multiply (5) by 16:

$$64I_{4n} - 16I_{2n} = 48I - \frac{3}{4}h^4 + O(h^6).$$

Now subtract (4)

$$64I_{4n} - 20I_{2n} + I_n = 45I + O(h^6).$$

Solving for I:

$$I = \frac{64I_{4n} - 20I_{2n} + I_n}{45}.$$

### Problem 3

(a)

$$\begin{aligned} ax &= b \\ -(A\tilde{x} &= b + \tilde{r}) \\ \hline A(x - \tilde{x}) &= -\tilde{r} \end{aligned}$$

. therefore,

$$\begin{aligned} \|A(x - \tilde{x})\| &= \|\tilde{r}\| \\ \|A\| \|x - \tilde{x}\| &\geq \|\tilde{r}\| \\ \|x - \tilde{x}\| &\geq \|\tilde{r}\| / \|A\|, \end{aligned}$$

also

$$\begin{aligned} x &= A^{-1}b \\ \|x\| &= \|A^{-1}b\| \leq \|A^{-1}\| \|b\|, \end{aligned}$$

. therefore

$$\frac{\|x - \tilde{x}\|}{\|x\|} > \frac{\|\tilde{r}\|}{\|A\| \|A^{-1}\| \|b\|}.$$

(b) :

$$\tilde{x} - A^{-1}b = A^{-1}(A\tilde{x} - b)$$

therefore

$$\|\tilde{x} - A^{-1}b\| \leq \|A^{-1}\| \|A\tilde{x} - b\|.$$

Since this expression can be a strict equality, the norm of the error ( $\|\tilde{x} - A^{-1}b\|$ ) can be up to  $\|A^{-1}\|$  times as large as the norm of the residual vector ( $\|b - A\tilde{x}\|$ ). And  $\|A^{-1}\|$  can be arbitrarily large.

Software Answers

1. A process is active - "A virtual processor" that executes a program or part thereof, while a program is passive - a specification of instructions and data.

2.

```

procedure QueueId (id : integer);
begin
  1:  disable:
      if free = 0 then
        begin
          Wait (freespaces);
          got0 1;
        end;
      free := free - 1;
      next := next + 1;
      if next = 51 then next := 1;
      Queue[next] := id;
      used := used + 1;
      if used = 1 then Signal (available);
      enable;
end;

```

```

procedure DequeueId (var id : integer):
begin
  2:  disable:
      if used = 0 then
        begin
          Wait (available);
          got0 2;
        end;
      used := used - 1;
      id := Queue[last];
      last := last + 1;
      if last = 51 then last := 1;
      free := free + 1;
      if free = 1 then Signal (freespaces);
      enable;
end;

```

3. Deadlock prevention involves a system design such that one of the four necessary conditions for deadlock cannot arise; avoidance attempts to prevent the system from entering states that could lead to deadlock as it runs; detection lets the system deadlock but attempts to detect the occurrence of deadlock and recover.

Prevention - usually resources must be allocated in advance of need, introducing waste.

Avoidance - overhead for checking safety of resource allocation on each request plus waste of resources in avoiding potentially unsafe states (that do not necessarily lead to deadlock).

Detection - overhead of detecting deadlock - periodic **checking** plus wasted resources when computation is aborted or backed up to a check point.

4. a) Replace page that will not be needed for largest time.
- b) Impractical because need to predict future to determine page to replace. Least recently used - is common, which assumes low use in



## Winter 1983 - Software Systems

In ALGOL, storage for OWN variables is statically allocated but storage for all other variables must be dynamically allocated at **runtime**. A simple stack-based allocation scheme will suffice.

In Pascal, stack-based allocation works for declared variables; however, storage allocated with calls to NEW must be managed as well. This requires a heap-based allocation scheme.

1. Let  $g(x) = f(x,x) + 1$ . Since  $f$  is recursive so is  $g$ .  
**Suppose**  $g = f_n$  for **some**  $n$ . Then  
 $f_n(n) = g(n) = f(n,n) + 1 = f_n(n) + 1$ ,  
 a contradiction.
2. (a) This is an axiom of our favorite system and so is its own proof.  
 (b) A counterexample is provided by the integers with  $B$  interpreted as the standard linear order on the integers.
3. Notation:  $a.b = \text{cons}(a,b)$   
 $a@b = \text{append}(a,b)$   
 $a^R = \text{reverse}(a)$

The definitions then yield directly:

$$\begin{array}{ll} \text{nil}^R = \text{nil} & \text{R1} \\ (\text{a.b})^R = \text{b}^R @ (\text{a.nil}) & \text{R2} \\ \text{nil}@v = v & \text{A1} \\ (\text{a.b})@v = \text{a} . (\text{b}@v) & \text{A2} \end{array}$$

Lemma 1.  $x@nil = nil$

Proof. Use structural induction on  $x$ .

$$\begin{array}{ll} \text{Basis.} & \text{nil} @ \text{nil} = \text{nil} & \text{A1} \\ \text{Step.} & (\text{a.b}) @ \text{nil} = \text{a} . (\text{b}@nil) & \text{A2} \\ & = \text{a.b} & \text{Ind. Hyp.} \end{array}$$

Lemma 2.  $x@(y@z) = (x@y) @z$

Proof. Use structural induction on  $x$ .

$$\begin{array}{ll} \text{Basis.} & \text{nil} @ (y@z) = y @ z & \text{A1} \\ & = (\text{nil}@y) @z & \text{A1} \\ \text{Step.} & (\text{a.b})@(y@z) = \text{a} . (\text{b}@(y@z)) & \text{A2} \\ & = \text{a} . ((\text{b}@y) @z) & \text{Ind. Hyp.} \\ & = (\text{a} . (\text{b}@y)) @z & \text{A2} \\ & = ((\text{a.b})@y) @z & \text{A2} \end{array}$$

Theorem.  $(u@v)^R = v^R @ u^R$

Proof. Use structural induction on  $u$ .

$$\begin{array}{ll} \text{Basis.} & (\text{nil}@v)^R = v^R & \text{A1} \\ & = v^R @ \text{nil} & \text{Lemma 1} \\ & = v^R @ \text{nil}^R & \text{R1.} \\ \text{Step.} & ((\text{a.b})@v)^R = (\text{a} . (\text{b}@v))^R & \text{A2} \\ & = (\text{b}@v)^R @ (\text{a.nil}) & \text{R2} \\ & = (v^R @ b^R) @ (\text{a.nil}) & \text{Ind. Hyp.} \\ & = v^R @ (b^R @ (\text{a.nil})) & \text{Lemma 2} \\ & = v^R @ (\text{a.b})^R & \text{R2} \end{array}$$

## Winter 1983—Theory of Computation (Solutions)

### Problem 4

(a) Not regular, but context-free. Grammar:

$$S \rightarrow OA2$$

$$A \rightarrow OA2 \mid 1B$$

$$B \rightarrow 1B \mid \epsilon$$

To show nonregular, assume it were regular and use the pumping lemma. Then there is a constant  $k$  such that if  $z$  is in the language and  $|z| \geq k$ , then  $z = uvw$ , where  $|uv| \leq k$ ,  $|v| > 0$ , and for all  $i \geq 0$ ,  $uv^i w$  is in the language. Let  $z = 0^k 1^k 2^k$ , so  $v$  consists of  $0$ 's only, and  $uvw$  has more  $0$ 's than  $2$ 's, contradicting the fact that  $uvw$  is in the language.

(b) Not CF. Use the CF pumping lemma, which says that there is a constant  $k$  for which every  $z$  in the language, with  $|z| \geq k$ , can be written  $z = uvwxy$ , where  $|vwx| \leq k$ ,  $|vx| \geq 1$ , and for all  $i \geq 0$ ,  $uv^i w x^i y$  is in the language. Let  $z = 0^k 1^k 2^k$ . If the right end of  $uvw$  is in the  $0$ 's, then  $uw$  has fewer  $0$ 's than  $1$ 's. If the right end of  $uvw$  is in the  $1$ 's, then  $uw$  either has fewer  $0$ 's than  $1$ 's or fewer  $1$ 's than  $2$ 's, since it has  $k$   $2$ 's and at most  $2k - 1$   $0$ 's and  $1$ 's combined. If the right end of  $uvw$  is in the  $2$ 's, then  $v$  and  $x$  have no  $0$ 's, because  $|vwx| \leq k$ . Thus  $uvwxy$  has  $k$   $0$ 's and at least  $2k + 1$   $1$ 's and  $2$ 's. It therefore either has fewer  $0$ 's than  $1$ 's or  $1$ 's than  $2$ 's. In all cases we have derived a word that is not in the language (b), yet that the pumping lemma tells us must be there. We conclude (b) is not CF.

(c) Regular. Here is a regular expression:  $(00)^* (0 + 1) (11)^*$ .



Algorithms and Data Structures

1. Data Structures [16 points].

We'd like to maintain a data structure for a set  $S$ .  $S$  will contain elements from some linearly ordered universe  $U$ ;  $S$  may contain duplicates. We'd like our data structure to efficiently implement the following two operations:

*Insert*( $b$ ) : inserts the element  $b$  into  $S$  (this will add another copy of  $b$  if there already was one)

$x := \text{ExtractMin}()$ : deletes from  $S$  a smallest element (there may be more than one) among all elements currently in  $S$ , and returns the value

- (a) [8 points] Suppose  $U$  is the set of integers between **1** and **100**. Briefly describe a data structure (and how it implements *Insert* and *ExtractMin*) that has a good worst-case running time for both operations; estimate the worst-case running time for each operation (you needn't justify your estimates). Your estimates should be of the form  $O(f(n))$  for a suitable  $f(n)$ , where  $n$  is the number of elements in  $S$ . To receive full credit for this part, your data structure must have constant worst-case time (that is,  $O(1)$ ) for each operation.
- (b) [8 points] (the words in smaller type are the same as in part (a)) **Suppose  $U$  is the infinite set of integers (you may assume comparisons between them, and arithmetic and logical operations on them, take 1 time unit). Briefly describe a data structure (and how it implements *Insert* and *ExtractMin*) that has a good worst-case running time for both operations; estimate the worst-case running time for each operation (you needn't justify your estimates). Your estimates should be of the form  $O(f(n))$  for a suitable  $f(n)$ , where  $n$  is the number of elements in  $S$ . To receive full credit for this part, your data structure must have logarithmic worstcase time (that is,  $O(\log n)$ ) for each operation.**

2. Sorting [12 points].

Suppose we've just finished sorting  $n$  distinct large positive integers using our favorite  $O(n \log n)$  sorting method, when the earthquake strikes. Miraculously, the machine is unaffected except that, somehow, each of the 4 low order bits of each of our integers has been randomly set to 0 or 1, and now we want to sort these new integers. Sketch **an algorithm** that sorts them in  $O(n)$  time (you may assume comparisons between integers, and arithmetic and logical operations on them, take 1 time unit).

**3. Recurrence Relations [6 points].**

In the divide-and-conquer technique, we divide a problem into subproblems, recursively solve the **subproblems**, and combine their solutions to give a solution for the original problem. Specifically, suppose that some algorithm has the following three properties:

- (i) it divides a problem of size  $n > 1$  into 2 subproblems, each of size  $n/3$
- (ii) it divides the problem into subproblems and combines their solutions in time  $5n$
- (iii) it solves a problem of size  $n = 1$  in time 7.

Further suppose that  $n$  is a power of 3 so that we have the recurrence

$$T(1) = 7$$

$$T(n) = 2T\left(\frac{n}{3}\right) + 5n \quad \text{for } n > 1.$$

What is the order of the dominant term in the solution of this recurrence? That is, find a function  $f(n)$  such that

$$0 < \lim_{n \rightarrow \infty} \frac{T(n)}{f(n)} < \infty$$

Briefly justify your answer (one sentence indicating you understand the issues will do).

**4. -Algorithm Analysis [6 points].**

Assume  $n \geq 0$ . How many times is procedure S called in this program?

```

for k := 1 to n do
  for i := 0 to k-1 do
    for j := 0 to k - 1 do
      if i ≠ j then
        S(i, j, k)
    
```

Try to simplify your answer.

5. **NP-Completeness** [20 points].

Below are two pairs of decision problems. You may assume that the first member of each pair is a known NP-complete problem, while the second is, for our purposes, unknown. For each pair, tell whether or not the unknown member is NP-complete. If an unknown is NP-complete, you should sketch a **proof** of its NP-completeness; if it is not, you should sketch a polynomial-time algorithm for it (any correct polynomial-time algorithm, no matter how slow, will earn you full credit),

**PAIR #1**

**CLIQUE**

**INSTANCE:** A graph  $G = (V, E)$  and a positive integer  $J \leq V$ .

**QUESTION:** Does  $G$  contain a *clique* of size  $J$  or more, that is, a subset  $V' \subseteq V$  such that  $|V'| \geq J$  and every two vertices in  $V'$  are joined by an edge in  $E$ ?

**UNKNOWN #1** [10 points]

**INSTANCE:** A graph  $G = (V, E)$ .

**QUESTION:** Does  $G$  contain a *clique* of size  $|V| - 3$  or more, that is, a subset  $V' \subseteq V$  such that  $|V'| \geq |V| - 3$  and every two vertices in  $V'$  are joined by an edge in  $E$ ?

**PAIR #2**

**PARTITION**

**INSTANCE:** A finite set  $A$  and a *nonnegative integer* "size"  $s(a)$  for each  $a \in A$ .

**QUESTION:** Is there a subset  $A' \subseteq A$  such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) \quad ?$$

**UNKNOWN #2** [10 points]

**INSTANCE:** A finite set  $A$  and a *nonnegative integer* "size"  $s(a)$  for each  $a \in A$ .

**QUESTION:** Is there a subset  $A' \subseteq A$  such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$$

and  $|A'| = |A - A'|$  (that is,  $A'$  contains exactly half the elements of  $A$ )?

# Artificial Intelligence

Time: 1 hour

## 1. Heuristic Search (15 points)

Consider the cryptarithmic puzzle

$$\begin{array}{r} \text{FORTY} \\ \text{TEN} \\ +\text{TEN} \\ \hline \text{SIXTY} \end{array}$$

where each of the letters stands for a different integer between 0 and 9 inclusive such that the addition is correct.

[2 pts] (a) One way of solving this puzzle is blind search. For a given assignment of values to the ten letters, the proposed solution is either verified or refuted by back-substitution in the puzzle. In the worst case, how many potential solutions must be checked? In the average case? .

[5 pts] (b) So far, the only constraint we have used is the uniqueness constraint, *ie*, each letter stands for a different digit. A human being looking at the problem would derive other symbolic constraints, eg ,

$$S = F + 1$$

$$F \neq 0, S \neq 0, T \neq 0 \text{ from the knowledge that numbers don't start with } 0$$

State two other such constraints.

[8 pts] (c) If these symbolic constraints are used to prune off unacceptable partial solutions, search can be considerably reduced. Suggest a good sequence of assigning values to the letters in order to effectively utilize this pruning. Try to use as many of the constraints as you can from part (b) - those given as examples and the ones that you found. Briefly justify your answer.

## 2. Vision (10 points)

[4 pts] (a) Represent the following geometrical solids as generalized cones

- i) a square pyramid
- ii) a torus

[4 pts] (b) Edge finding is usually the first step in bottom-up processing in vision. How is it usually performed?

[2 pts] (c) State two ways of recovering depth information from image data.

## 3. Speech Understanding (5 points)

Name 3 sources of knowledge useful in speech understanding. Also name the program architecture used for incorporating these knowledge sources in the HEARSAY **program**.

## 4. Learning (10 points)

(a) What is the primary difference between Winston's arch learning program and Mitchell's version spaces algorithms? Is either one capable of learning concepts that the other cannot learn? If so, which? If not, why not?

(b) What is the powerful idea underlying the performance of Lenat's AM program? What is its primary weakness?

**5. Representation (10 points) .**

The following symbols describe various relations relevant to the **plant world**.

*purple(x)* states that "x is purple"

*mushroom(x)* states that "x is a mushroom"

*poisonous(x)* states that "x is poisonous"

*nearby(x,y)* states that "x is near y "

You may assume that  $\forall xy. (nearby(x,y) \equiv nearby(y,x))$

Represent the following sentences as well-formed formulas in predicate calculus using these symbols.

- (a) No purple mushroom is poisonous.
- (b) A mushroom is poisonous only if it is purple.
- (c) There are exactly two purple mushrooms.
- (d) If a mushroom is purple, then all mushrooms near it are poisonous.
- (e) There is a mushroom such that every mushroom near it has a purple neighbor.

**6. Resolution (10 points)**

Prove the validity of the following well-formed formula using resolution.

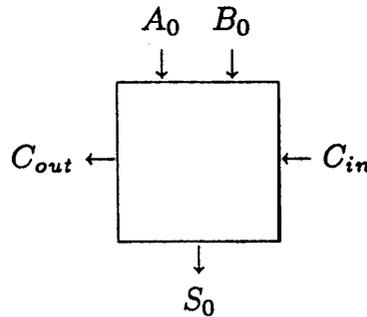
$$\forall x (P(x) \wedge (Q(A) \vee Q(B))) \Rightarrow \exists x (P(x) \wedge Q(x))$$

Good luck.

1. **Logic Design** [24 points].

A 12-fingered computer designer might find it natural to use a Binary-Coded Duodecimal system (BCDD), where the duodecimal (base 12) digits  $0, \dots, 9, A, B$  are represented by the four-bit combinations 0000,  $\dots$ , 1001, 1010, 1011. In this question you will design a full adder for BCDD, using only binary full adders.

(a) [6 points] Here is a diagram of a binary full adder:



Recall that a binary full adder takes in single bits  $A_0, B_0$ , and carry-in  $C_{in}$ . Its outputs are the binary sum,  $S_0$ , and the carry-out,  $C_{out}$ . Explain how to get the following boolean functions of single bits  $x$  and  $y$  using exactly one binary full adder for each (draw a full adder, label the inputs with  $x, y$  or constants, and indicate where to get the answer):

- (i)  $\bar{x}$
- (ii)  $x \wedge y$
- (iii)  $x \vee y$

(b) [3 points] A BCDD full adder is similar to a binary full adder, but it has 4bit inputs and outputs  $A_3A_2A_1A_0, B_3B_2B_1B_0$ , and  $S_3S_2S_1S_0$ , along with  $C_{in}$  and  $C_{out}$ . The output  $S_3S_2S_1S_0$  should be a proper BCDD number.

If  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  are added as ordinary binary numbers, what corrective action, if any, needs to be taken to get a proper  $S_3S_2S_1S_0$  and  $C_{out}$ ?

- (c) [13 points] Design a BCDD full adder using only binary full adders. Part **1(a)** implies that you may use inverters, 2-input AND, and 2-input OR gates.
- (d) [2 points] Give two reasons (relevant to computer design) why it would be **nice if we** had twelve fingers.

**2. Sequential Machine Design** [16 points].

A clocked state (Mealy) machine is to be built with one input bit,  $I$ , and one output bit  $O$ . The output  $O$  is to be 1 iff the last four  $I$  bits were **1101 (first 1, then 1, then 0, then 1)**. Overlaps are allowed, so that output  $O$  should be 1 twice in response to the input sequence **11011011**.

- (a) [7 points] Give a state diagram for a four-state deterministic **machine with this** behavior. Label a transition  $X/Y$  if it is to be taken when  $I = X$ , setting  $O = Y$ .
- (b) (9 points] Assign a bit pattern to each state (use all zeros for **the** start state), and for your state assignment find minimal equations for the **next-state bit pattern, and** the for value of  $O$ .

Partial credit will be given on question 2 for a machine with more **than four states**.

**3. Quickies** [20 points].

- (a) [3 points] There are a number of ways four bits could represent a signed binary number. Give the names of three such methods, along with the **range of representable** numbers and the representation of -1 in each case.
- (b) [3 points] What happens during a dynamic RAM refresh, and **why?**
- (c) [4 points] What is a master-slave flip-flop, and why is it used?
- (d) [3 points] What is non-restoring division?
- (e) [7 points]
  - (i) What is position-independent machine code?
  - (ii) Give an **architectural** feature that can be used to write position-independent code.
  - (iii) Why might position-independent code be chosen for a system incorporating ROMs?

Spring 1983  
 NUMERICAL ANALYSIS

1. Root Finding [20 points]

Consider the polynomial  $p(x) = x^3 + 3x^2 + 2x - 1$

(a) [5 points] \*Show that there is a positive root  $c$ , and find an interval of the form  $[a, a+1]$ , where  $a$  is an integer, that contains  $c$ .

(b) [10 points] Consider the iterative method

$$x_{n+1} = x_n - mf(x_n) \text{ where } x_0 \text{ is in } [a, a+1]$$

and

$$f(x) = \begin{cases} p(a), & x < a \\ p(x), & a \leq x \leq a+1 \\ p(a+1), & a+1 < x \end{cases}$$

( $a$  and  $p$  are as above.)

Find a value of  $m$  for which this method converges to  $c$  for any  $x_0$  in the interval  $[a, a+1]$ . Justify your answer. Discuss what happens if  $x_0 < a$ .

(c) [5 points] Does the iteration  $x_{n+1} = \frac{1}{2}(1 - x_n^3 - 3x_n^2)$  converge for all choices of  $x_0$  in the interval  $[a, a+1]$  (where  $a$  is as above) ?

2. Linear Algebra [15 points]

Suppose that  $A$  and  $C$  are square matrices and  $\|I - CA\| < 1$  in some matrix norm.

(a) [5 points] Show that both  $C$  and  $A$  are invertible.

Hint: Suppose first that  $Ax = 0$  for some  $x \neq 0$ .

You may choose a particular norm or prove the result in general.

(b) [10 points] Show that the iteration  $x^{(m+1)} = x^{(m)} + C(b - Ax^{(m)})$  will converge to the solution of  $Ax = b$  for any starting vector  $x^{(0)}$ .

## Spring 1983 - Numerical Analysis

3. Pot Pourri [25 points total, 5 points per questions]

(a) Suppose that  $A$  is a tridiagonal matrix. State (without proof) a non-trivial condition on  $A$  so that the linear system  $Ax = b$  can be solved by Gaussian elimination without pivoting.

(b) Suppose that you want to write a computer program to evaluate

$$y = \sqrt{e^{2n} - 1} - \sqrt{e^{2n}} \quad \text{for large } n.$$

If  $y$  is calculated directly from the formula, the **roundoff** error may be large. Give an alternative way to evaluate  $y$  in order to decrease **roundoff** error.

(c) Give a reason why one might prefer to use an iterative solution over Gaussian elimination to solve a linear system of equations.

(d) What is meant by a "cubic spline function?" Explain why such a function can be better suited than a polynomial for approximation over an interval.

(e) When can Aitken extrapolation be used to increase the order of convergence of a method for ~~finding~~ *finding* a root of a polynomial equation.

⋮

1. [8 points] Referencing Environments

- (a) (4 points] Suppose we have the following block structure in a statically **scoped** language such as PASCAL:

```
• procedure P;  
  • procedure Q;  
    procedure R;  
      end R;  
    end Q;  
  procedure S;  
    end S;  
end P;
```

Assume that P calls S, then S calls Q, then **Q calls** itself recursively, and finally the last activation of Q calls R. Sketch a picture of the activation record stack and show the links of the static chain.

- (b) [4 points] Why are both the dynamic and static chains desirable in such a language?

2. [14 points] Symbol Tables

Assume you are writing a PASCAL compiler.

- (a) [5 points] Name 5 things other than declared variables and type definitions which might be entered in the symbol table.
- (b) (3 points] A symbol table entry will typically have some fields that are in common for all entries and some that are specific to certain kinds of entries. What fields would be in common for all kinds of entries?
- (c) [3 points] What additional fields would you need in an entry for a declared variable?
- (d) (3 points] What additional fields would you need in an entry for a definition of an array type?

3. [12 points] Code Generation

Consider the following code fragment:

```

    if e <> 0
    then y := (d/e)*x + b;

```

- (a) [3 points] Show the intermediate code generated using quadruples as the intermediate language.
- (b) [5 points] Show the intermediate code generated using the following stack-based intermediate language:

<u>Op code</u>	<u>Meaning</u>
PUSH n	Push contents of n onto stack
PUSHA n	Push address of n onto stack (n may be a label)
+	push (pop <sub>2</sub> () + pop <sub>1</sub> () )
	push (pop <sub>2</sub> () - pop <sub>1</sub> () )
*	push (pop <sub>2</sub> () * pop <sub>1</sub> () )
/	push (pop <sub>2</sub> () / pop <sub>1</sub> () )
:=	pop <sub>2</sub> () := pop <sub>1</sub> () (pop <sub>2</sub> () should be an address)
JEQ	if pop <sub>1</sub> () = 0 goto pop <sub>2</sub> ()
JNE	if pop <sub>1</sub> () <> 0 goto pop <sub>2</sub> ()

Note: push (x) pushes x onto the stack; pop() pops the stack and returns the value. If pop() has subscripts they indicate the order that the pops are performed.

- (c) [4 points] Give two advantages of quadruples over stack code and two advantages of stack code over quadruples.

4. [5 points]                    **Attributes**

In a file system, an 'attribute' is information about a file that is not either the file name or any part of the contents of the file. List five attributes that are likely to be found in a file system.

5. [16 points]                    **File Organization**

- A. A physical record is the smallest unit of information that can be written on or read from a disk. Generally, a file is composed of more than one physical record. Briefly describe TWO ways to organize the physical records that constitute a file. (That is, describe TWO data structures suitable for representing a file that contains many physical records.)
- B. For each of the organizational methods that you described in part A, how can random access to a given physical record be achieved? (That is, for your methods of part A, tell how to find, e.g., the sixth physical record.)
- C. A "directory" or "catalog" is the data structure that relates file names to file contents. Assume that a directory is stored as a file. Describe TWO data structures suitable for storing data within a directory.
- D. Briefly describe TWO ways to keep track of free space in a file system. That is, what data structures might be useful for describing what physical records are available for allocation to new (or expanding) files?

6. [5 points]                    **Seek Optimization**

A multi-user operating system should try to maximize file system throughput. Some systems attempt to minimize seek-time (and thus increase disk throughput) by re-ordering the input/output requests that the users present to the system. (Assume that the file system is constructed so that short-term re-ordering of requests does not affect file system integrity.) The following seek-time optimization technique is called "shortest seek-time first": of all available requests, process the one that requires the shortest seek-time.

What problem does this technique cause in a multi-user environment?

Spring 1983

MATHEMATICAL THEORY OF COMPUTATIONFormal Languages (16 points)

(10 points) 1. Prove that the language  $L = \{0^n 1^n 0^n \mid n \geq 0\}$  is not context free.

(6 points) 2. If a language  $L \subseteq \{0,1\}^*$  is regular, is  $\{w \in L \mid w = 0^i 1^j 0^k, i \geq 0, j \geq 0, k \geq 0\}$  regular? Prove your claim.

Program Verification (20 points)

3. Consider the function  $\text{foo}(x)$  defined by:

```
function foo (x:integer);
begin
  y := x;
  while y>1 do
    y := y-2;
  return (y);
end
```

Using standard program verification techniques,

(6 points) a. Prove that  $\text{foo}(x)$  terminates for all non-negative integers  $x \geq 0$ .

(14 points) b. Prove that for all non-negative integers  $x \geq 0$ ,  $\text{foo}(x)$  computes  $x \bmod 2$ .

Recursion Theory (12 points)

(6 points) 4. a. Show that the range of a recursive function is not necessarily recursive.

(6 points) b. Show that the range of a strictly increasing recursive function is recursive.

Predicate Logic (12 points)

(5 points) 5. Given a set of axioms  $A_0, \dots, A_n$ , and the standard rules of inference, what constitutes a proof of a sentence  $\sigma$ ?

(7 points) 6. For a simple language in which each formula  $\psi$  has one of the following forms:

- i. An atomic formula  $R(x)$ ;
- ii.  $\neg \phi$  where  $\phi$  is a formula;
- iii.  $\phi_1 \wedge \phi_2$  where  $\phi_1, \phi_2$  are formulas;
- iv.  $\forall x. \phi$  where  $\phi$  is a formula;

when is a formula  $\psi$  satisfied under an interpretation  $I$  over a domain  $D$ . (Give an inductive definition. You may assume  $\psi$  has no free variables.)

## ANSWERS -- ALGORITHMS AND DATA STRUCTURES

la) Maintain an array A of length 100, initialized to 0.  $A[i]$  gives the number of elements in S whose value is i. To INSERT i merely increment  $A[i]$ . To EXTRACT-MIN, scan up the array starting at  $A[1]$  until some  $A[j]$  is nonzero, and decrement it; j is a minimum in S. Both operations work in constant time,  $O(1)$ , in the worst case. Alternatively, implementing the array A as a heap as in part b) will also have  $O(1)$  worst-case time complexity, since the heap will have at most 100 elements; this implementation will be slightly faster in some cases.

lb) A heap works here, implemented as an array  $A[1]$  to  $A[n]$ .  $A[1]$ , the root, contains a smallest element. The two children of the node  $A[j]$ , if they exist, are  $A[2j]$  and  $A[2j+1]$ ; neither child's value is smaller than the parent's. To INSERT a new element, put it at  $A[n+1]$  and filter it up the tree, swapping it with its parent, if the parent is larger, until it's not smaller than its parent. To EXTRACT-MIN, return the root (which contains a smallest element), replace it with the element  $A[n]$ , and filter this element down the tree, swapping it with the smaller of its two children, until it's not larger than either child; A now has n-1 elements. Both operations take  $O(\log n)$  time in the worst case. (A balanced-tree scheme, though slower, could also work in  $O(\log n)$  time in the worst-case.)

2) The following program sorts the new integers in  $O(n)$  time. Assume the integers were originally in increasing order in  $a[1]$  through  $a[n]$ . (indentation indicates the begin-end structure)

```
(1) a[0] := 0
(2) for i := 2 to n do
(3)     j := i
(4)     while a[j] < a[j-1] do
(5)         swap (a[j], a[j-1])
(6)         j := j - 1
```

Note that this is similar to bubblesort, which ordinarily performs  $O(n^2)$  comparisons; here however, for each of the n-executions of the FOR loop, the WHILE loop is executed at most 15 times (since there are at most  $2^4 - 1 = 15$  elements 'below'  $a[i]$  larger than it), so the program performs  $O(n)$  total comparisons (at line (4)). To see that this program is correct, just note that line (3) has the invariant " $a[1]$  through  $a[i-1]$  are sorted" and that the WHILE loop correctly inserts  $a[i]$ .

3) The basic issues are that there are 2 subproblems of size  $n/3$  and that the subdivision and recombination is linear. Since 2 is less than 3 we have an  $O(n)$  algorithm. Were 2 equal to 3 we'd have an  $O(n \log n)$  algorithm.

Alternatively, we could unroll the recurrence to find that the algorithm runs in time  $15n + \text{lower order terms}$ ; this is  $O(n)$ .

4) For each execution of the outermost loop, the statement "if  $i <> j$  then" is executed  $k^2 = k^2$  times, and on  $k$  of these the procedure  $S$  is not called; on the remaining  $k^2 - k$  times it is called. Thus, it's called a total of  $\sum_{k=1}^n (k^2 - k)$  times, which is  $\sum_{k=1}^n k^2 - \sum_{k=1}^n k = n(n+1)(2n+1)/6 - n(n+1)/2$  or  $(n^3 - n)/3$ .

Alternatively, we note that there are two cases,  $i < j$  and  $i > j$ , and by symmetry they are equal in number. For the case  $i < j$ , calls are made for all  $i, j, k$  satisfying  $0 \leq i < j < k \leq n$ , or equivalently  $1 \leq i+1 < j+1 < k+1 \leq n+1$ ; the values of  $i+1, j+1$ , and  $k+1$  can be any three distinct values in the range from 1 to  $n+1$ ; there are  $\binom{n+1}{3}$  such numbers; multiplying by two we get  $(n^3 - n)/3$ .

5) UNKNOWN #1: Solvable in polynomial time. If there is a clique of size  $|V|-3$  or more, there must be one of size  $|V|-3$ . Thus, for each of the possible  $\binom{|V|-3}{2} = O(|V|^3)$  such cliques, we can check whether all  $\binom{|V|-3}{2}$  edges between the  $|V|-3$  vertices exist. We can do this in time  $O(|V|^2)$  for each possible clique, so this algorithm runs in total time  $O(|V|^5)$ , which is polynomial.

UNKNOWN #2: NP-complete. The problem is in NP since in polynomial time we can guess a subset  $A'$  and check that  $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$  and that  $|A'| = |A-A'|$ . We transform PARTITION to UNKNOWN #2 in polynomial time as follows. Given a PARTITION instance, we construct an UNKNOWN #2 instance merely by padding the set  $A$  with 0's. That is, we form a set  $B$  so that  $|B| = |A|$  and we define the size function on  $B$  to be  $s(b)=0$  for all elements  $b$  in  $B$ . The set  $A \cup B$  together with the size function gives us our UNKNOWN #2 instance. We need only show that one instance has a 'yes' answer if and only if the other does. If the PARTITION instance has a 'yes' answer then some subset  $A'$  of  $A$  has the same sum of sizes as  $A-A'$ . We use this same  $A'$  and  $A-A'$  for the UNKNOWN #2 instance, adding appropriately many 0's from  $B$  -- that is,  $A'$  together\* with  $|A-A'|$  0's from  $B$  forms one set while  $A-A'$  together with  $|A'|$  0's from  $B$  forms the other. These two sets have the same sum of sizes and the same cardinality, so the UNKNOWN #2 instance has a 'yes' answer. Conversely, if the UNKNOWN #2 instance has a 'yes' answer then, ignoring the  $B$ -elements, we must have a subset  $A''$  of  $A$  such that  $\sum_{a \in A''} s(a) = \sum_{a \in A-A''} s(a)$ , since the  $B$ -elements all have size 0. But by construction this  $A''$  and  $A-A''$  works for the PARTITION instance as well, so it too has a 'yes' answer.

1.(a) In the worst case all the  $10!$  combinations of assignments must be checked. In the average case we expect to check  $10!/(n+1)$  potential solutions where  $n$  is the number of solutions for the puzzle (It turns out that  $n$  is 1)

(b)  $N=0$  or  $5$

$$I=0+1(\text{mod } 10) \text{ or } I=0+2(\text{mod } 10)$$

(c) The broad idea is to proceed from the most tightly constrained to the least tightly constrained.  $N$  is the most tightly constrained (only two choices) and so we put a trial value to it first. Next try  $E$  for which you have the same constraint.  $R$ ,  $T$ , and  $X$  can be assigned in any order. Assignments to  $F$  and  $S$  should be done together so as to utilise the pruning potential of the constraint  $S = F + 1$ . Similarly for  $O$  and  $I$ .  $Y$  should be assigned last as there is no constraint involving it.

2.(a) A square pyramid is a generalized cone with the sweeping area a square, a spine which is a straight line perpendicular to the square at its midpoint, and a linearly shrinking sweeping rule.

For a torus the sweeping area is a circular crosssection, the spine is a circular arc to which the circular crosssection is kept at right angles, and the sweeping rule is to keep the sweeping area constant.

(b) Edge finding is usually a two step process. The first step is convolution with an operator (Laplacian of a Gaussian is currently quite popular) followed by either a thresholding of the output or a detection of the zero-crossings (depending on the operator used). This gives a set of candidate edge elements. The next step is to link neighboring edge elements into extended edges.

(c) Stereo (different views of the same scene)

Shape from shading. This actually produces a map of orientation vectors which can be integrated to give depth.

3. Phonetics- Physical characteristics of the sounds in words of the vocabulary.

Syntax- Rules of sentence formation enabling us to eliminate certain word combinations like 'King horse fast blue is'.

Semantics- Meanings of words and sentences enabling us to throw out meaningless but grammatically legal sentences like 'The tree hunts wild animals\*.

and many others (AI handbook Vol I pg 332)

The HEARSAY program used the BLACKBOARD architecture for incorporating diverse knowledge sources.

4.(a) Winston's system conducts a depth first search of the concept space and keeps a SINGLE current concept description. The version-space algorithm is a breadth-first search algorithm and maintains a SET of plausible hypotheses. The version-space algorithm can learn an incompletely specified concept which Winston's program cannot do. Using the single current concept description which is kept is incorrect as wrong choices could have been made during the depth-first search.

(b) AM uses a set of heuristics and operators to go from existing concepts to new concepts and decide on their interestingness. The power springs from the fact that a small and general purpose set could be found which was adequate to go from a few set-theoretic concepts to fairly deep results in elementary number theory. Its weakness is that the discovery of new concepts is limited to those expressible in the representation language used for the old concepts. The close link between elementary mathematics concepts and LISP accounts for its success in that domain.

5.(a)  $\sim \exists x. \text{mushroom}(x) \wedge \text{poisonous}(x) \wedge \text{purple}(x)$

- (b)  $\forall x. (\text{mushroom}(x) \wedge \text{poisonous}(x) \supset \text{purple}(x))$   
 (c)  $\exists xy. (\text{purple}(x) \wedge \text{mushroom}(x) \wedge \text{purple}(y) \wedge \text{mushroom}(y) \wedge x \neq y \wedge (\forall z. (\text{purple}(z) \wedge \text{mushroom}(z) \supset z=x \vee z=y)))$   
 (d)  $\forall x. (\text{mushroom}(x) \wedge \text{purple}(x) \supset (\forall y. \text{nearby}(x,y) \wedge \text{mushroom}(y) \supset \text{poisonous}(y)))$   
 (e)  $\exists x. (\text{mushroom}(x) \wedge (\forall y. \text{mushroom}(y) \wedge \text{nearby}(x,y) \supset (\exists z. \text{purple}(z) \wedge \text{nearby}(y,z))))$

6. First negate the formula and convert to conjunctive normal form.

$$\begin{aligned} & \neg(\forall x (P(x) \wedge (Q(A) \vee Q(B))) \supset \exists x. (P(x) \wedge Q(x))) \\ & \forall x. (P(x) \wedge (Q(A) \vee Q(B))) \wedge \neg \exists x. (P(x) \wedge Q(x)) \\ & \forall x. (P(x) \wedge (Q(A) \vee Q(B))) \wedge \forall x. (\neg P(x) \vee \neg Q(x)) \\ & \forall x. (P(x) \wedge (Q(A) \vee Q(B))) \wedge \forall y. (\neg P(y) \vee \neg Q(y)) \\ & \forall xy. (P(x) \wedge (Q(A) \vee Q(B))) \wedge (\neg P(y) \vee \neg Q(y)) \end{aligned}$$

which gives us the clauses

$$\begin{aligned} & P(x) \\ & Q(A) \vee Q(B) \\ & \neg P(y) \vee \neg Q(y) \end{aligned}$$

Resolving the first and third clauses we get

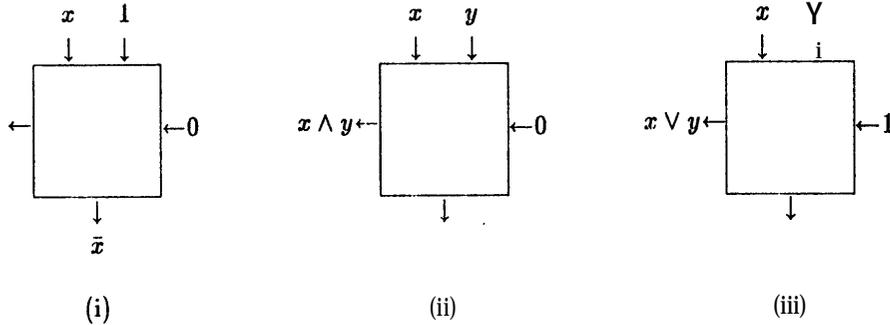
$$\neg Q(y)$$

Resolving this with the second clause we get the null clause.

Answers to Hardware section

1. Logic Design

(a)



Any permutation of the labels on the  $A_0$ ,  $B_0$ , and  $C_{in}$  inputs will also work.

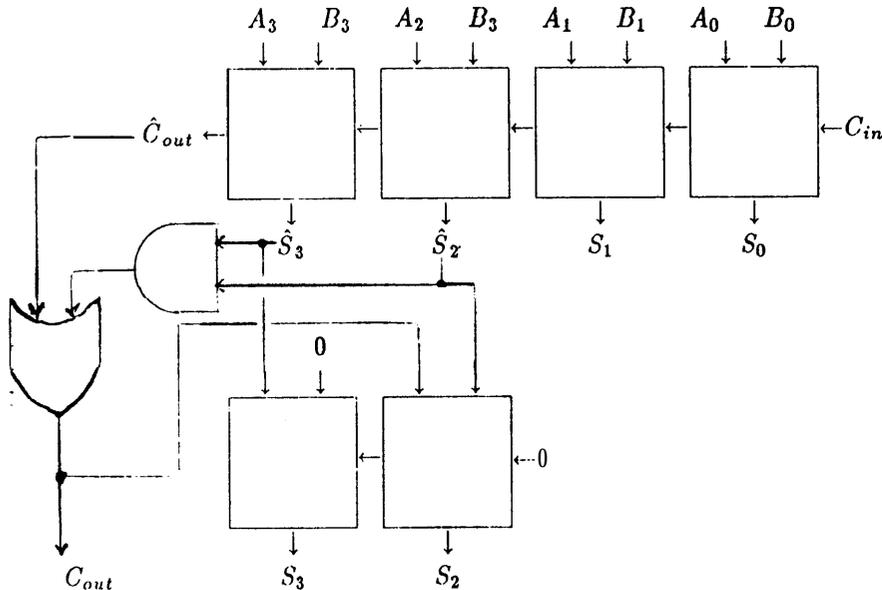
(b) If the binary sum  $\hat{C}_{out}\hat{S}_3\hat{S}_2\hat{S}_1\hat{S}_0$  is less than  $12_{10}$ , then the BCDD sum  $C_{out}S_3S_2S_1S_0 = \hat{C}_{out}\hat{S}_3\hat{S}_2\hat{S}_1\hat{S}_0$ . Otherwise, we must set

$$C_{out} = 1$$

$$S_3S_2S_1S_0 = \hat{C}_{out}\hat{S}_3\hat{S}_2\hat{S}_1\hat{S}_0 - 12_{10}.$$

This is sufficient when the inputs are proper BCDD numbers, since the sum will never be more than  $23_{10}$ .

(c) The formula  $\hat{C}_{out} \vee \hat{S}_3\hat{S}_2$  is true iff we have to correct the binary sum. The subtraction of  $12_{10}$  can be done by adding  $10100_2$  to  $\hat{C}_{out}\hat{S}_3\hat{S}_2\hat{S}_1\hat{S}_0$ . If the inputs are proper BCDD numbers, only the four rightmost bits can be nonzero after this subtraction, so addition is only needed there.

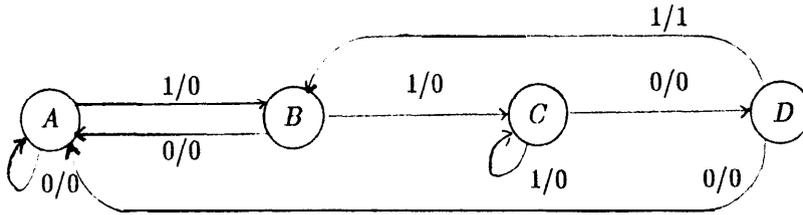


(d) Some possible answers:

- Compared to BCD, BCDD packs bigger numbers into the same number of bits.
- A larger percentage of 'natural' fractions would have exact representations.
- The logic for BCDD addition is slightly less complex than that for BCD addition.
- Typing might be easier(!)

2. Sequential Machine Design

(a) In the following machine, A is the start state.



(b) Use the state assignment  $A = 00$ ,  $B = 01$ ,  $C = 10$ ,  $D = 11$ . If the current state is  $XY$ , then the next state  $xy$  and  $O$  are given in the following table:

$XYI$	$xyO$
0 0 0	000
0 0 1	010
0 1 0	000
0 1 1	100
1 0 0	110
1 0 1	100
1 1 0	000
1 1 1	011

Using Karnaugh maps, minimal equations can be found:

		$YI$			
		00	01	11	10
$X$	0	0	0	1	0
	1	1	1	0	0

$$x = X\bar{Y} \vee \bar{X}YI$$

		$YI$			
		00	01	11	10
$X$	0	0	1	0	0
	1	1	0	1	0

$$y = \bar{X}\bar{Y}I \vee X\bar{Y}\bar{I} \vee XYI$$

		$YI$			
		00	01	11	10
$X$	0	0	0	0	0
	1	0	0	1	0

$$O = XYI$$

3. Quickies

(a)

(i) Signed-magnitude. Range:  $-7 \dots 7$ . Representation of -1: 1001.

(ii) Twos-complement. Range:  $-8 \dots 7$ . Representation of -1: 1111.

(i) Excess-8. Range:  $-8 \dots 7$ . Representation of -1: 0111.

(b) During a dynamic RAM refresh, all of the memory cells are read and rewritten with their current values. This must be done every once in a while because charge leaks off the memory capacitors.

(c) A master-slave flip-flop consists of a master flip-flop whose output is connected to the input of a slave flip-flop. The master is enabled when the clock is high and the slave is enabled when the clock is low (or vice versa). This means that a system of master-slave flip-flops can avoid race conditions, since the inputs will not change state during the period when the master is enabled.

(d) Hardware division of fixed point numbers often works by repeatedly subtracting the divisor from the dividend, adding it back if it was bigger, and shifting the partial remainder left. In non-restoring division the **add-back** is omitted; instead, the negative partial remainder is shifted left and the divisor is *added* instead of subtracted (this compensates).

(e)

(i) Position-independent machine code can be put anywhere in memory and will execute properly (with no need for adjusting any part of the code).

(ii) PC-relative addressing or base register addressing can be used to get position-independent code.

(iii) A system incorporating ROMs may exist in different configurations, and the address of a ROM may change from configuration to configuration. If the ROM contains position-independent code, this will be no problem.

1. (a) Since  $p(0) < 0$  and  $p(1) > 0$ , there is a root in  $[0, 1]$ .

(b) The iteration is of the form  $x_{n+1} = g(x_n)$ .

This converges if  $|g'(x)| < 1$  in  $[0, 1]$ .

Choosing  $m=1/11$  will guarantee this.

Suppose the iteration converges to some number,  $r$ . Then

$$r = r - mf(r),$$

so  $f(r) = 0$ .

$\therefore p(r) = 0$  and  $r$  is in  $[0, 1]$ .

We need to show that  $c$  is the only root of  $p$  in  $[0, 1]$ . Suppose there is another one. Then Rolle's theorem implies that  $p'$  has a zero in  $[0, 1]$ . But  $p'(x) = 3x^2 + 6x + 2$ , which has no non-negative roots. Therefore  $c$  is the only root of  $p$  in  $[0, 1]$ . So the iteration converges to  $c$ . If  $x_0 < a$ , the  $x_i$ 's increase by  $m$  until inside  $[0, 1]$ .

(c) We gave credit to all answers to this question, because it was too hard. This iteration is of the form

$$x_{n+1} = g(x_n).$$

A correct solution could show that  $|g'(r)| > 1$  if  $r$  is any zero of  $p$ , indicating failure to converge.

2. (a) Suppose  $CAx = 0$  for some  $x \neq 0$ . Then  $(I - CA)x = Ix - 0 = x$ ,

$$\|(I - CA)x\| = \|x\|.$$

But  $\|(I - CA)x\| \leq \|(I - CA)\| \|x\| < \|x\|$ ,

since  $\|(I - CA)\| < 1$  and  $\|x\| > 0$ .

This contradiction implies that  $CA$  is non-singular. Therefore  $C$  and  $A$  are non-singular, and therefore invertible.

(b) Since  $A$  is invertible the solution,  $x$ , exists. Define  $e^{(m)} = x^{(m)} - x$ .

Then  $x^{(m+1)} = x + e^{(m)} + C(b - A(x + e^{(m)}))$

$$x^{(m+1)} - x = e^{(m)} + Cb - CAx - CAe^{(m)}$$

$$e^{(m+1)} = (e^{(m)} - CAe^{(m)}) + (Cb - CAx)$$

$$= (I - CA)e^{(m)} + C(b - Ax)$$

$$e^{(m+1)} = (I - CA)e^{(m)}$$

$$\|e^{(m+1)}\| \leq \|I - CA\| \|e^{(m)}\|.$$

Therefore the size of the error decreases by at least a constant factor (namely  $\|I - CA\|$ ) with each iteration. Therefore the iteration converges.

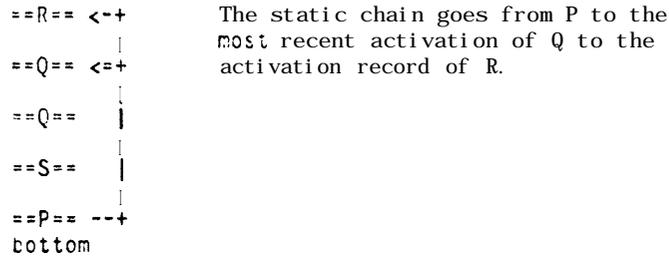
3. (a) Solution 1: A is positive definite.  
Solution 2: A is diagonally dominant.

$$(b) y = \frac{-1}{\sqrt{e^{2n-1}} + \sqrt{e^{2n}}}$$

- (c) If the matrix is sparse then an iterative solution can **save** time and space.
- (d) A cubic spline is an interpolate passing through **a set of points**  $x_0 < x_1 < x_2 < \dots < x_n$  that has a continuous derivative and is a cubic polynomial when restricted to any of the sub-intervals  $[x_{3k}, x_{3k+3}]$ . It **can** be preferred to a polynomial because it doesn't wiggle as much.
- (e) Whenever the convergence is linear.

Software #1 Solution

1a



1b The dynamic chain is desirable for efficient returns from blocks and subroutines. The static chain is desirable for efficient resolution of non-local references.

Software #2 Solution

2a

Among possible answers are procedure names, function names, labels, constants, values of enumerated types, reserved words, predefined types, the program name...

2b

Every entry requires the name of the symbol, the type of the symbol (identifier, label...), and some indication of the scope of the symbol.

2c

A declared variable also needs its type and information about where it is stored.

2d

An array type definition also needs its base type, the number of dimensions, and the type of each subscript.

Software #3 Solution

3a

Quads for this code fragment: (other solutions are also possible)

```

T1 := e = 0
JNE T1 NEXT
T2 := d/e
T3 := T2 * x
T4 := T3 + b
Y := T4
    
```

NEXT:

3b

Stack code for this code fragment:

```

PUSH NEXT
PUSH E
JEQ
PUSHA Y
PUSH D
PUSH E

PUSH X
•
PUSH B
+
:=
    
```

NEXT:

3c

Quads are closer to the actual instruction set for many machines. Quads are much easier to move around for optimization. Stack code eliminates the need to manage temporary variables at intermediate code generation time. Since it has no temporary variables, stack code can be easier to interpret, making it better for portable code generation.

## SPring 1983 - Software Systems (solutions)

### Software #4 Solution

File attributes include:

- size of the file
  - in records, in bytes,
- location of the file on the disk
- files dates
  - date of creation, of last write, of last access
- file people
  - creator, last writer, last reader, owner
- access control
  - protection bits, name of a program to run to validate access,
  - group membership
- backup status
  - date & location of backup copy
- billing information
  - who to charge for the space occupied

Some unacceptable answers were:

- list of users who are currently accessing the file;
- file position for each current reader of the file

These are characteristics of the operating system's file management are not part of the file in the sense of the attributes listed above.

Software #5 Solution

A. File Organizational Techniques:

1. contiguous allocation of a block of space (much as an array is allocated)
2. linked allocation. each physical record (or block) points to the next. A distinguishable pointer marks the last record.
3. index table of record addresses (or hierarchy of index tables).

B. Random access.

1. similar to array indexing. The sixth record is five records past the first one.
2. similar to following a list: must chain through pointers. (Not efficient for random access)
3. use array indexing to select the right pointer from the index table. use that pointer to access the disk.

C. Data structures for directory files:

1. Array of File Name/File Address records.
2. Array as above, but sorted by file names and kept compact for efficient lookup, e.g., binary search.
3. Hash table and linked buckets for efficient lookup. (Hard to scan in lexicographic order)

D. Data structures for keeping track of free space.

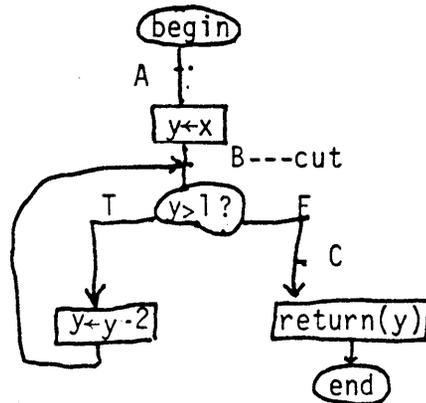
1. Create a file of free space. When a record is wanted, delete a record from the file of free space and allocate it as the new record.
2. Linked list of free records. (Or, a record of free records with a link to the next record of free records.)
3. Bit map. One bit per record to mark if the record is free or in use. The bit map is kept as a separate file. Helpful for finding contiguous space.
4. One bit per record, in the record, to signal used/free. Very disadvantageous as the disk becomes full, since the disk has to be scanned to locate free space. (partial credit)

Software #6 Solution

The problem with shortest seek time first is "starvation." Some processes may not get service while a small number of processes that do lots of I/O to a small region of the disk get all the service. This algorithm reduces the mean time to service, at the expense of increased variance.

Spring 1983 - Mathematical Theory of Computation (Solutions)

- Assume, by way of contradiction, that  $L$  is a context-free language. Using the pumping lemma for CFLs as in [Hopcroft & Ullman, p.125], let  $n$  be the constant of the lemma. Let  $z = 0^n 1^n 0^n$ . Write  $Z = uvwxy$  s.t.  $|vx| \geq 1$ ,  $|vwx| \leq n$  and  $uv^i wx^i y \in L$ , for  $i \geq 0$ .  $vx$  must have the form  $0^k 1^k 0^k$  for some  $k > 1$ , otherwise  $uw^i (=uv^i wx^i y)$  would not be in  $L$ . Therefore either  $v$  contains the substring  $01$  or  $x$  contains the substring  $10$ .  $uv^2 wx^2 y$  contains two substrings of the form  $01$ , or two substrings of the form  $10$ , and so  $uv^2 wx^2 y \notin L$ , since every  $w \in L$  has exactly one substring of each form. **Contradiction.**
- $\{w \in L : w = 0^i 1^j 0^k, i \geq 0, j \geq 0, k \geq 0\} = L \cap 0^* 1^* 0^*$ .  $0^* 1^* 0^*$  is regular,  $L$  is regular by assumption, and using the well known theorem [H&U, p.59], regular sets are closed under intersection.
- Our program corresponds to the flowchart in figure 1:



- For  $x=0$ , the while loop is never entered, and the program trivially terminates. For  $x > 0$ , the well founded relation we use is  $<$  on natural numbers. Every iteration of the while loop reduces the variable  $y$ , and when  $y < 1$ , the loop is exited.
- Use the following assertions at the cutpoints:

$$A: \mathcal{P}_A = x \geq 0$$

$$B: \mathcal{P}_B = 2 \mid x - y \wedge y \geq 0$$

$$C: \mathcal{P}_C = (y = x \bmod 2)$$

The assertion at A is true by assumption. Moving along the path AB, we have at B:  $x = y \wedge x \geq 0 \Rightarrow 2 \mid x - y \wedge y \geq 0$ . Assume  $\mathcal{P}_B$  is true at B.

Case 1:  $y > 1$  - the path BB is chosen.  $y$  is reduced by 2, so  $2 \mid x - y \wedge y > 1 \Rightarrow 2 \mid x - (y - 2) \wedge (y - 2) \geq 0$ , so  $\mathcal{P}_B$  is maintained.

Case 2:  $y \leq 1$  - the path BC is chosen.  $y \geq 0 \wedge y \leq 1 \wedge 2 \mid x - y \Rightarrow y = x \bmod 2$ .

Spring 1983 - Mathematical Theory of Computation (Solutions)

4. (a) Let  $M_1, M_2, \dots, M_n, \dots$  be a standard enumeration of Turing machines,

It is well known (Manna - p.57) that the set  $K = \{n : M_n \text{ halts on the empty word}\}$  is not recursive. We will show that it is r.e. : let  $f: \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$  be a 1-1 function from the natural numbers onto pairs of natural numbers. Let  $M$  be a TM s.t.  $\forall n$  if  $f(n) = \langle k, l \rangle$ ,  $M$  simulates  $l$  steps, of  $M_k$  computation on the empty word.  $M$  outputs  $k$  if that computation halted, loops otherwise.  $M$  halts on all inputs.

$$K = \{M(n) : n \in \mathbb{N}\}, \text{ so } K \text{ is r.e.}$$

Therefore,  $K$  is an r.e. set that is not recursive,

- (b) Let  $S = \{f(n) : n \in \mathbb{N}\}$ , where  $f$  is a strictly increasing function,  $f(0) \geq 0$ , and by easy induction  $f(n) \geq n$  for all  $n$ . For every  $m \in \mathbb{N}$ ,  $m \in S$  iff  $m \in \{f(i) : i \leq m\}$ . Since  $f$  is recursive, this last set is recursive, and so is  $S$ .
5. A proof of  $\sigma$  is a list of sentences ending in  $\sigma$ , each of which is either an axiom, or follows from previous ones by one of the inference rules.
6. (i)  $R(c)$ , is satisfied if the interpretation of  $R$  is true of the interpretation of  $c$  in  $D$ .
- (ii)  $\neg \phi$  is satisfied if  $\phi$  isn't.
- (iii)  $\phi_1 \wedge \phi_2$  is satisfied if both  $\phi_1$ , and  $\phi_2$  are.
- (iv)  $\forall x. \phi$  is satisfied if for all  $d \in D$   $\phi[d/x]$  is satisfied.  $\phi[d/x]$  denotes the result of substituting  $d$  for  $x$  in  $\phi$ , renaming the bound variables so that  $d$  will not be bound in  $\phi$ .



**Algorithms and Data Structures**

**Problem 1** (10 points).

Which of the following statements is true for arbitrary nonnegative-valued functions  $f(n)$  and  $g(n)$ , as  $n \rightarrow \infty$ ? Justify your answers.

- 1a.  $\max(f(n), g(n)) = O(f(n) + g(n))$ .
  - 1b.  $f(n) + g(n) = O(\max(f(n), g(n)))$ .
  - 1c.  $O(f(n)) = O(g(n)) = O(\max(f(n), g(n)))$ .
  - 1d.  $\min(f(n), g(n)) = \Omega(f(n) \cdot g(n))$ .
- .....  $g(n) = \Omega(\min(f(n), g(n)))$ .

**Problem 2** (10 points).

Given the formula  $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \ln n + \gamma + \frac{1}{2n} + O(n^{-2})$ , where  $\gamma$  is a constant, find constants  $a$  and  $b$  such that

$$\frac{1}{n+1} - \frac{1}{n+2} + \dots + \frac{1}{2n} - a - \frac{b}{n} = O(n^{-2})$$

**Problem 3** (5 points).

The **restricted n queens** problem consists of trying to place  $n$  queens on an  $n \times n$  chessboard so that (a) no two queens attack each other; and (b) no queen occupies any of the **squares** in a specified set of forbidden positions.

A formal description of the problem appears below, but you need not study it carefully, nor should you bother to verify that the formalism has anything to do with chess or queens. Your task is simply this: Prove or disprove (informally) that RESTRICTED QUEENS is in NP.

**RESTRICTED QUEENS**

INSTANCE: Positive integer  $N$ , and set  $S$  of pairs of integers  $\{(i_1, j_1), \dots, (i_M, j_M)\}$  where  $1 \leq i_k, j_k \leq N$  for  $1 \leq k \leq M$ .

QUESTION: Is there a permutation  $(a_1, \dots, a_N)$  of the integers  $\{1, \dots, N\}$  such that  $|a_i - a_j| \neq |i - j|$  for  $1 \leq i < j \leq N$  and such that  $(i, a_i) \notin S$  for  $1 \leq i \leq N$ ?

**Problem 4** (5 points).

Assume that **random(n)** is a function that returns a random integer value between 0 and  $n-1$ , inclusive, such that each of the  $n$  possible outcomes occurs with equal probability (independent of the values returned by previous invocations of **random**).

Let  $f(n)$  be the average number of calls to **random** when the following program is executed:

```

k := n;
while k > 0 do k := random(k) ;
    
```

State a recurrence that defines the values of  $f(n)$  for all integers  $n \geq 0$ . [You need not solve the recurrence; just come up with a correct one.]

Winter 1984

### **Algorithms and Data Structures**

**Problem 5** (15 points total).

Consider the following PASCAL program fragment, where *accumulator*, *divisor*, *m*, and *n* are integer variables and *x* is an array of integers.

```
accumulator := m; divisor := n;  
while divisor > 0 do  
  begin x[divisor] := accumulator div divisor;  
  accumulator := accumulator - x[divisor];  
  divisor := divisor - 1;  
end;
```

**5a** (4 points). If *m* and *n* are positive integers and if  $m = qn$  where *q* is an integer, prove that the above computation makes  $x[1] = \dots = x[n] = q$ .

**5b** (2 points). What is the final value of *accumulator*, assuming only that *m* and *n* are positive integers but not that *m* is a multiple of *n*?

**5c** (9 points). What is the final value of the array  $x[1..n]$ , if *m* and *n* are positive integers, and  $m = qn + r$ , where *q* is an integer and  $0 \leq r < n$ ? (Express your answer in terms of *n*, *q*, and *r*.)

**Problem 6** (15 points).

The deregulation of long distance phone service has spawned many new companies. One, the Tri-Tel Corporation, divides the USA into *n* regions in such a way that each region is directly connected to exactly three others.

We can model this situation with a graph, where each node corresponds to a region, and each (undirected) edge corresponds to a connection between regions. In this model, each node is adjacent to exactly three other nodes. We can assume that the graph is connected, so that a person in any region can call a person in any other region (possibly by a chain of connections).

A positive cost  $c(e)$  is associated with each edge of the graph; this is the amount that Tri-Tel charges customers to use the corresponding connection.

Your task is to design an algorithm that computes an  $n \times n$  matrix  $C = (C_{ij})$ , where  $C_{ij}$  is the least cost of a phone call from region *i* to region *j*, given the data in Tri-Tel's graph. You need not spell out the algorithm in detail; just give enough explanation to make your method clear, and mention the data structures that are used.

Estimate the worst case running time of your algorithm, and justify your estimate.

Note: There is an algorithm for this problem that takes  $O(n^2 \log n)$  steps. At least 10 points of partial credit will be given for any correct algorithm that runs in polynomial time.

*Artificial Intelligence*

**Problem 1 (10 points).**

Represent the sentences below as well-formed formulas in predicate calculus, using the following vocabulary:

- Person ( $x$ ) means that  $x$  is a person.
- Dog( $x$ ) means that  $x$  is a dog.
- Child( $x$ ) means that  $x$  is a child.
- Hates( $x, y$ ) means that  $x$  hates  $y$ .
- All bad ( $x$ ) means that  $x$  is all bad.
- Gentle ( $x$ ) means that  $x$  is gentle.
- Well trained( $x$ ) means that  $x$  is well-trained.
- Barks at( $x, y$ ) means that  $x$  barks at  $y$ .
- Lucky ( $x$ ) means that  $x$  is lucky.
- Owns( $x, y$ ) means that  $x$  owns  $y$ .
- Visits ( $x, y$ ) means that  $x$  visits  $y$ .

- 1a.** Any person who hates dogs and children can't be all bad.
- 1b.** Some dogs are gentle and well-trained.
- 1c.** Dogs are gentle only if they are well-trained.
- 1d.** A person is lucky if he has a dog that doesn't bark at everyone who visits its owner.
- 1e.** No person who barks at children is well-trained.

**Problem 2 (10 points).**

Here is some information about the text processing behavior of three individuals:

- If Knuth uses Scribe, then Floyd uses  $\text{\TeX}$ .
- Either Knuth or Reid uses Scribe, but they don't both use it.
- Reid and Floyd don't both use  $\text{\TeX}$ .
- Everyone uses either  $\text{\TeX}$  or Scribe or both.

Use resolution to prove that Reid uses Scribe. In your answer, let  $S(x)$  denote the predicate 'x uses Scribe', and let  $T(x)$  denote 'x uses  $\text{\TeX}$ '.

**Problem 3 (10 points total).**

Consider the following context free grammar for a limited subset of an English-like language:

- $S \rightarrow NP VP$
- $NP \rightarrow \text{determiner } NPR$
- $NP \rightarrow NP2$
- $NP2 \rightarrow \text{noun}$
- $NP2 \rightarrow \text{adjective } NP2$
- $NP2 \rightarrow NP2 P P$
- $PP \rightarrow \text{preposition } NP$
- $VP \rightarrow \text{verb}$
- $VP \rightarrow \text{verb } NP$

*Artificial Intelligence*

**3a (7 points).** Construct a Recursive Transition Network that defines the language derivable from  $S$ .

**3b (3 points).** This grammar allows non-English sentences, because (for example) it doesn't prohibit a sentence like 'Reid use Scribe' in which a singular noun is followed by a plural verb. How does an Augmented Transition Network solve this problem?

**Problem 4 (10 points total).**

Consider a domain in which each object has exactly two features, color and shape, which are chosen from the sets {red, blue, green} and {cube, pyramid, sphere}, respectively.

**4a (7 points).** Trace through Mitchell's candidate-elimination algorithm to learn a concept for which we have the following sequence of training instances:

- Positive: (red cube)
- Negative: (blue pyramid)
- Positive: (blue cube)

Indicate what the  $G$  and  $S$  sets are after the algorithm has seen each of these instances.

**4b (3 points).** Given an example of a concept in this domain that cannot be learned using candidate-elimination.

**Problem 5 (10 points total);**

**5a (3 points).** What is epipolar geometry in stereo vision? What is the major advantage of using it?

**5b (4 points).** Suppose you are given a stereo pair of  $512 \times 512$  images in which you want to correlate each  $8 \times 8$  square subarray in the left image with an  $8 \times 8$  square subarray in the right image. The pixels of the images are  $L[0..511, 0..511]$  and  $R[0..511, 0..511]$ , and the  $8 \times 8$  subarrays are  $l[p, q] = L[p..p+7, q..q+7]$  and  $r[p, q] = R[p..p+7, q..q+7]$  for  $0 \leq p, q \leq 504$ . The subarray  $r[u, v]$  that best matches a given subarray  $l[p, q]$  is one that maximizes the correlation coefficient.

$$C(p, q, u, v) = \frac{\sum L[p+i, q+j] R[u+i, v+j]}{\sqrt{\sum L[p+i, q+j]} \sqrt{\sum R[u+i, v+j]}}$$

over all choices of  $u$  and  $v$ . (Here  $\sum$  stands for  $\sum_{i=0}^7 \sum_{j=0}^7$ .)

Consider two methods for finding the  $r[u, v]$  that corresponds to each  $l[p, q]$ : Method **A** finds  $\max_{u,v} C(p, q, u, v)$  for all  $p$  and  $q$ . Method **B** assumes that the raster lines of both views  $L$  and  $R$  correspond to epipolar lines, hence it is possible to restrict consideration to the case  $u = p$ . About how much faster is Method **B** than Method **A**?

**5c (3 points).** Explain briefly how the identification of edges in  $L$  and  $R$  can be used to reduce the complexity of subarray matching still further.

**Problem 6 (10 points total).**

**6a (2 points).** What is nonmonotonic reasoning? Give an example.

**6b (2 points).** What is the frame problem?

**6c (2 points).** In what way is nonmonotonic reasoning helpful in solving the frame problem?

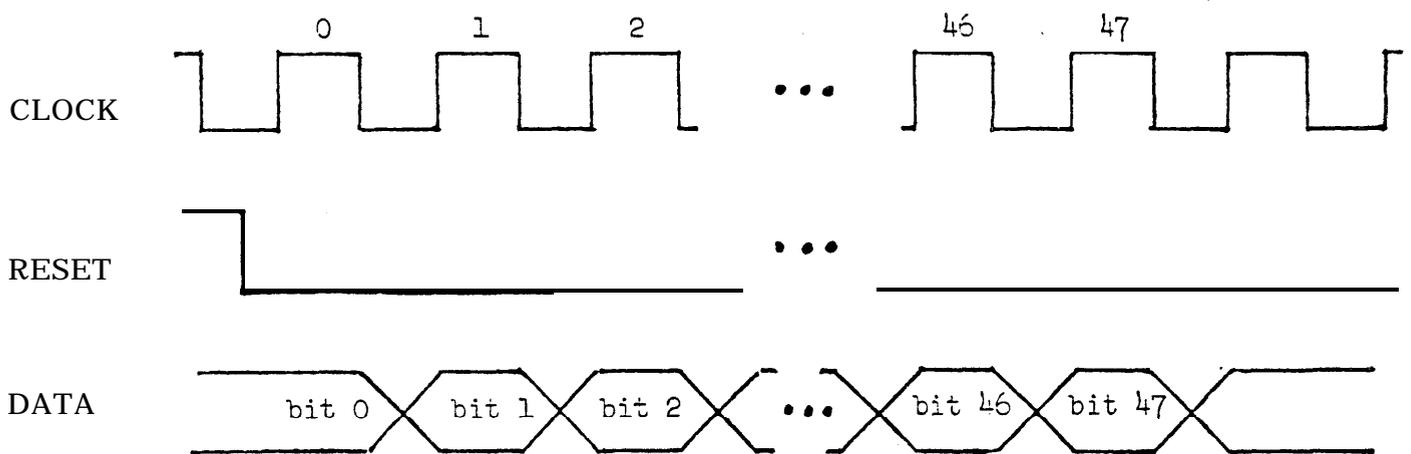
**6d (4 points).** Identify three techniques for conflict resolution in production systems.

**Hardware Systems**

Problem 1. (15 points total).

The Ethernet local area network uses addresses that are 48 data bits long. One of the problems in designing an interface for this network is to recognize a packet that is addressed to you.

We can assume that there is a serial DATA stream, a synchronized 10 MHz CLOCK, and a RESET signal. The RESET signal is true until the start of the address, when it drops to false, as shown in the following diagram:



1a (10 points). Draw a block diagram of hardware that will recognize a particular pattern of 48 data bits, assuming that the pattern will be known at the time of manufacture **but** not at the time of design. Your diagram should be at a level of detail that could easily be implemented in standard TTL. The device must produce its decision within a reasonable time after the address ends (say less than 5 clock cycles).

1b (2 points). If the 48-bit address to be recognized is already known at the time you are designing the circuit, outline if/how you would change the design in (1a).

1c (3 points). If the 48-bit address is to be programmable (i.e., dynamically changeable), outline if/how you would change the design in (1a).

Hardware Systems

Problem 2 (5 points).

For each of the devices/systems listed below, choose an appropriate technology and scale from the lists below, and briefly indicate why you made that choice. (There may be more than one reasonable answer; just give one.)

Technologies		Scales
CMOS	PMOS	Gate' array
ECL	RTL	Fully custom chip
I <sup>2</sup> L	TTL	Off- the-shelf ICs
NMOS	Williams Tube	

- 2a. A digital wristwatch.
- 2b. An implantable adaptive heart pacemaker.
- 2c. An image synthesizer for an arcade videogame.
- 2d. A LISP machine.
- 2e. A computer on board an orbiting spacecraft.

Problem 3 (5 points).

Before virtual paging became fashionable, various portions of programs and data would be overlaid in memory under program control, instead of using a demand paging scheme. List several advantages and disadvantages of overlaying versus virtual memory demand paging.

Problem 4 (15 points total).

Problems 4 and 5 refer to a hypothetical computer that has a serial byte I/O system; you can imagine that the serial I/O channel communicates with a dumb terminal. Figure 1 applies to both problems 4 and 5; the present problem deals with serial input, while problem 5 will deal with serial output.

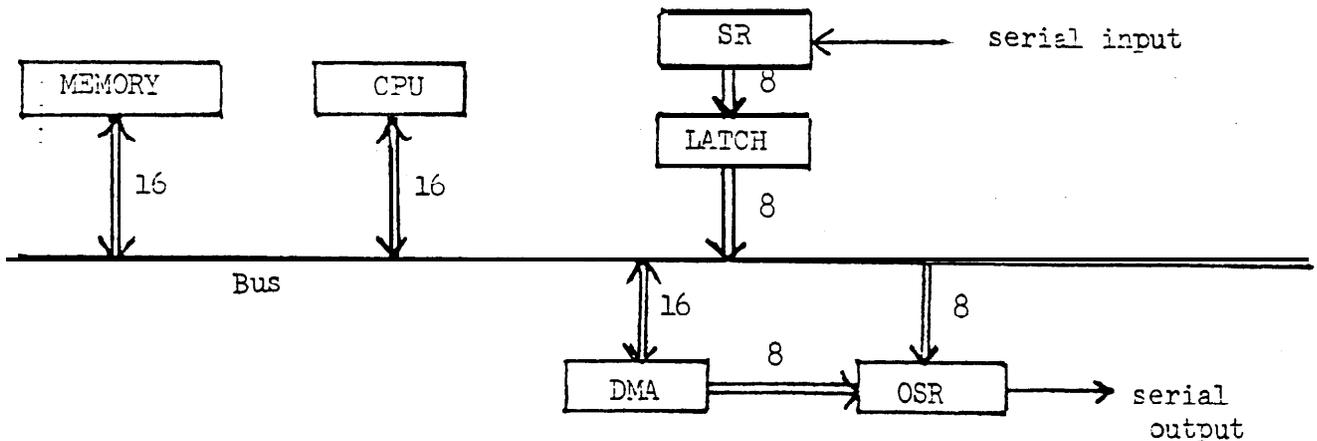


Figure 1. Hypothetical computer for problems 4 and 5. Items below the bus (namely the DMA and OSR) apply only to problem 5.

### **Hardware Systems**

The CPU is assumed to have eight levels of interrupts (0.. 7). As usual, when an interrupt request arrives at the CPU, it is not serviced immediately unless the priority of the incoming interrupt is higher than that of the interrupt currently being serviced. Furthermore, interrupts are serviced only between instructions.

A character that arrives on the serial input line is accumulated in the shift register SR. When it is complete, it is transferred into the LATCH, and the SR is free to receive a new character. Whenever a character is sent from the SR to the LATCH: an input interrupt signal is sent to the CPU. The CPU's input interrupt routine reads the latched character and places it in some sort of buffer (whose details are unimportant for purposes of this problem); then it clears the interrupt signal. We say that an **overrun** occurs if an input interrupt has not been completely serviced when a new character is ready to move from SR to LATCH.

The purpose of problem 4 will be to analyze certain performance properties of such an input system, based on the following quantities (some of which may turn out to be irrelevant to the analysis):

$T_{i,avg}$ , the average time to execute one CPU instruction.

$T_{i,max}$ , the maximum time to execute one CPU instruction.

$T_{avg}$ , the average time between input interrupt signals.

$T_{min}$ , the minimum time between input interrupt signals.

$T_{get}$ , the maximum time taken by the CPU to service an input interrupt.

**4a** (2 points). Assuming that no other interrupts occur besides input, what is the minimum  $T_{min}$  under which we can guarantee that no overruns occur? (This determines the maximum incoming character rate.) Give your answer as a formula of the form  $T_{min} \geq \dots$ , where the formula on the right-hand side gives a lower bound that is as tight as possible.

**4b** (3 points). If you want the processing of input characters to take at most  $P$  percent ( $0 < P < 100$ ) of the total CPU time, on the average, what is the minimum value of  $T_{avg}$ ? (Assume again that no other interrupts occur; give your answer as a formula of the form  $T_{avg} \geq \dots$ , where the right-hand side is a tight lower bound.)

**4c** (7 points). Assume now that the input interrupt is at level 6, and that there is a clock interrupt at level 7, occurring at fixed intervals  $T_{clk}$ . The CPU time needed to service this clock interrupt is  $T_{time}$ . What relationships must the parameters  $T_{clk}$ ,  $T_{time}$ ,  $T_{min}$ , etc., satisfy if overruns are to be impossible?

**4d** (3 points). If it is necessary to accept incoming data at a faster rate than that achievable with the system described above, what sort of hardware modifications would you suggest? Briefly describe an alternative approach by which higher-speed input transmission becomes possible.

**Problem 5** (20 points total).

This problem continues problem 4, but focusses on output instead of input. The hypothetical processor in Figure 1 has two ways to send characters to the serial output line:

*CPU output* occurs when the CPU simply stores a character in the output shift register (OSR). The OSR immediately begins to send the bits of the character; an output interrupt is signalled when the OSR is once again ready to service another character.

*DMA output* occurs when the CPU sets up the DMA to transfer a block of characters. In this case the DMA sends characters to the OSR at the maximum possible rate, and the CPU is interrupted only when the last character has been sent out of the OSR.

The CPU always outputs “messages” of characters in blocks of  $N$  characters each.

The purpose of problem 5 is to analyze the performance of such an output system, based on the following possibly relevant quantities:

$T_{iavg}$ , the average time to execute one CPU instruction.

$T_{imax}$ , the maximum time to execute one CPU instruction.

$T_{send}$ , the time required to send a character after it has been loaded into the OSR.

$T_{put}$ , the time to service an output interrupt, after CPU output.

$T_{iput}$ , the initial portion of  $T_{put}$  before a character is placed into OSR, if the output interrupt routine decides to output another character.

$T_{enddma}$ , the time to service an output interrupt, after DMA output.

$T_{stdma}$ , the CPU time needed to start the DMA output process.

$T_{stcpu}$ , the CPU time needed to initialize the CPU output process.

$T_{dma}$ , the average CPU time lost each time the DMA uses the bus to fetch a byte of memory.

$N$ , the number of characters in an output message.

**5a** (7 points). Assuming that no interrupts occur except for output interrupts, what is the maximum rate (chars/sec) at which characters can be sent using CPU output exclusively? What fraction of the CPU time is required to maintain this rate? (Give formulas.)

**5b** (7 points). Assuming that no interrupts occur except for output interrupts, what is the maximum rate (chars/sec) at which characters can be sent using DMA output exclusively? What fraction of the CPU time is required to maintain this rate? (Give formulas.)

**5c** (2 points). If  $N$  is very small, which method is better? Why?

**5d** (2 points). If  $N$  is very large, which method is better? Why?

**5e** (2 points). So, if you were the designer of a high performance computer, would you implement CPU output, DMA output, or both? Why?

**Problem 1** (25 points).

Note: This problem has 18 subparts, each of which may be answered simply 'true' or 'false', with no further commentary. Your score will be

$$2((\text{right answers}) - (\text{wrong answers})),$$

except that any score exceeding 25 will be worth 25 points, and any score less than **zero** will be worth zero points. Unanswered questions neither add nor subtract from your score. [Don't worry about the *details* of this formula; it *basically* says that random guessing won't help or hurt your *expected* score. *Just go ahead and give your best answers, spending about one minute* on each subpart.]

**1a.** True or false: Gaussian elimination without pivoting is stable for symmetric positive definite matrices.

**1b.** True or false: Gaussian elimination without pivoting is stable for symmetric non-singular matrices.

**1c.** True or false: Gaussian elimination with *partial* pivoting is always stable.

**1d.** True or false: Gaussian elimination with complete pivoting is always stable.

**1e.** True or false: If  $A$  is well-conditioned, then  $A + E$  is nonsingular for any  $E$  such that  $\|E\|$  is sufficiently small.

**1f.** True or false: If  $A$  is a nonsingular matrix, and if  $Ax = b$ ,  $(A + E)(x + e) = b$ , then

$$\frac{\|e\|}{\|x\|} \leq \frac{\|E\|}{\|A\|}.$$

**1g.** True or false: Iterative methods are always faster than direct methods for solving  $Ax = b$ .

**1h.** True or false: Tridiagonal systems of order  $n$  can be solved in  $O(n)$  operations.

**1i.** True or false: Interpolation by cubic splines is a stable procedure.

**1j.** True or false: Polynomial interpolation at equally spaced points is a stable procedure.

**1k.** True or false: Gaussian quadrature is more accurate than Romberg integration for sufficiently large numbers of quadrature points.

**1l.** True or false: For any choice of  $k$  distinct quadrature points, weights can be found to give a method that is exact for polynomials of degree  $k - 1$ .

**1m.** True or false: Romberg integration is unstable.

**1n.** True or false: The composite trapezoidal rule is a good method for integrating smooth periodic functions.

**1o.** True or false: The secant method always converges, but Newton's method may not.

**1p.** True or false: If  $f(\alpha) = 0$  and  $f'(\alpha) = 0$ , but  $f''(x)$  is continuous and nonzero at  $x = \alpha$ , then Newton's method converges linearly to the root  $\alpha$ .

**1q.** True or false: The iterative method  $x_{n+1} = g(x_n)$  cannot converge to the fixed point  $\alpha = g(\alpha)$  if  $g(x) < x$  for  $x < \alpha$  and  $g(x) > x$  for  $x > \alpha$ , unless  $x_0 = \alpha$ .

**1r.** True or false: The order of convergence of the iteration  $x_{n+1} = x_n^3$  is 4, if  $x_0 \in (-1, 1)$ .

*Numerical Analysis*

**Problem 2** (15 points total).

The iteration  $x_{n+1} = g(x_n)$ , where

$$g(x) = \frac{3x^3 - 8x^2 + 3x}{4x^2 - 12x + 6},$$

has a fixed point at  $x = 1$ .

**2a** (4 points). Determine whether convergence is linear or superlinear in the neighborhood of  $x = 1$ , and if linear give the rate of convergence.

**2b** (4 points). What are the other fixed points of the iteration?

**2c** (2 points). Does the iteration  $x_{n+1} = g(x_n)$  converge to  $x = 1$  regardless of the initial approximation  $x_0$ ?

**Problem 3** (25 points total).

Let  $p(x) = a_0x^n + a_1x^{n-1} + \dots + a_n$  be a polynomial with real coefficients and with  $n$  distinct roots  $(r_1, r_2, \dots, r_n)$ . It is known that the general solution to the linear recurrence equation

$$a_0x_k + a_1x_{k-1} + \dots + a_nx_{k-n} = 0 \quad \text{for } k \geq n$$

has the form

$$x_k = c_1r_1^k + c_2r_2^k + \dots + c_nr_n^k \quad \text{for } k \geq 0$$

where the coefficients  $(c_1, c_2, \dots, c_n)$  are linear combinations of the initial values  $(x_0, x_1, \dots, x_{n-1})$ .

**3a** (5 points). Show that if  $|r_1| > |r_i|$  for  $2 \leq i \leq n$ , and if we define  $\rho_k = x_{k+1}/x_k$  where the sequence  $\{x_k\}$  satisfies the stated linear recurrence, then

$$\lim_{k \rightarrow \infty} \rho_k = r_1$$

for almost all choices of initial values  $(x_0, x_1, \dots, x_{n-1})$ .

**3b** (5 points). The result in (3a) can be used to approximate  $r_1$ , by starting with random initial values and computing  $x_k$  and  $\rho_k$  for larger and larger values of  $k$ . If this method is used, will it be necessary to rescale the iterates  $x_k$  every few steps? Why or why not?

**3c** (10 points). Consider the special case  $p(x) = (x - 2)(x - 3)$  and  $x_0 = 2, x_1 = 5$ . Approximately how many iterations are required before we have

$$\left| \frac{\rho_k - 3}{3} \right| < 10^{-5}?$$

(Don't give a numerical answer; simply state a formula—possibly involving logarithms—from which the desired number of iterations could readily be calculated on a pocket calculator.)

**3d** (5 points). In practice, this method (called Bernoulli's method) is not used by itself, but only to generate an initial guess for another method, such as Newton's method. Why is Bernoulli's method inappropriate for the computation of highly accurate results?

**Problem 1** (5 points).

For each of the two operator precedence schemes below, draw a tree &presentation of the expression

$$A * B > C - D \uparrow - E / F + G$$

**1a.** Use the following operator hierarchy, and left-to-right associativity:

unary - (highest)  
binary  $\uparrow$   
\* /  
+ binary -  
< = > (lowest)

**1b.** Give all operators equal precedence, and associate from right to left.

**Problem 2** (3 points).

Why is it comparatively easier to implement dynamic rather than lexical (static) scoping with an interpreted language? Why is it comparatively easier to implement lexical rather than dynamic scoping with a compiled language?

**Problem 3** (8 points).

Consider the following program fragment:

```
var s: array[1..3] of char;  
var i, j: integer;  
procedure P(x: integer; y: char);  
  var j: integer;  
  begin  
    j := 2;  
    x := x + 1;  
    output(y);  
    output(i);  
  end;  
  
s[1] := 'A'; s[2] := 'B'; s[3] := 'C';  
i := 0; j := 1;  
P(i, s[i + j]);  
output(i);
```

What three values will be output if the procedure parameters are transmitted by the following conventions?

- 3a.** Call by value.
- 3b.** Call by value-result.
- 3c.** Call by reference.
- 3d.** Call by name.

*Software Systems*

**Problem 4** (2 points).

What is the difference between a macro call and a procedure call?

**Problem 5** (2 points).

Why is lexical analysis usually separated from the rest of the parsing process? What is a traditional mechanism used to implement lexical analysis?

**Problem 6** (5 points).

Briefly explain the most notable difference between a recursive descent parser and an SLR or LALR parser, and explain the (relatively minor) difference between SLR and LALR parsers. Are there context free grammars that can be parsed with the recursive **descent** method, but not with SLR or LALR, or vice versa?

**Problem 7** (2 points).

How might scope be represented in a symbol table? (One method is sufficient.)

**Problem 8** (5 points).

Assume you have written a compiler that uses a run-time *display*, separate from the activation stack. Give a quick sketch of the state of the stack and the display when control reaches the positions of the two comments in the following program. Assume that all parameters are called by value.

```
program main;
  procedure P(a: integer);
    procedure Q(b: integer);
      begin {8a. What are the stack and display now?}
        R(a + 2, b + 3);
      end;
    begin Q(a + 1);
    end;
  procedure R(c, d: integer);
    begin {8b. What are the stack and display now?}
    end;
begin P(1);-
end.
```

**Problem 9** (2 points).

Explain the difference between preemptive and non-preemptive processor scheduling. Which would be more appropriate for a timesharing system such as LOTS, if you were the manager of the LOTS facility? Why?

**Problem 10** (2 points).

When operating system designers speak of memory management, what do they mean by the terms external and internal fragmentation?

*Software Systems*

**Problem 11** (1 point).

How does a physical address differ from a logical (virtual) address?

**Problem 12** (8 points).

Consider the following *page reference string*:

**1, 2, 3, 4, 4, 2, 1, 4, 1, 3, 4.**

Determine \*how many page faults will occur for each of the following page replacement algorithms, assuming a memory capable of holding 2 page frames.

**12a.** Least recently used (LRU).

**12b.** First in, first out (FIFO).

**12c.** An optimal policy (OPT).

**12d.** Solve parts a, b, and c when there is only **1** page frame.

**12e.** Solve parts a, b, and c when there are **4 page frames**.

**Problem 13** (2 points).

What is the difference between *deadlock* and *starvation*?

**Problem 14** (5 points).

Consider a system containing 4 interchangeable resources that are being shared by 3 processes, each of which needs at most 2 resources. Prove that this system is deadlock free, or give a scenario in which deadlock can occur.

**Problem 15** (2 points).

If Dijkstra's *P* and *V* operations are not executed atomically, is mutual exclusion in danger? Why or why not?

**Problem 16** (6 points).

Explain how to implement *counting semaphores* **when** the only synchronization primitives available to you are *binary semaphores*.

**Problem 1 (10 points).**

Sketch a proof of the sentence

$$\exists y \forall x (A(y) \supset A(x)).$$

**Problem 2 (15 points total).**

The language  $L_0 = \{ a^n b^n c^n \mid n \geq 0 \}$  is not context free.

**2a (5 points).** Is the language  $L_1 = \{ a^n a^n b^n \mid n \geq 0 \}$  context free? Prove your claim.

**2b (5 points).** Is the language  $L_2 = \{ a^n b^n a^n \mid n \geq 0 \}$  context free? Prove your claim.

**2c (5 points).** Is the language  $L_3 = \{ a^n a^n a^n \mid n \geq 0 \}$  regular? Prove your claim.

**Problem 3 (10 points).**

Give an example of a recursively enumerable set that is not recursive. Indicate the reasons for non-recursiveness and for recursive enumerability.

**Problem 4 (15 points total).**

**4a (6 points).** Give definitions of Lisp functions  $alt[u]$  and  $double[u]$ , assuming that the argument  $u$  is a list, where  $alt[u]$  extracts alternate elements of  $u$  starting with the first, and where  $double[u]$  duplicates every element. Thus, for example,

$$\begin{aligned} alt[NIL] &= double[NIL] = NIL; \\ alt[(A B C D E)] &= (A C E); \\ double[(A B C)] &= (A A B B C C). \end{aligned}$$

**4b (9 points).** Prove by list induction that  $\forall u (alt [double[u]] = u)$ , where  $u$  ranges over all lists, using your definitions in (4a). State explicitly the induction schema that you are using.

**Problem 5 (10 points).**

A Post system  $S$  over an alphabet  $\Sigma$  consists of a set of  $k$  ordered pairs  $(\alpha_i, \beta_i)$  of words over  $\Sigma$ ; that is,

$$S = \{(\alpha_1, \beta_1), \dots, (\alpha_k, \beta_k)\}.$$

A solution to a Post system is a sequence of integers  $i_1 i_2 \dots i_m$  (where  $m \geq 1$  and  $1 \leq i_j \leq k$ ) such that

$$\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_m} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_m}.$$

It is known that the problem of finding a solution to a Post system over the alphabet  $\Sigma = \{a, b\}$  is unsolvable.

Is the corresponding problem solvable over a one-letter alphabet  $\Sigma = \{u\}$ ? Prove your claim.

## Winter 1984 Comprehensive Solutions: Algorithms and Data Structures

- 1a.** True;  $\max(f(n), g(n)) \leq f(n) + g(n)$  for all  $n$ .
- 1b.** True;  $f(n) + g(n) \leq 2 \max(f(n), g(n))$  for all  $n$ .
- 1c.** True: If  $|x_n| \leq M_0|f(n)|$  for  $n \geq n_0$  and  $|y_n| \leq M_1|g(n)|$  for  $n \geq n_1$ , then  $|x_n + y_n| \leq (M_0 + M_1) \max(f(n), g(n))$  for  $n \geq \max(n_0, n_1)$ .
- 1d.** False; e.g., let  $f(n) = 0$  and  $g(n) = n$ .
- 1e.** True;  $f(n) + g(n) \geq 2 \min(f(n), g(n))$  for all  $n$ .
2.  $\frac{1}{n+1} + \frac{1}{n+2} + \dots + \frac{1}{2n} = H_{2n} - H_n = \ln 2n + \gamma + \frac{1}{4n} - \ln n - \gamma - \frac{1}{2n} + O(n^{-2}) = \ln 2 - \frac{1}{4n} + O(n^{-2})$ . Hence the result is

$$e^{\ln 2} e^{-1/(4n)} e^{O(n^{-2})} = 2 \left(1 - \frac{1}{4n} + O(n^{-2})\right) (1 + O(n^{-2})) = 2 - \frac{1}{2n} + O(n^{-2}).$$

3. It is easy to compute all  $N$ -tuples  $(a_1, \dots, a_N)$  of integers  $1 \leq a_k \leq N$  nondeterministically. For every such  $N$ -tuple it takes polynomial time to verify the  $O(N^2 + M)$  conditions  $a_i \neq a_j$ ,  $|a_i - a_j| \neq |i - j|$ , and  $(k, a_k) \notin S$ . Hence the problem is in NP.
4.  $f(0) = 0$ ; and for  $n > 0$ ,  $f(n) = 1 + \frac{1}{n}(f(0) + f(1) + \dots + f(n-1))$ , since each of the values  $(0, \dots, n-1)$  occurs with probability  $\frac{1}{n}$ . Incidentally,  $f(n) = H_n$ .
- 5a.** The relation  $accumulator = q \cdot divisor$  is invariant at the beginning of the **while** loop.
- 5b.** On the final iteration,  $divisor = 1$ , so  $accumulator$  is reduced to zero. Incidentally, since  $accumulator$  has changed only by subtracting  $x[n], \dots, x[1]$ , this proves that  $x[1] + \dots + x[n] = m$ .
- 5c. The relation  $accumulator = q \cdot divisor + r$  will hold at the beginning of the **while** loop until  $divisor = r$ , after which time  $accumulator = (q + 1) \cdot divisor$  as in part (a). Hence  $x[i] = q + 1$  for  $1 \leq i \leq r$  and  $x[i] = q$  for  $r < i \leq n$ .

The program can be regarded as a recursive method for partitioning  $m$  things into  $n$  parts that are as equal as possible: It sets the first part equal to  $\lfloor m/n \rfloor$ , then divides the remaining  $m - \lfloor m/n \rfloor$  things into  $n - 1$  parts by the same technique.

6. This is a thinly disguised shortest path problem on a graph with  $m = \frac{3}{2}n$  edges. Dijkstra's well known algorithm [AHU, pp. 203ff] can be used to find any particular row of the matrix in  $O(n \log n)$  time, since it involves  $O(m)$  operations of summing and comparing costs, plus  $O(n)$  operations of inserting and deleting nodes of a priority queue; each of the priority queue operations takes  $O(\log n)$  time. Therefore all  $n$  rows of  $C$  can be computed in  $O(n^2 \log n)$  time.

**Winter 1984 Comprehensive Solutions: Artificial Intelligence**

**Problem 1.**

- (a)  $\forall x. (\text{person}(x) \wedge (\forall y. (\text{dog}(y) \vee \text{child}(y)) \supset \text{hates}(x, y)) \supset \sim \text{all bad}(x)).$
- (b)  $\exists x. \text{dog}(x) \wedge \text{gentle}(x) \wedge \text{well trained}(x).$
- (c)  $\forall x. \text{dog}(x) \wedge \text{gentle}(x) \supset \text{well trained}(x).$
- (d)  $\forall x. (\text{person}(x) \wedge (\exists y. \text{dog}(y) \wedge \text{owns}(x, y) \wedge \exists z. (\text{visits}(z, x) \wedge \sim \text{barks at}(y, z)))) \supset \text{lucky}(x)).$
- (e)  $\forall x. (\text{person}(x) \wedge (\exists y. \text{child}(y) \wedge \text{barks at}(x, y)) \supset \sim \text{well trained}(x)).$

**Problem 2.** Representing the information as wffs we have

$$\begin{aligned}
 &S(\text{Knuth}) \supset T(\text{Floyd}) \\
 &(S(\text{Knuth}) \vee S(\text{Reid})) \wedge \sim (S(\text{Knuth}) \wedge S(\text{Reid})) \\
 &\sim (T(\text{Reid}) \wedge T(\text{Floyd})) \\
 &\forall x. T(x) \vee S(x)
 \end{aligned}$$

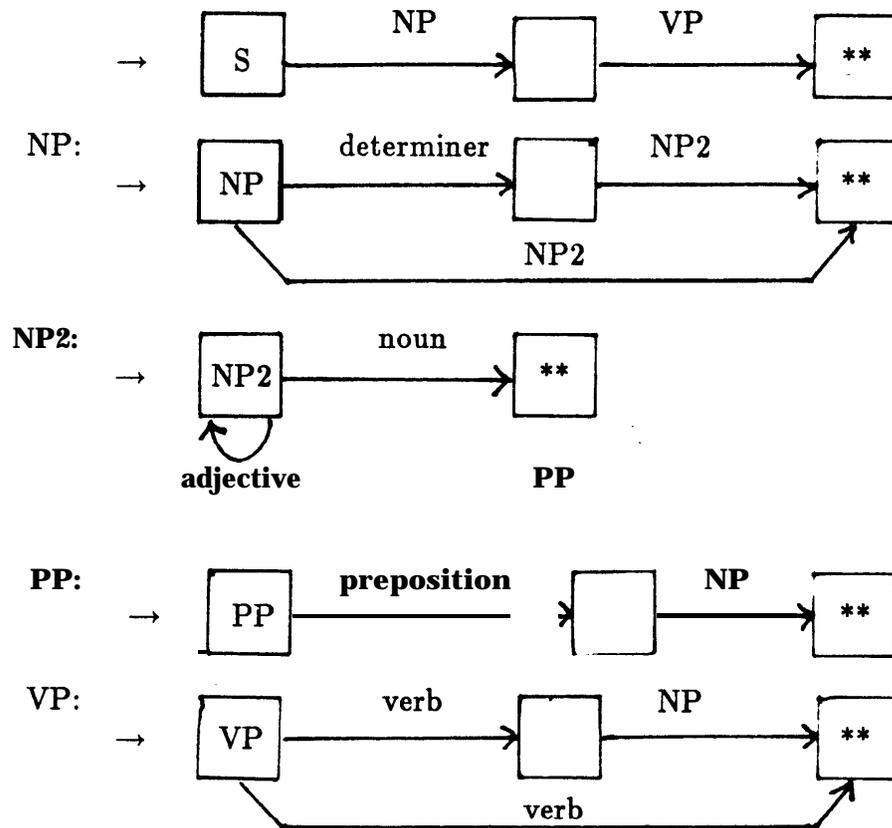
To these we add the negation of the goal  $S(\text{Reid})$  and convert to CNF. The clauses are

$\sim S(\text{Knuth}) \vee T(\text{Floyd})$	. . . <b>1</b>
$S(\text{Knuth}) \vee S(\text{Reid})$	. . . <b>2</b>
$\sim S(\text{Knuth}) \vee \sim S(\text{Reid})$	. . . <b>3</b>
$\sim T(\text{Reid}) \vee \sim T(\text{Floyd})$	. . . <b>4</b>
$S(x) \vee T(x)$	. . . <b>5</b>
$\sim S(\text{Reid})$	. . . <b>6</b>

Resolving 5, 6 we get	$T(\text{Reid})$	. . .	<b>7</b>
<b>7, 4</b>	$\sim T(\text{Floyd})$	. . .	<b>8</b>
<b>8, 1</b>	$\sim S(\text{Knuth})$	. . .	<b>9</b>
<b>9, 2</b>	$S(\text{Reid})$	. . .	<b>10</b>
<b>10, 6</b>	$\square$	empty clause	

**QED**

**Problem 3. (a)**



- (b) While parsing the **NP** use a register to store the number of the noun. Attach to the verb arc a test for agreement of number.

**Problem 4.**

**(a) Initialize**

$$G = \{(x y)\}$$

$$S = \{(\text{red cube})\} \text{ after first training instance}$$

The second training instance (blue pyramid) forces the G set to be specialized so as not to accept this negative example. Now we have

$$G = \{(x \text{ cube})(\text{red } y)\}$$

$$S = \{(\text{red cube})\}.$$

Note that the possible specialization ( $x$  sphere) is not considered as it has been removed from version space during the previous training instance.

At the next positive example: (blue cube). Update  $S$  prunes  $G$  to eliminate (red  $y$ ), since it does not cover (blue cube). Then  $S$  is generalized.

$$G = \{(x \text{ cube})\}$$

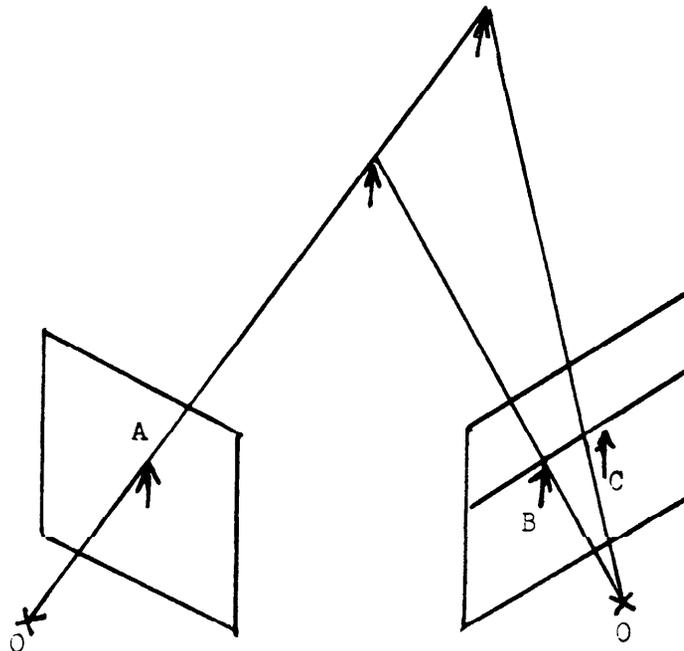
$$S = \{(x \text{ cube})\}.$$

Since  $G = S$  the algorithm halts having learned the concept ( $n$  cube).

- (b) A disjunctive concept {Either red or a cube} cannot be learned.

**Problem 5.**

- (a)



The object responsible for feature  $A$  in the left image lies along a ray starting at the origin of left coordinate system,  $O$ , and proceeding through the feature  $A$ . The image of the object in the right picture must lie along the projection of the ray  $OA$  on to the right image plane which is the line  $BC$ . This line is called an epipolar line. The search for a match with the corresponding feature is thus restricted to be a one-dimensional search along the epipolar line.

- (b) : 505 times.  
 (c) Since there are fewer edges than pixels, finding the best match for edges takes much less time. Area correlation then can be constrained to regions between corresponding edges.

**Problem 8.**

- (a) In classical logic, addition of new facts results in new inferences but the old inferences remain valid. Nonmonotonic reasoning permits us to retract past inferences in the presence of new evidence. Knowing that 'All birds fly' and 'Tweety is a bird' we infer that 'Tweety can fly'. If, in addition, we come to know that Tweety is an ostrich we can now infer that 'Tweety can't fly'.

Winter 1984 Comprehensive Solutions: Artificial Intelligence

- (b) In a formalism for representing facts about the world, it is the problem of specifying what should change and what should not as a consequence of an action.
- (c) We assume that facts not explicitly changed by the action have remained unchanged.
- (c)
  - (i) Use the first rule.
  - (ii) Use the most recently fired rule.
  - (iii) Use the rule with maximum number of antecedent clauses.

## Winter 1984 Comprehensive Solutions: Hardware

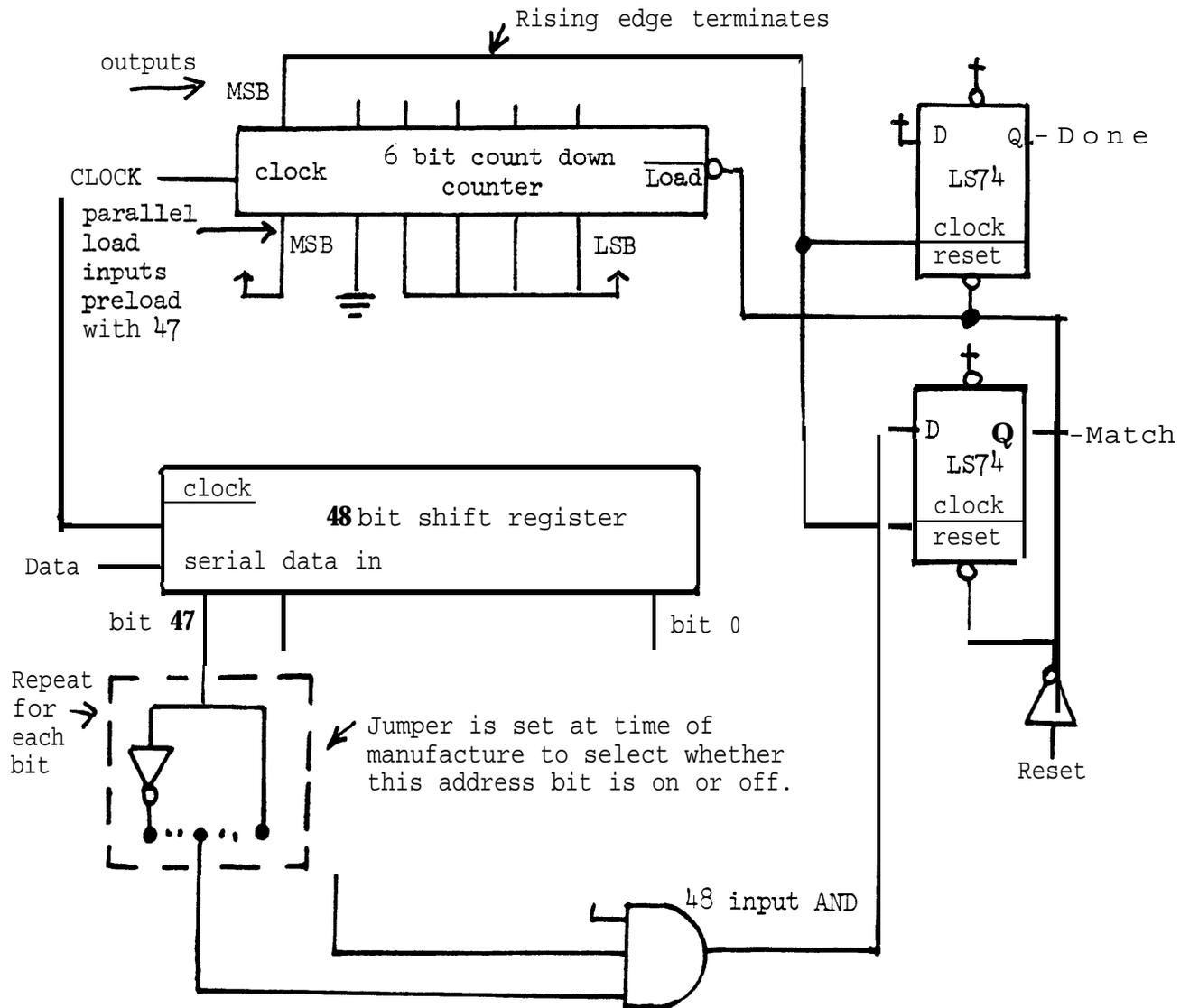
### Problem 1.

At least three architectures are possible:

- (i) The incoming data is shifted into a serial-in/parallel-out shift register. When all **48** bits have come in then the contents of the shift register is compared with the required address and the result is pronounced.
- (ii) The required address is preloaded into a parallel-in/serial-out shift register. Then each bit of the incoming data is sequentially compared with the incoming data stream. Initially it is assumed that the incoming address is OK. (An OK flag is set to true.) As each bit arrives, if there is a mismatch between the incoming data and the serial output of the shift register the OK flag is reset. When all **48** bits have arrived, the OK flag indicates the correct answer.
- (iii) A finite state machine is set up to recognize a particular address. The machine advances once for each bit received. After **64** cycles it ends up in one of two states (match or mismatch).

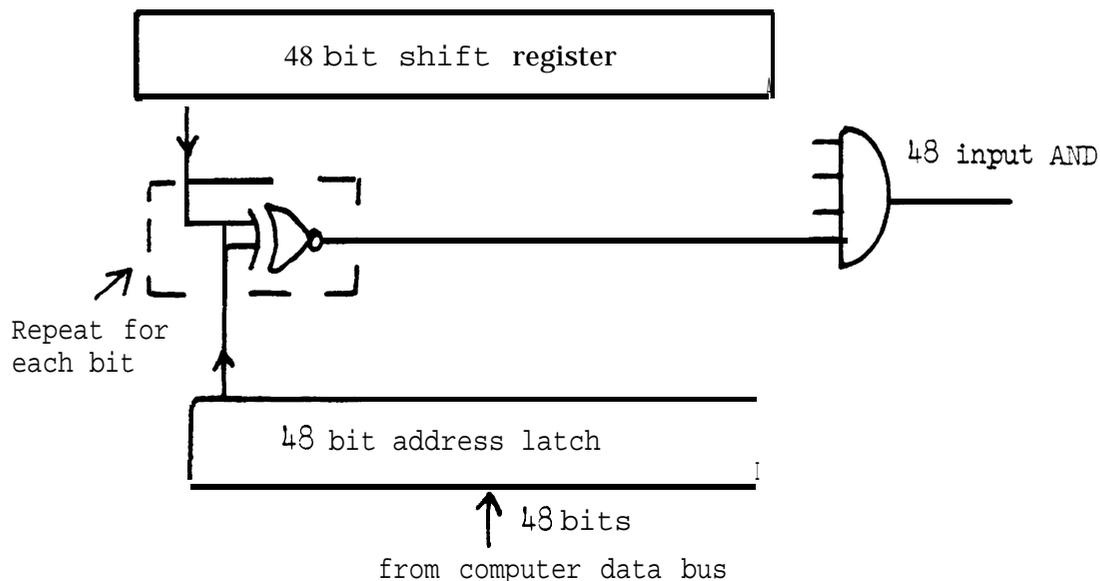
For the rest of this discussion, architecture (i) is used.

1a.



**1b.** Change the jumpers to solid wires and omit unused inverters.

**1c.** In this case, we must load the desired address into a latch and use a general equality comparator as shown:



Problem 2.

**2a.** Very large numbers manufactured, very low power, very low speed, cost is important. These devices are usually fully custom IIL or CMOS.

**2b.** Small numbers manufactured, very low power, low speed, high reliability, cost not critical. IIL or CMOS gate arrays are good choices here. Some commercial products are built using off-the-shelf devices in chip form (these are older designs).

**2c.** Moderate volume (a few thousand), fairly high speed. Off-the-shelf TTL is quite common while newer designs use TTL or NMOS gate arrays. Some use can be made of the standard NMOS display controller chips, but these usually don't have all of the features needed. Really high performance systems will have some ECL in them.

**2d.** A few hundred of each design to be built, high speed important. The LM3600 Lisp Machine uses off-the-shelf ECL. A good case can be made that modern NMOS (or state-of-the-art CMOS) gate arrays would yield good enough performance at lower cost.

**2e.** Low power, very high reliability, radiation resistant, moderate speed. A good choice is a mix of CMOS gate arrays and off-the-shelf CMOS.

Winter 1984 Comprehensive Solutions: Hardware  
**Problem 3.**

Some advantages of overlaying: greatly expands the size of programs that a system can run without specialized hardware support; faster memory access since no address translation is needed; requires little or no support in the operating system; quite efficient if the user knows the program characteristics well; can assure predictable realtime response; I/O operations are efficient since the whole overlay is read/written at one time.

Some disadvantages of overlaying: does not work when large amounts of data must be available at the same time; places a significant additional burden on the user to deduce and specify the overlay structure; very inefficient if the user blunders in the overlay specifications; more total I/O may result even in a good overlay structure since an entire overlay is read even if only a small part will be referenced this pass; memory is not efficiently shared among multiple users; programs that are overlaid cannot easily adapt to varying amounts of available space.

Problem 4.'

**4a.**  $T_{\min} > T_{\max} + T_{\text{get}}$ . Some computers that were otherwise good at realtime work have been rendered unsuitable by having a few instructions that took a very long time. Most interrupt systems allow at least one non-interrupt instruction to execute between interrupts. (Character move or floating point instructions are common worst-case instructions.)

**4b.** During  $T_{\text{avg}}$  we must use at most  $(P/100)T_{\text{avg}}$  CPU time to service the interrupt. As a result, we must have  $T_{\text{get}} \leq (P/100)T_{\text{avg}}$ . Thus

$$T_{\text{avg}} \geq \frac{100}{P} T_{\text{get}} .$$

**4c.** In order to be able to service all clock interrupts, we must have at least

$$T_{\text{clk}} > T_{\text{time}} + T_{\text{imax}}$$

as in 4a.

The clock interrupt leaves time gaps of size  $(T_{\text{clk}} - T_{\text{time}})$  during which the character interrupt routine can execute. Thus, the character routine must execute in  $K = \lceil T_{\text{get}} / (T_{\text{clk}} - T_{\text{time}}) \rceil$  time gaps. So, the total time elapsed from a character interrupt



Winter 1984 Comprehensive Solutions: Hardware  
 The transmission time is

$$T_{stcpu} + T_{input} + N(T_{send} + T_{imax} + T_{input}) .$$

Thus, the maximum rate is

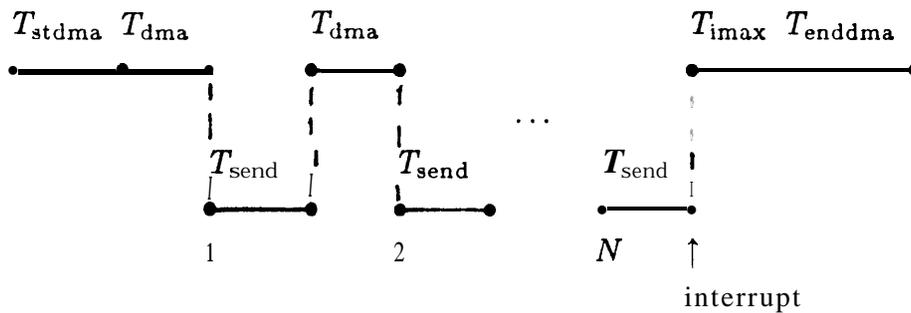
$$\text{CPU RATE} = \frac{N}{T_{stcpu} + T_{input} + N(T_{send} + T_{imax} + T_{input})}$$

which takes

$$100 \left[ \frac{T_{stcpu} + N(T_{put} + T_{input}) + T_{input}}{T_{stcpu} + T_{input} + N(T_{send} + T_{imax} + T_{input})} \right]$$

percent of CPU time.

5b.



As shown above, the transmission time required to send  $N$  characters is  $T_{stdma} + N(T_{dma} + T_{send}) + T_{imax} + T_{enddma}$ . This requires  $T_{stdma} + N(T_{dma}) + T_{enddma}$  of CPU time. Thus the maximum rate is

$$\text{DMA RATE} = \frac{N}{T_{stdma} + N(T_{dma} + T_{send}) + T_{imax} + T_{enddma}}$$

which requires

$$100 \left[ \frac{T_{stdma} + N(T_{dma}) + T_{enddma}}{T_{stdma} + N(T_{dma} + T_{send}) + T_{imax} + T_{enddma}} \right]$$

percent of CPU time.

5c. Let  $N \rightarrow 1$  in 5(a) case (ii) and 5(b).

$$\text{DMA RATE} = \frac{1}{T_{stdma} + T_{dma} + T_{enddma} + T_{send} + T_{imax}}$$

$$\text{CPU RATE} = \frac{1}{T_{stcpu} + 2T_{input} + T_{send} + T_{imax}}$$

Winter 1984 Comprehensive Solutions: Hardware  
 In typical systems  $T_{\text{stepu}} + 2T_{\text{input}} \ll T_{\text{stdma}} + T_{\text{enddma}}$ . Thus, CPU interrupt driven output is superior for small  $N$ .

**5d.** Let  $N \rightarrow \infty$  in 5(a) and 5(b).

$$\text{DMA RATE} = \frac{1}{T_{\text{dma}} + T_{\text{send}}}$$

$$\text{CPU RATE} = \frac{1}{T_{\text{imax}} + T_{\text{input}} + T_{\text{send}}}.$$

In typical systems  $T_{\text{dma}} \ll T_{\text{input}}$ . Thus DMA driven output is superior for large  $N$ .

**5e.** If the computer will be sending mostly large packets, then we need only implement DMA output. If the computer will be sending mostly small packets, then we need only implement CPU interrupt driven output. If it is not known, we must implement both and choose one of the methods based on the size of each outgoing packet.

The exact point of crossover must be determined by examining where the CPU overhead is lowest and/or where the transmission rate is highest.

## Winter 1984 Comprehensive Solutions: Numerical Analysis

### Problem 1.

- a. True.
- b. False. Consider  $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ .
- c. False. There are  $n \times n$  matrices for which elements grow to  $2^n$  times their original size.
- d. True.
- e. **True.**
- f. False. The term  $\|A\| \|A^{-1}\|$  must appear on the right hand side.
- g. False.
- h. True.
- i. True. One solves a well-conditioned tridiagonal system.
- j. False. Runge's function shows this.
- k. True.
- l. True.
- m. False.
- n. True. It has "infinite order".
- o. False.
- p. True.
- q. True.
- r. False. It is 3.

### Problem 2.

- a. Superlinear. If  $g'(2)$  is continuous in a neighborhood of a fixed point and  $g' = 0$  at the fixed point, convergence is superlinear. One need only check this.
- b. The fixed points are 0, 1, and 3.
- c. No. If  $x_0 = 0$ , for example, then  $x_n = 0$  for  $n = 0, 1, \dots$

**Problem 3.**

.....  $\mathbf{x}_k = c_1 r_1^k + c_2 r_2^k + \dots + c_n r_n^k$  for  $k \geq 0$  where the coefficients  $\mathbf{c} = (c_1, c_2, \dots, c_n)^T$  depend on the initial values  $\mathbf{x} = (x_0, \dots, x_{n-1})^T$ , we have that

$$(1) \quad \rho_k = \frac{x_{k+1}}{x_k} = r_1 \left( \frac{c_1 + c_2 \left(\frac{r_2}{r_1}\right)^{k+1} + \dots + c_n \left(\frac{r_n}{r_1}\right)^{k+1}}{c_1 + c_2 \left(\frac{r_2}{r_1}\right)^k + \dots + c_n \left(\frac{r_n}{r_1}\right)^k} \right).$$

The coefficients  $\mathbf{c}$  must satisfy the system of equations

$$\begin{aligned} x_0 &= c_1 + c_2 + \dots + c_n \\ x_1 &= c_1 r_1 + c_2 r_2 + \dots + c_n r_n \end{aligned}$$

$$x_{n-1} = c_1 r_1^{n-1} + c_2 r_2^{n-1} + \dots + c_n r_n^{n-1}$$

or

$$\mathbf{x} = \mathbf{V} \mathbf{c}$$

where  $V$  is the nonsingular Vandermonde matrix

$$V = \begin{pmatrix} \mathbf{1} & 1 & \dots & 1 \\ r_1 & r_2 & \dots & r_n \\ r_1^{n-1} & r_2^{n-1} & \dots & r_n^{n-1} \end{pmatrix}.$$

Thus  $\mathbf{c} = V^{-1} \mathbf{x}$ . The set  $\{\mathbf{c} \mid c_1 = 0\}$  is a one-dimensional subspace of  $R^n$ . Its image under  $V$  is, likewise, a one-dimensional subspace of  $R^n$ . Thus, unless  $\mathbf{x}$  is in this subspace,  $c_1 \neq 0$ .

If  $c_1 \neq 0$ , the term in parentheses in (1) converges to 1, as  $k \rightarrow \infty$ , since  $|r_j/r_1| < 1$  for  $j = 2, 3, \dots, n$ . Thus, when  $c_1 \neq 0$ ,

$$\lim_{k \rightarrow \infty} \rho_k = r_1.$$

- b. If  $|r_1| > 1$  the iterates  $\mathbf{x}_k$  grow exponentially. Overflow may occur unless they are **rescaled**.

If  $|r_1| < 1$  the iterates decay to zero exponentially, and underflow may occur unless we **rescale**.

Rounding error and loss of significance have nothing to do with it.

c. Since

$$x_0 = 2 = c_1 + c_2$$

$$x_2 = 5 = 2c_1 + 3c_2$$

we have  $c_1 = c_2 = 1$ . Thus

$$\rho_k = \frac{2^{k+1} + 3^{k+1}}{2^k + 3^k},$$

and

$$\left| \frac{\rho_k - 3}{3} \right| = \frac{2^k}{3(2^k + 3^k)}.$$

Thus  $|(\rho_k - 3)/3| < 10^{-5}$  when  $(\frac{3}{2})^k > \frac{1}{3} \times 10^5$ ; this happens for  $k > \ln 10^5 - \ln 3 / \ln(3/2) \doteq 25.7$ .

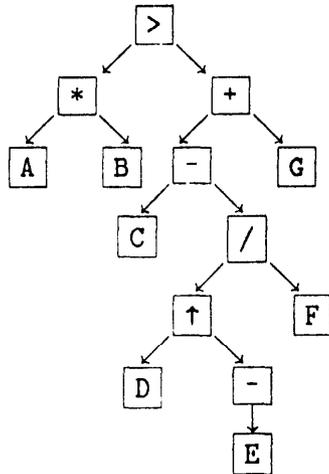
Thus 26 iterations suffice.

- d. It converges linearly. When  $|r_2/r_1|$  is close to 1, convergence is too slow. Newton's method is much faster once a good initial guess is available.

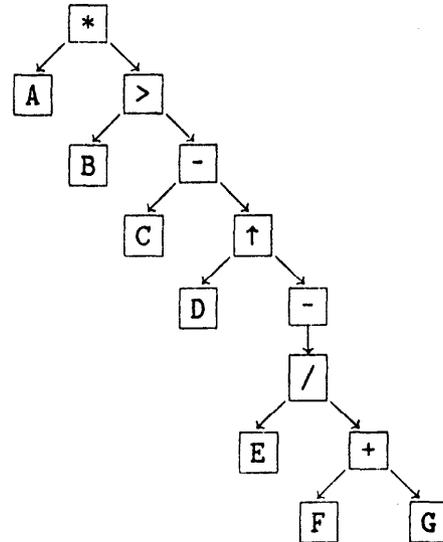
## Winter 1984 Comprehensive Solutions: Software Systems

### Problem 1.

a.



b.



### Problem 2.

Basically, static scoping is oriented to binding identifiers to objects depending on syntactic hierarchies (procedure *definition* nesting), which is efficiently done at compile-time.

Dynamic scoping binds identifiers depending on semantic hierarchies (procedure *invocation* nesting), which depends on the dynamic, i.e. run-time, behavior of the program. Since most of the work must be deferred to run-time, interpretation is natural.

With dynamic scoping, the run-times have to either search through the stack, or do extra hook-keeping at procedure entry and exit times. The system must maintain a symbol table. This is normally needed with interpretation in any case, so there is no extra overhead.

### Problem 3.

a: Call by value.

The arguments of  $P$  evaluate to 0 and 'A'. Then the local  $x$  is incremented to 1, but this has no effect on the global  $i$ , which is still 0. Answer: 'A', 0, 0.

b: Call by value-result.

Again, the parameters evaluate to 0 and 'A'. The local  $x$  again is incremented to 1.  $y = 'A'$  and  $i$  (which is still 0) are printed, and then  $i$  is updated to 1. Answer: 'A', 0, 1.

**c: Call by reference.**

The arguments evaluate to *pointers to i* and  $s[1]$ , which have initial values 0 and 'A'. Now  $x$  (which really refers to  $i$ ) is incremented, so  $x = i$  becomes 1. **Answer: 'A', 1, 1.**

**d: Call by name.**

This part very few people got right. The parameters are evaluated into *thunks*, which basically are parameter-less procedures, which when called evaluate to the argument expression. (It may help to think of the argument as being a short-hand for a lambda-function.)

Thus the arguments evaluate to  $\lambda.i$  and  $\lambda.s[i + j]$ , **But note** that the variables  $i$ ,  $j$  and  $s$  are bound to the values in the *calling* global context. Thus the  $j$  in  $\lambda.s[i + j]$  refers to the global  $j$ , and not to the one defined in  $P$  (which is just a red herring). (This (and that, you can do recursion) is the semantic difference between call by name and macros (see next problem).) Inside  $P$ ,  $j$  is set to 2, and  $x$ , which evaluates to  $i$ , is set to 1. Now  $y$  is evaluated and output. That is, we evaluate  $\lambda.s[i + j]$ .  $i$  has been changed to 1, but  $j$  refers to the global  $j$ , which is still 1. Thus we print  $s[1 + 1]$ , which **is 'B'**. **Answer: 'B', 1, 1.**

**Problem 4.**

With a macro call, there is inline textual substitution of the source at compile/parse time. There will be one instance of code generated for each call. Parameter binding is done textually at compile-time.

A procedure call is done at run-time. In the code generated, there is only one instance of the procedure and all procedure calls cause a branch to it at run-time.

**Problem 5.**

Lexical analysis is the process of breaking the input stream of characters into a stream of tokens, which is then fed to the parser. The main reason for having a separate lexical analyzer is one of efficiency. A significant part of the execution time of a compiler is spent in lexical analysis of the input characters, since there normally are so many of them. Therefore it is important that this part of the compiler is as fast as possible, and a parser for a context-free grammar is overly general.

Separating the stages also has advantages in terms of modularity, and making the tables smaller.

**Problem 6.**

A recursive descent (RD) parser is top-down, and generates a left-most derivation. It is normally program-driven, in that it is implemented as a set of mutually recursive procedures, (in principle) one for each syntactic non-terminal. LL can be table driven, in

which case it is called an LL parser. Its major virtue is that it is simple to implement and understand.

A LR parser is table-driven and bottom-up. It scans input Left-to-right, generating a Right-most-derivation in reverse. These parsers are efficient, general and detect errors early. SLR (Simple LR) and LALR (Look-Ahead LR) refer to two different methods of generating the tables. Though they have the same number of states, LALR tables take into account a one-item look-ahead when constructing the tables.

LALR is a more general class of parsers than SLR, which is a more general class than RD. For example, left-recursive grammars cannot be implemented by RD parsers.

**Problem 7.**

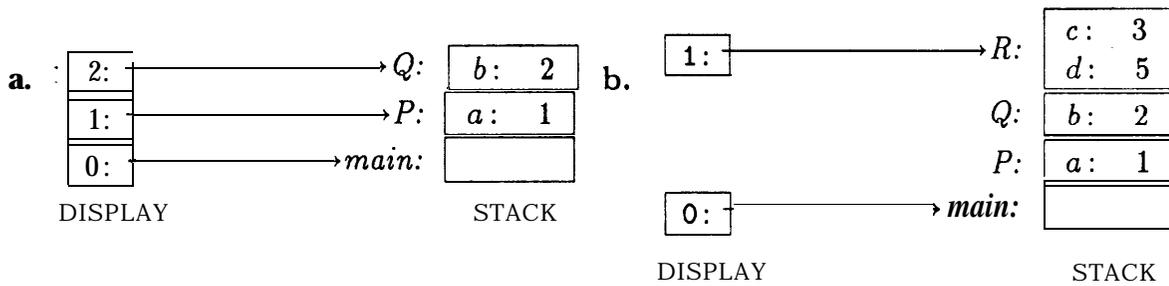
Give a unique block-number (or a relative level-number) to all blocks. Each identifier is determined by the pair of its name and the number of the block where it is defined. When an identifier is referenced, the compiler must determine its block-number by searching through the symbol-table (from innermost block to outermost surrounding block) For a match. The symbol-table data structure (e.g. hash table) must be designed to make this efficient.

**Problem 8.**

The *stack* is a list of stack frames (activation records). Each frame contains the parameters and local variables for one procedure invocation. It also contains the return address, and other control information which is not shown here.

The *display* is a mapping from (lexical) level numbers to stack frames. It is really an auxiliary stack, but since the number of nested procedures will normally be small, we can decide on some small (4- 16) maximum, and preallocate the stack in a fixed location. (It is easy to check at compile time that this number is not exceeded.)

Knowing the level number of an identifier (see previous problem), and the offset of a-variable in its stack frame, it is fast to access a non-local variable, even with nested, recursive procedures.



**Problem 9.**

Once a process under non-preemptive scheduling gets a processor, it runs until it decides that it no longer needs it. **A** process will typically release the processor when waiting for I/O, or the programmer might carefully add calls to the scheduler at suitable intervals.

Under preemptive scheduling, a process can be interrupted in the middle of doing simple computation without even calling on the run-time system, because of some external event, usually because some process with higher priority becomes ready.

Note that priorities may change dynamically, depending on things such as previous consumption of CPU. Getting yanked off because you've used up your time slice is an example of preemption.

With this clarification, it should be clear that LOTS (and any time-sharing system) *must* use preemptive scheduling to stop run-away processes. Single-user systems, special-purpose systems, and batch systems (if the operator is awake) can get away with using non-preemptive scheduling.

**Problem 10.**

External fragmentation refers to the memory wasted *between* allocated chunks. **As** items are allocated and released, there will be holes in memory between items. Many of these will become too small to be useful, so memory is wasted there. Some systems will do compaction when needed, other systems just assume that the problem will not make **a** critical difference. This problem is basically associated with segmentation systems.

Internal fragmentation refers to the fact that memory is wasted *inside* allocated chunks, normally because it is **allocated** in fixed-size chunks (pages, blocks). Therefore there will be round-off error, and **on average** there will be wasted  $\frac{1}{2}$  chunks per "item," unless care is taken when setting item sizes. This problem is characteristic of paging systems.

**Problem 11.**

**A** physical address is absolute and **refers** to some specific piece of hardware, like **a** specific memory cell, or disk address. A logical (virtual) address is relative to some user or process. There is a mapping between the virtual address and the physical address, and the mapping is implemented at least partially in software. (Occasionally, there may be multiple layers of logical addresses and mappings.)

**Problem 12.**

Note that it is assumed that memory is initially empty, so the first page reference generates a page fault. Some people did not count page faults when there were available

Winter 1984 Comprehensive Solutions: Software Systems

free page frames (i.e. until memory was filled). Though this is perhaps less logical, it was accepted. Correct answers under this assumption are calculated by subtracting the memory size from the answers given below.

**a.** Least recently used: Throw out the page whose most recent reference was furthest back in time, on the assumption that this is likely to be an “obsolete” page. **Answer: 9.**

**b.** First in, first out: Throw out the page which was loaded furthest back in time. **Answer: 8.**

**c.** An optimal policy: Throw out the page which will be needed furthest away in the future. In general, this requires prescience, so is of largely theoretical interest. **Answer: 6.**

**d.** Since there is only one page frame, there can only be one page in memory, so there is never any choice about which page to throw out. Thus all the algorithms will be equivalent, and the number of page faults will be equal to the number of references which differ from the previous reference. **Answer: 10.**

**e.** Since the size of the memory is sufficient for all the pages referenced, there will be no page faults once the memory is initially loaded. Thus again, all the algorithms are equivalent. **Answer: 4.**

Problem 13.

Both refer to the problem of a process never getting to run, because it is never allocated some resource (such as a processor or a tape drive) that it needs.

Deadlock refers to a problem of cyclic dependencies: All processes in a cyclic chain have acquired some but not all of the resources they need to continue. They are waiting to receive the extra resources they need. But everyone is waiting for some resource that is tied up by some other process, in a vicious circle of dependencies. Therefore no one gets anywhere, and outside intervention (from a human or from the operating system) is needed.

Starvation occurs when some waiting process never gets some resource it needs because other processes always manage to grab it first. In this case, productive work is being done (by the other processes), and the starved process could theoretically get to run, but it has too low a priority and the system is too busy.

**Problem 14.**

Since the resources are all equivalent, they would have to be all allocated if a request is denied. Since there are four resources and three processes, at least one process must have been assigned two resources. But that is all it needs, so at least that one process can run. Therefore deadlock is impossible, though starvation is possible.

**Problem 15.**

$P$  and  $V$  must be atomic. Otherwise, one danger is that one process could test the semaphore, find it available, and become suspended. Then another process could test the same semaphore, also find it available (since the first process did not have time to mark it as decremented), and proceed to use some resource. Then the first process could get rescheduled, also decrement the semaphore, and proceed to use the resource, thinking it has exclusive use.

**Problem 16.**

We can implement an integer semaphore with an integer counter, and two binary semaphores (one to control mutual exclusion and one to control waiting). (Actually, the one to control mutual exclusion could be shared by all the semaphores.)

(Note on terminology: Dijkstra's  $V$  and  $P$  operations are also called *up* and *down*, as well as *Signal* and *Wait*. We here use the latter names.)

Operations and type names subscripted by  $B$  are those for binary (Boolean) semaphores, while those subscripted by  $I$  are integer semaphores.

**type**

$Sema \leftarrow B = \bullet \bullet \bullet \bullet \bullet$  {Binary semaphore}  
 $Sema \leftarrow I = \mathbf{record}$  {Integer semaphore}  
     *mutex*:  $Sema \leftarrow B$ ;                    {used for mutual exclusion}  
     *q*:  $Sema \leftarrow B$ ;                        {used for queue of waiting processes}  
     *count*: *integer*;                        {number of excess Signals}  
     { Invariant: if *count* < 0, Abs (*count*) is the number of suspended processes

waiting on *q*. }

**end;**

**procedure** *Init*  $\leftarrow I$  (**var** *s*:  $Sema \leftarrow I$ ; *v*: *integer*);                    {Initialize}

**begin**

**with s do**

**begin**

*Init*  $\leftarrow B$ (*mutex*, 1);

*Init*  $\leftarrow B$  (*q*, 0);

*count* := *v*;

**end;**

**end;**

**procedure** *Wait*  $\leftarrow I$  (**var s**:  $Sema \leftarrow I$ );                    (*Wait* = *down* = *P*)

**begin**

**with s do**

**begin**

*Wait*  $\leftarrow B$  (*mutex*);

*count* := *count* - 1;

**if** *count* < 0 **then begin** *Signal*  $\leftarrow B$  (*mutex*); *Wait*  $\leftarrow B$  (*q*) **end**

**else** *Signal*  $\leftarrow B$  (*mutex*);

**end;**

**end;**

**procedure** *Signal*  $\leftarrow I$  (**var s**:  $Sema \leftarrow I$ );                    (*Signal* = *up* = *V*)

**begin**

**with s do**

**begin**

*Wait*  $\leftarrow B$  (*mutex*);

*count* := *count* + 1;

**if** *count*  $\leq$  0 **then** *Signal*  $\leftarrow B$  (*q*);                    {Release some process waiting

*Signal*  $\leftarrow B$  (*mutex*);

**end;**

**end;**

on *q*. }

## Winter 1984 Comprehensive Solutions: Theory of Computation

### Problem 1.

**Case 1:** For some  $y_0$ ,  $A(y_0)$  is false. Then for any  $x$ ,  $A(y_0) \supset A(x)$ . Hence,  $\exists y \forall x (A(y) \supset A(x))$ .

**Case 2:** For no  $y_0$ ,  $A(y_0)$  is false. Then for all  $x$ ,  $A(x)$  is true. Therefore,  $A(y) \supset A(x)$  holds for all  $y, x$ . Hence,  $\exists y \forall x (A(y) \supset A(x))$ .

An alternative solution-proof by resolution:

Negate the formula:  $\neg[\exists y \forall x (A(y) \supset A(x))] = \forall y \exists x (A(y) \wedge \neg A(x))$

Remove quantifiers by Skolemization:  $A(y) \wedge \neg A(f(y))$

We get two clauses:  
 1.  $A(y_1)$   
 2.  $\neg A(f(y_2))$

Resolve 1 and 2 using the unifier

$y_1 \leftarrow f(y_2)$ : 3.  $\square$

We have shown that the negation of  $\exists y \forall x (A(y) \supset A(x))$  is unsatisfiable, therefore

$$\exists y \forall x (A(y) \supset A(x)) \text{ is valid.}$$

### Problem 2.

**a.**  $L_1 = \{ a^n a^n b^n \mid n \geq 0 \}$  is context free. The following is a context free grammar for  $L_1$ :

$$S \rightarrow aaSb \mid \epsilon.$$

**b.**  $L_2 = \{ a^n b^n a^n \mid n \geq 0 \}$  is not context free. We will prove this fact by contradiction; assume  $L_2$  was context free. Consider the homomorphism  $h$  defined by  $h(a) = a$ ;  $h(b) = b$ ;  $h(c) = a$ . Context free languages are closed under inverse homomorphisms, therefore  $h^{-1}(L_2) = \{ (a+c)^n b^n (a+c)^n \mid n \geq 0 \}$  would be context free. Context free languages are also closed under intersection with regular languages, and therefore  $h^{-1}(L_2) \cap a^*b^*c^* = \{ a^n b^n c^n \mid n \geq 0 \}$  would be context free, but that is  $L_0$ , a language known not to be context free. Contradiction.

**c.**  $L_3 = \{ a^n a^n a^n \mid n \geq 0 \}$  is regular. Clearly  $L_3 = (aaa)^*$ .

**Problem 3.** Let  $\{ T_m \mid m = 1, 2, \dots \}$  be a standard enumeration of all Turing machines. Then the set

$$S = \{ (n, m) \mid T_m \text{ halts on input } n \}$$

Winter 1984 Comprehensive Solutions: Theory of Computation  
 is recursively enumerable but not recursive. If  $S$  were recursive the halting problem would be decidable, and a standard dovetailing computation can be constructed to describe a Turing machine that accepts  $S$ .

**Problem 4.**

**a.**  $alt[u] \leftarrow$

**if**  $n$  **or**  $ndu$  **then**  $u$   
**else**  $au. alt[ddu]$

$double[u] \leftarrow$

**if**  $n$  **then**  $NIL$   
**else**  $au. au. double[du]$

In internal notation these definitions become:

```
(DEFUN ALT (U)
  (COND
    ( (OR (NULL U) (NULL (CDR U)) ) U)
    (T (CONS (CAR U) (ALT (CDDR U))))))
(DEFUN DOUBLE (U)
  (COND
    ( (NULL U) NIL)
    (T (CONS (CAR U) (CONS (CAR U) (DOUBLE (CDR U))))))
```

**b.** We use the induction schema

$$[\Phi(NIL) \wedge (\forall x u. \Phi(u) \supset \Phi(x.u))] \supset \forall u. \Phi(u).$$

This is the simple form of rank induction on the length of the list  $u$ , also called **LIST-INDUCTION**. Define  $\Phi(u) \equiv alt[double[u]] = u$ . By the definitions in 4a.,  $alt[NIL] = double[NIL] = NIL$ , so  $alt[double[NIL]] = NIL$ , and  $@(NIL)$  is established.

Now for the inductive step, assume that  $\Phi(u)$  holds, that is,  $alt[double[u]] = u$ . By 4a.,  $double[x.u] = x.x.double[u]$  and  $alt[x.x.double[u]] = x.alt[double[u]]$ . Therefore  $alt[double[x.u]] = x.alt[double[u]] = x.u$  by the inductive hypothesis. Thus  $\Phi(u) \supset \Phi(x.u)$  is established. Since we made no assumptions about  $x$  and  $u$ , we have  $\forall x u. \Phi(u) \supset \Phi(x.u)$ .

Now, by **LISTINDUCTION** we have  $\forall u. \Phi(u)$ , or  $\forall u. alt[double[u]] = u$ . Q.E.D.

**Problem 5.**

This problem is solvable:

In the case of a unary alphabet, each string  $\alpha$  is uniquely characterized by its length  $l(\alpha)$ . A Post system is given by a set of ordered pairs of numbers

$$S' = \{(n_1, m_1), \dots, (n_k, m_k)\},$$

where  $n_i = l(\alpha_i)$ ,  $m_i = l(\beta_i)$ .

A solution to  $S'$  is then a sequence of integers  $i_1 \dots i_s$  ( $s \geq 1$ ,  $1 \leq i_j \leq k$ ) such that

$$n_{i_1} + n_{i_2} + \dots + n_{i_s} = m_{i_1} + m_{i_2} + \dots + m_{i_s}.$$

By combining repetitions, this is equivalent to finding integers  $x_1 \dots x_k \geq 0$  such that at least one  $x_i \neq 0$  and

$$x_1 n_1 + \dots + x_k n_k = x_1 m_1 + \dots + x_k m_k;$$

i.e.,

$$x_1(n_1 - m_1) + \dots + x_k(n_k - m_k) = 0.$$

If all the  $n_i - m_i$  are  $> 0$  or all are  $< 0$ , no such  $x_i$  exist.

The converse also holds: If for some  $i$ ,  $n_i - m_i = 0$ , then we can choose  $x_i = 1$ ,  $x_j = 0$  for  $j \neq i$ . If for some  $i \neq j$ ,  $n_i - m_i < 0$  and  $n_j - m_j > 0$ , then we can choose  $x_i = n_j - m_j$ ,  $x_j = -(n_i - m_i)$  and  $x_k = 0$  for all  $k \neq i, j$ .



*Algorithms and Data Structures*

Problem 1 (2 points).

The EULERIAN CIRCUIT problem is to determine whether or not a given graph has a cyclic path that includes every edge of the graph exactly once. The HAMILTONIAN CIRCUIT problem is to determine whether or not a given graph has a cyclic path that includes every vertex of the graph **exactly once**.

A polynomial-time algorithm is known for only one of these two problems. Which one is it? (Don't specify the algorithm, just the problem. But give a brief reason to justify your answer.)

Problem 2 (5 points).

Consider the following two program fragments that set an array variable equal to an  $n \times n$  identity matrix:

```
A: for  $i := 1$  to  $n$  do
    begin for  $j := 1$  to  $n$  do  $a[i, j] := 0$ ;
            $a[i, i] := 1$ ;
    end;

B: for  $i := 1$  to  $n$  do
    for  $j := 1$  to  $n$  do
        if  $i = j$  then  $a[i, j] := 1$  else  $a[i, j] := 0$ ;
```

Which is faster if the assignment of a value to an array variable takes about **the same** amount of time as comparing  $i$  to  $j$ ? Which is faster if the assignment of a value to an array variable takes much longer than comparing  $i$  to  $j$ ? Justify your answers.

Problem 3 (10 points total).

Recall that  $\lfloor x \rfloor$  denotes the greatest integer  $\leq x$ .

**3a. (2 points).** Evaluate the sum  $\left[ \sum_{-10 < k < 10} 2^k \right]$  in closed form.

**3b. (2 points).** Evaluate the sum  $\sum_{-10 \leq k \leq 10} \lfloor 2^k \rfloor$  in closed form.

**3c. (6 points).** Prove or disprove:  $\lfloor x \rfloor + \lfloor y + (x \bmod 1) \rfloor = \lfloor x + y \rfloor$  for all real nonnegative  $x$  and  $y$ .

Problem 4 (15 points).

Assume that you have been hired by the R. J. Drofnats Corporation to write Phase 17.5 of a complex program. The application deals with an ordered tree in which every node has either no children or two children.

Phase **17** of the program has already been written by someone else. It represents the tree by having two pointers for each node  $x$ , namely  $x.l$  and  $x.r$ , where  $x.l$  is the left child and  $x.r$  is the right child. (These pointers are zero if  $x$  has no children; hence either  $x.l = x.r = 0$ , or  $x.l$  and  $x.r$  are both nonzero.)

Phase **18** of the program has, likewise, already been written by someone else. It also uses two pointers,  $x.l$  and  $x.r$ , for each node  $x$ . And again  $x.l$  stands for the left child of  $x$ . But  $x.r$  stands for the right sibling of  $x$  (i.e., the right child of  $x$ 's parent, if  $x$  is a left child, otherwise 0).

*Algorithms and Data Structures*

Your job is to write an interface between these two existing phases. Use an Algol-like language to sketch an algorithm that converts from the representation of Phase 17 to the representation of Phase 18.

Problem 5 (14 points total).

The Fibonacci numbers are defined by the recurrence  $F_0 = 0, F_1 = 1; F_{n+1} = F_n + F_{n-1}$ , for  $n \geq 1$ . This problem concerns the exact calculation of  $F_n$  when  $n$  is very large. It is known that there is a constant  $\alpha$  such that the exact binary representation of  $F_n$  fits in an  $n + O(1)$  bytes.

- 5a.** (4 points). Suppose you are using an algorithm for addition that takes  $\max(m, n)$  units of time to add an  $m$ -byte number to an  $n$ -byte number. How many units of time does it take to compute the value of  $F_n$  for large  $n$ , using the program

```

F0 = 0 ;
F1 = 1 ;
for i := 2 to n
    Fi := Fi-1 + Fi-2 ;
    
```

and the assumed algorithm for multiple-precision addition? Give an asymptotic answer that is correct to within  $O(n)$ .

- 5b.** (8 points). Let  $L_n = F_{n-1} + F_{n+1}$ . It is known that

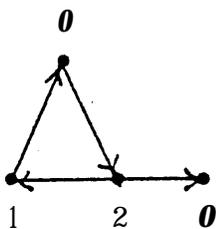
$$F_{2n} = F_n L_n \quad \text{and} \quad L_{2n} = L_n^2 - 2(-1)^n.$$

Therefore it is possible to compute  $F_{2^m}$  and  $L_{2^m}$  in  $O(m)$  multiplicative steps instead of order  $2^m$  additive steps. Assume that you can multiply an  $m$ -byte number by an  $n$ -byte number in  $mn$  units of time, and that you can add or subtract 2 from a large number in constant time. Determine the asymptotic time needed to compute  $F_{2^m}$  by this “accelerated” method, correct to within  $O(2^m)$ .

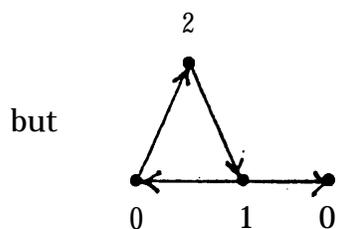
- 5c.** (2 points). When  $n = 2^m$  is large, which of these two methods is faster, given that the constant  $\alpha$  is approximately 0.08678?

Problem 6 (14 points total).

A *Grundy numbering* of a directed graph is a way to label its nodes in such a way that each vertex is labeled with the least nonnegative integer that does not appear as a label on any immediate successor of that vertex. For example,



is a Grundy numbering

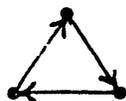


is not,

because the vertex labeled 2 violates the condition.

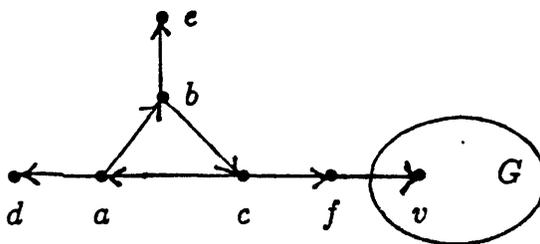
6a. (2 points). Prove that all nodes with  $k$  immediate successors must be labeled  $k$  or less, in a Grundy numbering.

6b. (3 points). Prove that the graph



has no Grundy numbering.

6c. (8 points), Consider a directed graph  $G'$  of the form



where  $G$  is arbitrary; the only directed arc between  $G$  and the special vertices  $a, b, c, d, e, f$ , is the arc from  $f$  to  $v$ .

Prove or disprove the following statement:  $G'$  has a Grundy numbering if and only if  $G$  has a Grundy numbering in which vertex  $v$  is labeled zero.

**Problem 1** (20 points),

You and some friends are going to a Chinese restaurant. You want to order 4 dishes from the menu to share. They must satisfy various constraints. Examples of such constraints are:

- One of your friends is a quasi-vegetarian, so at most one dish can use beef or pork.
- One of your friends doesn't like hot food, so at most one dish can be spicy.
- o No two dishes can have the same main ingredient.

- (A) (10 points). Translate this task into a state space search problem. Tell what state space you are using. What are the operators that transform one state into another? What are the goal states? What is the initial state?
- (B) (3 points). How does the state space change if you are to find a legal combination of 4 dishes with the least total cost? How must the search change?
- (C) (3 points): Describe a heuristic evaluation function for the problem of part B, suitable for use in the  $A^*$  algorithm, that can improve performance over blind search.
- (D) (4 points). Suppose you and a friend who you are taking to dinner instead play a game where you alternate choosing dishes, always satisfying the constraints, until 4 dishes have been chosen. You try to make the total price as low as possible, while your friend tries to make it as high as possible. How does the search space for this game compare with that for the original problem. Explain why the  $A^*$  algorithm does not apply to finding an optimal strategy for this game. Mention an algorithm which does.

**Problem 2** (15 points).

Consider the 4 propositions:

- S. Sandy is blond.
- K. Kim is blond.
- C. Chris is blond.
- D. Dana is blond.

Using the letters given, express each of the following statements in clause form, suitable for resolution theorem proving. Some statements may require several clauses. Half credit will be given for propositionally correct statements that are not in clause form (except for part (E))

- (A) (1 point). Sandy or Kim is blond.
- (B) (2 points). If Dana is blond, then Kim is blond.
- (C) (3 points). If Chris is blond, then Sandy and Dana are not blond.
- (D) (4 points). At least 2 of the 4 people are blond (warning: don't waste too much time on this part), .
- (E) (5 points). How many clauses are needed to express a statement of the form "At least  $M$  out of  $N$  people are blond." Express your answer as an expression in terms of  $M$  and  $N$ .

**Problem 3 (5 points).**

“LISP treats programs as data.” What does this mean, and why **is** it useful for AI programming. Give an example, specific to AI programming, of how this capability can be used.

**Problem 4 (20 points).**

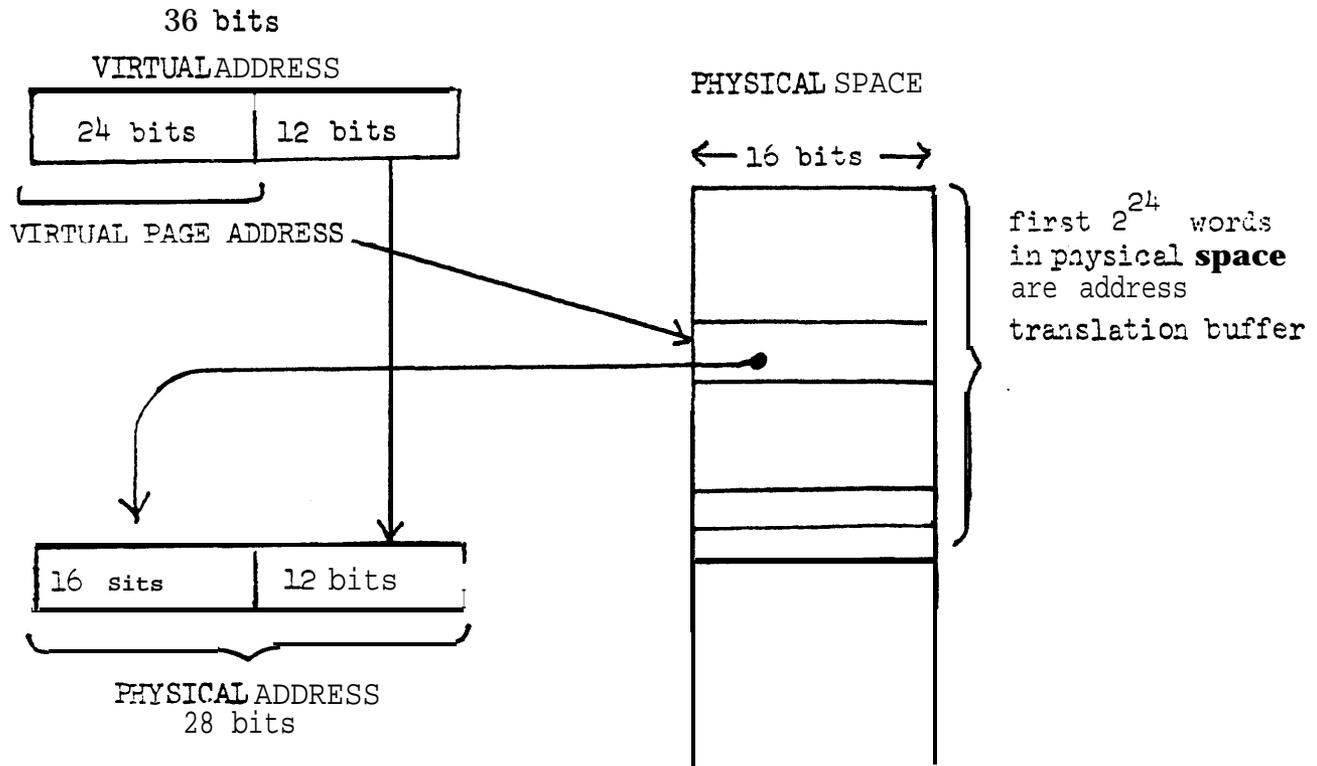
MYCIN uses backward chaining over a set of production rules to diagnose bacterial infection. HEARSAY-II uses a blackboard model with multiple knowledge sources to do speech understanding.

- (A) (10 points). Describe each of these expert system architectures. Describe the form of the dynamic database, the control mechanism, and the way knowledge is represented.
- (B) (6 points). What characteristics of each problem domain make the corresponding architecture suitable?
- (C) (4 points). What difficulties would you encounter in using the MYCIN style architecture to do the speech understanding problem?

⋮

**Problem 1** (15 points).

Consider the following overly simplistic **virtual** address mapping system:



$T_A =$  Time to access one word of memory.

Assume all pages are in memory for parts (a) and (b). Thus, a very straightforward implementation of a memory fetch mechanism would require an access to the address translation buffer for each access to memory. This would effectively require time  $2T_A$  to access a word.

- (a) (7 points). Design, draw and describe a hardware block diagram of a memory fetch mechanism. that implements the described mapping system and incurs *much* less than  $2T_A$  per mapped access, on the average.
- (b) (2 points). Make a realistic assumption about user program behavior and express the average memory access time required by your fetch mechanism per memory access as a function of the parameters of your model.
- (c) (6 points). Now, assume that the total physical space is  $2^{23}$  words, but you still want to have a virtual user space of  $2^{24}$  pages of 4096 bytes each. Describe how you would modify the virtual address mapping system. Note: You don't have to design a hardware block diagram, just describe how such a system would operate.

**Problem 2** (15 points).

Design a block diagram of a hardware system to find the GCD of two positive 16 bit numbers  $A$  and  $B$  (already residing in two 16 bit registers) using the algorithm provided

Spring 1984

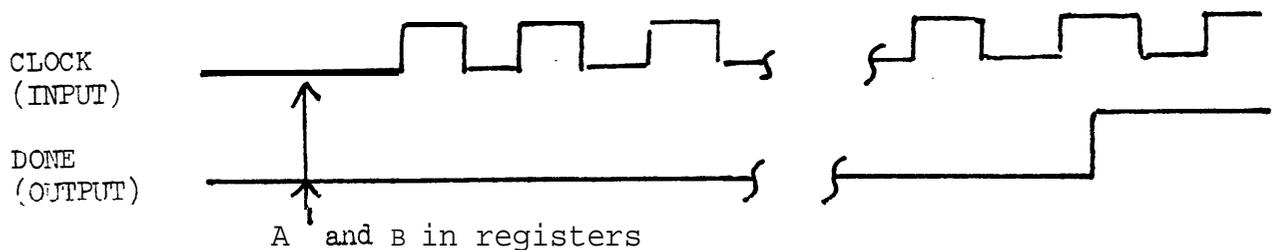
Hardware

below:

```
GCD(A, B)
  WHILE (A ≠ B)
    IF (A < B) THEN B := B - A
    ELSE A := A - B;
  RETURN(A);
```

- No microcoding is allowed.
- Provide a “done” signal when the answer is available.
- Assume you are given a clock input which begins after  $A$  and  $B$  are in their registers and which continues to infinity.
- The result should be stable in some register before the done output is raised.

Briefly explain its operation.



Problem 3 (30 points).

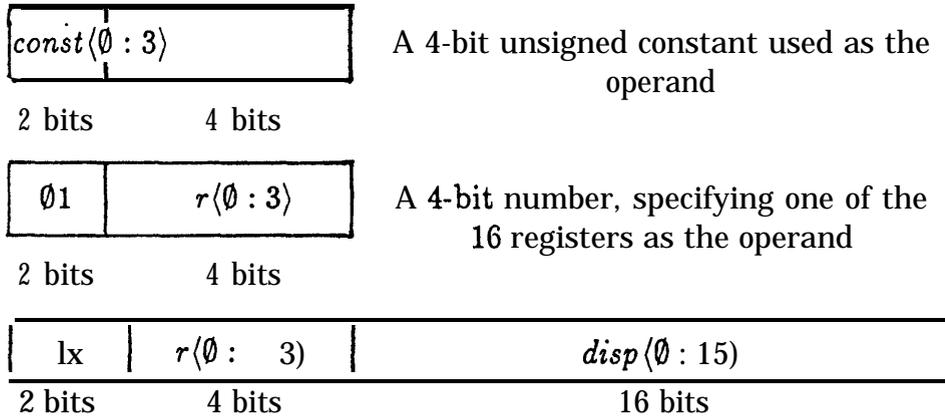
(For the purposes of this question; most-significant bits are marked as “0”, and least-significant ones are marked with positive indexes.)

You are given a 16-bit computer architecture, with 16-bit addresses, sixteen X-bit general-purpose registers, and *two-operand* instructions. Instructions have the form:

“op”: opcode 4 bits	“X”: Destination/Source operand specifier 6 or 22 bits	“Y”: source-2 operand specifier 6 or 22 bits
------------------------	--	--

These instructions mean  $X := X \text{ op } Y$  (ignore jumps, calls, returns, etc.). The “X” and “Y” operands can have the following forms each:

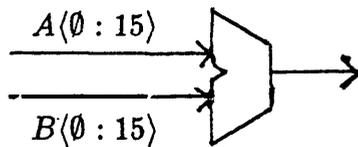
Hardware



Indexed addressing: the contents of register  $r$  and the N-bit displacement constant  $disp$  are added together, to form the effective address of the operand in memory.

The instruction set is fully orthogonal: any  $op$  can be combined with any two types of operands. (If  $X$  is a  $const$ , then the instruction means: “compute  $X \text{ op } Y$ , set condition-codes, and discard result”.)

The computer has an Arithmetic-Logic Unit (ALU), which is used both for address computations and for executing instructions ( $X \text{ op } Y$ ):

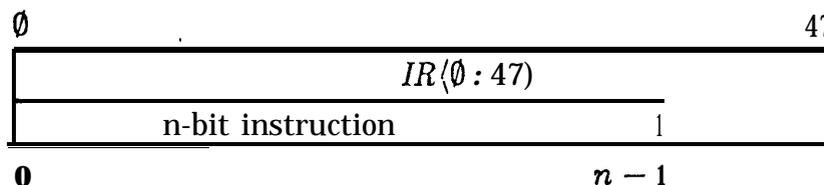


You are to design the part of the circuit which feeds the “first two” inputs to the ALU. These are the two 16-bit quantities which are to be fed to the ALU as *quickly as possible* after an instruction becomes available. Your goal is to feed such inputs to the ALU, so that the latter can produce a “useful” result—a result which is necessary and contributes towards executing the instruction.

(a) (7 points). For all nine possible combinations of addressing modes, what is the first “useful” result of the ALU? Provide the results in tabular form.

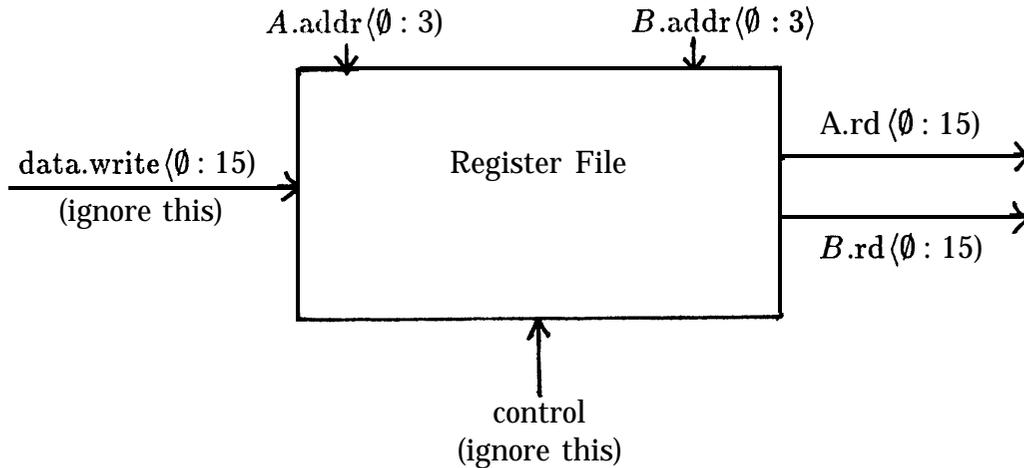
Besides the ALU, the computer also has:

- a 48-bit Instruction Register,  $IR\langle 0 : 47 \rangle$ . Assume that an instruction has already been placed and **left-justified** in that register.



**Hardware**

- a two-port 16 × 16 Register File:



- multiplexors (with any number of inputs) and small PLA's (i.e., Programmable Logic Arrays that can implement, say, up to 10 arbitrary boolean functions of up to 8 inputs).
  - has other blocks that you may ignore.
- (b) (10 points). What can the inputs to the ALU be, in all the various cases in question (a)? Which part of the instruction register do they come from? (Give bit position numbers.) Provide the results in tabular form. Ignore the problem of supplying ALU inputs for subsequent (micro-) cycles.
- (c) (13 points). Show the circuit which Feeds the above "first two" inputs to the ALU. Because you are aiming for high performance, you should *not* rely on slow, sequential decoding of the instruction and extraction of its operands—use combinational blocks (multiplexors and small PLA's, as mentioned above). The inputs to the ALU may come from the Instruction Register or from the register-file. If they come from the register-file you must show the circuit which supplies the corresponding address(es) to the register-File. If you use PLA(s), give their truth-table(s) or boolean equation(s). Wherever you need field-extractions, concatenations, etc., show precisely which bits (wires, (? : ?)), connect where.

**Numerical Analysis****Problem 1** (20 points).

- (a) (9 points).  $\mathbf{A}$  is an  $n \times n$  nonsingular matrix. Write a program in a FORTRAN, ALGOL, or PASCAL-like language to solve  $\mathbf{A}\mathbf{s} = \mathbf{b}$  using Gaussian elimination without pivoting.
- (b) (3 points). Give counts of the number of floating-point multiplications, additions, and divisions used by the program.
- (c) (4 points). Give a symmetric nonsingular matrix for which this algorithm fails. Define partial pivoting and show how it prevents failures of this kind.
- (d) (2 points). What is an alternative approach for solving  $\mathbf{A}\mathbf{x} = \mathbf{b}$  for  $\mathbf{x}$  when  $\mathbf{A}$  is large and sparse?
- (e) (2 points). Why is it useful, or even necessary, to change from the algorithm in part (a) to the algorithm in part (d) if  $\mathbf{A}$  is large and sparse?

**Problem 2** (20 points).

Let  $x_0$  be a given real number. Let the sequence  $\{x_n\}$  satisfy

$$x_n = (2 - x_{n-1})x_{n-1}.$$

- (a) (4 points). Prove that  $x_n \rightarrow -\infty$  as  $n \rightarrow \infty$  if  $x_0 < 0$  or  $x_0 > 2$ .
- (b) (8 points). Prove that if  $0 < x_0 < 2$  then  $x_n \rightarrow 1$  as  $n \rightarrow \infty$ .
- (c) (2 points). What is the order of convergence in part (b)?
- (d) (6 points). Suppose  $x_0$  is very *small* and positive. Show that approximately  $-\log_2 x_0$  steps are required before  $x_n$  becomes close to 1.

**Problem 3** (5 points).

Consider this computer program for calculating  $e^x$  when  $x \approx -10$ .

(Note:  $e^x = \sum_{i=0}^{\infty} x^i/i! = 1 + x + x^2/2 + x^3/6 + \dots$ )

```

xtoi := 1
i := 0
sum := 0.
while (xtoi > .0001 * sum) do
  begin
    sum := sum + xtoi
    i := i + 1
    xtoi := xtoi * x/i
  end
return (sum)

```

Will this program return an answer with relative error close to .0001 on a computer that does floating-point arithmetic with 10 significant decimal digits? Why or why not? Give a detailed explanation.

Spring '1984

**Numerical Analysis**

**Problem 4** (5 points).

Let  $E_n = \int_0^1 x^n e^{x-1} dx$ . It can be shown that  $E_0 = 1 - \frac{1}{e} \doteq .6321206$  and that  $E_n = 1 - n E_{n-1}$ . We could use this recurrence to compute  $E_{10}$ . Is this a stable algorithm? Why or why not?

**Problem 5** (5 points).

Let

$$\mathbf{H} = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & \cdots & 1/10 \\ 1/2 & 1/3 & & & & \cdot \\ 1/3 & & & & & \cdot \\ \cdot & & & & & \vdots \\ 1/10 & \cdot & \cdot & \cdot & \cdots & 1/19 \end{pmatrix}.$$

It is known that the condition number of  $\mathbf{H}$  is  $1.6 \times 10^{13}$ . If the system  $\mathbf{H}\underline{x} = \underline{b}$  is solved (by Gaussian elimination with partial pivoting) on a computer with 10 significant decimal digits, how many correct digits can we guarantee that the solution will have?

**Problem 6** (5 points).

Let  $\underline{x}^{(0)}$  be the zero vector. For  $n = 0, 1, 2, \dots$  let  $\underline{x}^{(n+1)}$  be the solution to

$$\mathbf{M}\underline{x}^{(n+1)} = \mathbf{N}\underline{x}^{(n)} + \underline{b}$$

where  $\mathbf{M}$  is a nonsingular square matrix. Let  $p = \|\mathbf{M}^{-1}\mathbf{N}\|$ . Suppose  $p < 1$ . Let  $\underline{x}$  satisfy  $\mathbf{M}\underline{x} = \mathbf{N}\underline{x} + \underline{b}$ . Let  $\mathbf{e}^{(n)} = \underline{x}^{(n)} - \underline{x}$ . Show that

$$\mathbf{e}^{(n+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{e}^{(n)}.$$

Given  $\epsilon > 0$ , how many iterations are required before  $\|\mathbf{e}^{(n)}\| \leq \epsilon$ ?

Spring 1984

**Software**

Problem 1 (APL: 3 points).

APL evaluates  $\times/2\ 3$  to 6. What does:

$$-/(10\ 2\ 0\ 30)\times-(5\ 4\ 3)+1$$

evaluate to?

Problem 2 (Array Layout: 2 points).

FORTRAN IV (1966) used to restrict array subscript expressions to the form “ $\pm k_1 * x \pm k_2$ ”. Here the  $k_i$  are integer constants, and  $x$  is a (simple) integer variable. Either of the  $k_i$ , or  $x$ , may be omitted. The reason for this was that such expressions compiled into very efficient code. Why would this be so? (Hint: Consider what operations are necessary to access array elements in a language where arrays are allocated statically.)

Problem 3 (Data Structures: 4 points).

Three *equivalent* definitions of a structured variable X value are presented here, in Pascal, C, and COBOL. Study one of them, and diagram how the various fields could reasonably be laid out in memory.

You are to assume a byte-addressable computer (8-bit bytes). An integer takes 4 bytes, and must be aligned on a 2 byte boundary. Characters take one byte, and are not aligned. Values in an enumeration type take one byte.

Pascal:

**type**

id = integer;

deps = **record**

**case** kind: (child, adult) **of**

    child: (mother, father: id);

    adult: (numchildren: integer; children: array [1..5] of id);

**end;**

**var X:**

**record**

    name: **array** [1..15] of char;

    age; integer;

    relatives: deps;

    idnum: id;

**end;**

Spring 1984

**Software**

C:

```
typedef int id;
struct deps
{
    enum (child, adult) kind;
    union {
        struct {id mother, father;} dep_a;
        struct {int numchildren; id children[5];} dep_b;
    } u;
};
struct
{
    char name[15];
    int age;
    struct deps relatives;
    id idnum;
} X;
```

COBOL:

```
01  x.
02  NAME PICTURE X(15).
02  AGE COMPUTATIONAL.
02  RELATIVES.
03  KIND PICTURE X.
88  CHILD VALUE 1.
03  DEP-A.
04  MOTHER COMPUTATIONAL.
04  FATHER COMPUTATIONAL.
03  DEP-B REDEFINES DEP-A.
04  NUMCHILDREN COMPUTATIONAL.
04  CHILDREN COMPUTATIONAL OCCURS 5 TIMES.
02  IDNUM COMPUTATIONAL.
```

**Problem 4** (Heap Memory: 3 points).

Give three of the most important tradeoffs involved in deciding between reference counting and mark-scan garbage collection.

**Problem 5** (Recursive-Descent Parsing: 8 points).

Program a recursive-descent parser for the grammar:

$$\begin{aligned} S &\rightarrow \text{if } C \text{ then } S X \mid A \\ X &\rightarrow \text{else } S \mid (\text{empty}) \end{aligned}$$

Procedures to recognize  $C$  and  $A$  are assumed given. **{if,then,else}** are tokens.  $A$  cannot begin with “if”.

It is suggested that *you* use a *pseudo-Pascal* with the constructs:

```
type token = . . . ;
F: file of token;
F ↑ – returns the next input token
Get(F) – advances to the next token.
```

Example: if  $F \uparrow = \text{“if”}$  then begin Get(F); . . . end else error;

**Problem 6** (Ambiguous Grammars: 7 points).

Consider the following suggested grammar for the if . . . then . . . else statement.

$$\begin{aligned} \text{stat} &\rightarrow \text{if cond then stat} \mid \text{substat} \\ \text{substat} &\rightarrow \text{if cond then substat else stat} \mid \text{simplestat} \end{aligned}$$

Show that the following statement is ambiguous:

**if** cond<sub>1</sub> **then** **if** cond<sub>2</sub> **then** simplestat<sub>3</sub> **else** **if** cond<sub>4</sub> **then** simplestat<sub>5</sub> **else** simplestat<sub>6</sub>

**Problem 7** (LISP: 2 points).

Mention some things found in LISP implementations that are not considered “pure” LISP.

**Problem 8** (SNOBOL: 1 point).

Describe informally what strings this SNOBOL pattern matches.

```
( 'aa' | 'b' ) ARB '9'
( ARB matches any string.)
```

**Problem 9** (File Systems and I/O: 7 points).

- (a) (2 points). (Aliases). Some file systems support the notion of aliases, which represent multiple names for the same file or directory. What fundamental problem does aliasing introduce, and how might you solve it?
- (b) (3 points). (File Allocation). For a file consisting of a sequence of bytes, rank *linked*, *contiguous*, and *indesed* file allocation with respect to each of:
  1. speed of sequential access
  2. speed of random access
  3. disk utilization
- (c) (2 points). (Buffer Management). Disk systems frequently provide *buffering* to support read-ahead and write-behind. Under what circumstances might you expect this facility to be more of a disadvantage than an advantage?

**Soft ware**

**Problem 10** (Multiprogramming: 7 points).

- (a) (2 points). (Process State Transitions). Assuming preemptive scheduling, draw a state diagram containing the principal process states and the transitions between them.
- (b) (3 points). (Performance). Why, in general, is it not possible to maximize both **throughput** and response?
- (c) (2 points). (Scheduling Policies). Distinguish between the following policies with respect to the degree to which they favor short jobs:
- First-Come-First-Served
  - Round-Robin
  - Multi-level feedback queues.

**Problem 11** (Concurrent Programming: 8 points).

To gain higher performance through concurrency, a simple batch system might consist of a `CardReader` process to read input cards, a `Compute` process to compute the results, and a `LinePrinter` process to output the results. **Assume:**

- disjoint address spaces
- messages are buffered in finite queues by the kernel, one queue per process
- messages are large enough to contain complete card or line images
- `Send` blocks the caller if the destination queue is full, but otherwise allows the caller to continue
- **Receive** blocks the caller until a message arrives

- (a) (4 points). Sketch these three processes.
- (b) (4 points). In the above problem, assume that **Send** blocks the caller until the recipient replies to the message via a **Reply** primitive. Sketch changes to your solution that would still provide the same level of concurrency.

**Problem 12** (Memory Management: 8 points).

- (a) (5 points). (Paging and Segmentation). Define and contrast paging and segmentation. **Be** sure to discuss performance, fragmentation, and level of visibility to application programmers.
- (b) (3 points). (Thrashing). If you were designing a multiprogramming demand-paging system and were concerned about thrashing, what mechanisms might you employ:
1. to reduce the likelihood of thrashing;
  2. to detect it, should it occur; and
  3. to eliminate it, if detected?

Assume that a single job never thrashes.

*Mathematical Theory of Computation***Problem 1** (Lisp: 17 points).

- (a) (7 points). Give a Lisp definition for the predicate “suffix”. For lists  $u, v$ ,  $\text{suffix}(u, v)$  is true if  $u$  is a suffix of  $v$ , nil otherwise. For example,

$$\begin{aligned} \text{suffix}((B A B C A), (B A B C A)) &= T, \\ \text{suffix}((A B C A), (B A B C A)) &= T, \\ \text{suffix}(\text{NIL}, (A B C)) &= T, \\ \text{suffix}((B), (B A B C A)) &= \text{NIL}, \\ \text{suffix}((A B A), (B A B C A)) &= \text{NIL}. \end{aligned}$$

- (b) (10 points). Prove by list induction that the predicate suffix as you have defined it is reflexive and transitive.

**Problem 2** (Logic: 5 points).

Let  $\alpha, \beta$  be sentences in propositional calculus. Suppose that  $\alpha \models \beta$  ( $\alpha$  tautologically implies  $\beta$ ),  $\not\models \neg\alpha$  ( $\alpha$  is not tautologically false),  $\not\models \beta$  ( $\beta$  is not tautologically true). Show that  $\alpha$  and  $\beta$  have a proposition symbol in common.

**Problem 3** (Program logics: 13 points).

Consider the following piece of code:

```

read c;
z := 0;
s := 1;
t := 3;
while s < c do
  begin
    z := z + 1;
    s := s + t
    t := t + 2;
  end ;
write z

```

Given a natural number  $c$ , it computes the natural number  $z$  such that  $z^2 \leq c < (z + 1)^2$  (in other words,  $z = \lfloor \sqrt{c} \rfloor$ ). Suppose that you were to write the proof of the correctness of this program.

- (a) (10 points). State the loop invariant for the loop in this program.  
 (b) (3 points). Give a short informal argument for the termination of this program.

Spring 1984

*Mathematical Theory of Computation*

**Problem 4** (Formal languages: 25 points).

(a) (18 points). Consider the following classes of languages:

K: Regular

CF: Context-free

CCF: Complements of context-free languages (that is, languages whose complement is context-free)

R: Recursive

RE: Recursively enumerable

CRE: Complements of recursively enumerable languages (that is, languages whose complement is recursively enumerable)

A: All languages

Give the Venn diagram of these classes. In other words, represent each of these classes as a region in the plane, showing the appropriate intersections, inclusions, etc. Give an example of a language from each region formed. Remember that partial credit will be given for partial answers.

(b) (7 points). Where on this diagram would you place

DCF: Deterministic context-free languages

P: Polynomial-time recognizable languages?

No examples are required for (b).

⋮

## Spring 1984 Comprehensive Solutions: Algorithms and Data Structures

**Problem 1.** It must be EULERIAN CIRCUIT, because HAMILTONIAN CIRCUIT is well known to be W-complete, and no polynomial-time algorithms for NP-complete problems are known.

(In fact, an Eulerian circuit exists if and only if the graph is connected and each vertex has even degree. But full credit was given to people who did not mention this fact; an indirect solution was all that was expected, since the reading list for this exam doesn't mention anything about Eulerian circuits.)

**Problem 2.** A does  $n$  more assignments than B, while B does  $n^2$  more comparisons than A; in other respects they are the same. If assignments and comparisons are about equally costly, fragment A is therefore preferable. If assignments are much costlier than comparisons, fragment B is preferable unless  $n$  is large. (More precisely, fragment B wins only when a single assignment costs more than  $n$  comparisons.)

**Problem 3a.** This is a geometric series that sums to  $\lfloor 2^{11} - 2^{-10} \rfloor = 2^{11} - 1$ .

**Problem 3b.** This is  $\sum_{0 < k < n} 2^k = 2^{11} - 1$ .

**Problem 3c.** The identity is true, since  $\lfloor x + y \rfloor = \lfloor \lfloor x \rfloor + (x \bmod 1) + y \rfloor = \lfloor x \rfloor + \lfloor (x \bmod 1) + y \rfloor$ .

**Problem 4.**

```

procedure convert (x: integer); {let  $x \neq 0$  denote a tree node}
begin if  $l[x] = 0$  then
    begin if  $r[x] \neq 0$  then
        writeln ( ' Error : Left son missing at node ', x : 1);
    end
else if  $r[x] = 0$  then
        writeln( ' Error: Right son missing at node ', x : 1)
else begin convert( $l[x]$ ); convert( $r[x]$ );
     $r[l[x]] := r[x]$ ;  $r[r[x]] := 0$ ;
end;
end;

procedure interface17to18;
var x: integer;
begin x := root of tree;
if  $x \neq 0$  then
    begin convert(x);  $r[x] := 0$ ;
    end;
e n d ;
    
```

**Problem 5a.** The time to compute  $F_{n+1}$  will be the time to add two numbers of  $\alpha n + O(1)$  bytes, namely  $\alpha n + O(1)$ . So the answer is  $\sum_{1 < k < n} (\alpha k + O(1)) = \alpha n^2/2 + O(n)$ .

**Problem 5b.**  $F_{2^k}$  and  $L_{2^k}$  have  $\alpha 2^k + O(1)$  bytes, so they can be formed from  $F_{2^{k-1}}$  and  $L_{2^{k-1}}$  in  $2\alpha^2 2^{2k-2} + O(2^k)$  units of time. Summing for  $1 \leq k \leq m$  gives  $2\alpha^2(4^m/3) + O(2^m)$ ; but subtract  $\alpha^2 2^{2m-2}$ , since  $L_{2^m}$  need not be computed.

**Problem 5c.** The second method takes  $(\frac{2}{3} - \frac{1}{4})\alpha^2 n^2$ , which clearly beats  $\frac{1}{2}\alpha n^2$  because  $\alpha$  is so small. (But it wins by only a constant factor. If addition were faster with respect to multiplication, the “brute force” first method would actually be superior. On the other hand faster multiplication methods will make the second method better than  $O(n^2)$ .)

**Problem 6a.** The  $k$  successors cannot contain all the labels  $\{0, 1, \dots, k\}$ , so the least unused label is  $\leq k$ .

**Problem 6b.** By part a, all labels must be 0 or 1. But if a vertex of outdegree 1 is labeled 0, its successor must be labeled 1, and conversely. Hence an isolated odd cycle has no Grundy numbering. (But a cycle of even length can be numbered in two ways.)

**Problem 6c.** If  $G'$  has a Grundy numbering,  $d$  and  $e$  are labeled 0 and  $\{a, b, c\}$  must contain distinct labels  $\{0, 1, 2\}$  in some order. Hence  $c$  must be labeled 0, hence  $j$  must be labeled 1, and  $v$  must be labeled 0. This establishes the statement in one direction. Conversely, if  $G$  has a Grundy numbering in which  $v$  is labeled 0, then we can label  $(a, b, c, d, e, j)$  with  $(2, 1, 0, 0, 0, 1)$ ; this is a Grundy numbering of  $G'$ .

## Spring 1984 Comprehensive Solutions: Artificial Intelligence

**Problem 1a.** The state space consists of all sets of up to four dishes. The operator is adding a dish to the set. The goal states are all sets of four dishes meeting all the constraints. The initial state is the empty set.

**Problem 1b.** The states must be augmented with the total cost of all dishes in the set. The search must deal with the possibility that the first legal combination that it finds may not be the least expensive, for example by searching through all legal states. (Note: just repeatedly choosing the least expensive dish that is still legal, will not necessarily yield the combination with the least total cost.)

**Problem 1c.** The final cost can be estimated by adding the total cost so far to the number of dishes not yet chosen times the cost of the least expensive dish.

**Problem 1d.** This is an adversary situation with alternate minimizing and maximizing. A\* won't work, because it finds a simple global minimum. Minmax can be used, augmented with alpha-beta pruning as desired.

**Problem 2a.** S or K.

**Problem 2b.**  $\sim$  D or K.

**Problem 2c.**  $\sim$  C or  $\sim$  S.  
 $\sim$  C or  $\sim$  D.

**Problem 2d.** S or K or C.  
S or K or D.  
S or C or D.  
K or C or D.

**Problem 2e.** At least  $A/4$  out of  $N$  people are blond if and only if every set of  $N - M + 1$  people has at least one blond. The later statement requires  $N$  choose  $N - M + 1$  clauses.

**Problem 3.** LISP programs are represented as LISP data structures. A LISP program can call EVAL on a data structure to have it interpreted as a program. The ability of a program to modify and examine other programs, or itself is a powerful tool for building complex programs, and for an ability for introspection. An example is converting a precondition for firing a rule from a predicate calculus type of notation into a procedure that will efficiently test the predicate.

**Problem 4a.** MYCIN's knowledge base consists of a set of rules of the form: IF condition . . . condition THEN conclusion. The rules have associated certainty information. The control structure reasons backward from each possible diagnosis, considering other propositions that could affect it, until it is confirmed or disconfirmed. The dynamic information consists of the conclusions that the system has reached about the propositions that it has considered. HEARSAY-II encapsulates its knowledge in several independent experts, which use various representations, including production rules. The experts examine the common blackboard for information of interest to them, and

## Spring 1984 Comprehensive Solutions: Artificial Intelligence

post hypotheses onto the blackboard. The blackboard is divided into layers which reflect the different levels of speech organization. The layering makes finding hypotheses of interest to a given expert more efficient. The control structure activates the expert which shows the most promise of improving the state of knowledge on the blackboard.

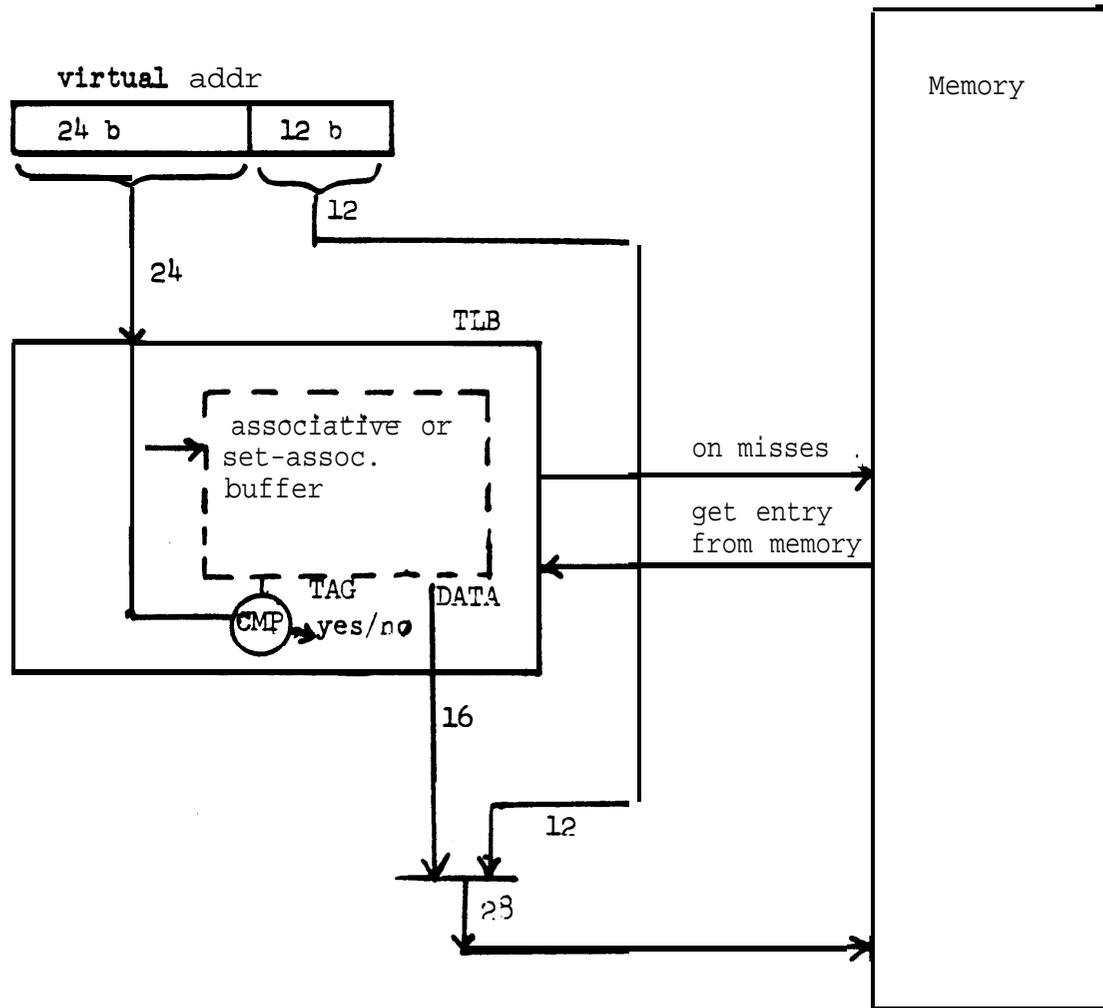
**Problem 4b.** The bacterial infection diagnosis domain has a fixed set of relevant hypotheses, which can all be listed in MYCIN's rules. The form of the rules is fairly natural for doctors to express, which makes knowledge acquisition easier. The backward chaining makes the system appear purposeful in the questions that it asks. Since there are few possible diagnoses, backward chaining can be very efficient. Since the number of possible conclusions in speech understanding is virtually unlimited, some amount of data driven hypothesizing is necessary, but backward chaining can be useful to fill in noisy spots in the data. The blackboard model can incorporate both. The partitioning into experts facilitates representation of the different types of linguistic information that can be brought to bear on the problem.

**Problem 4c.** Backward chaining is completely unsuitable, since the number of hypotheses is virtually unbounded. Also, MYCIN only deals with a fixed number of non-parameterized hypotheses, while in speech understanding, the hypothesis that a word occurs must be parameterized by the time of occurrence.

Spring 1984 Comprehensive Solutions: Hardware

Problem 1.

a)



The “TLB” (**T**ranslation lookaside buffer) is a small and fast buffer that maintains the few most recently used entries of the translation tables. It is like a cache (fully associative or set-associative). It works because of the locality-of-references property of usual real programs.

(b)

$$\text{Locality of references} \Rightarrow h \triangleq \text{hit-ratio} \triangleq \frac{\# \text{ of times the TLB hits}}{\# \text{ of requests (total) to the TLB}}$$

$$\approx 1 \text{ (but smaller)}$$

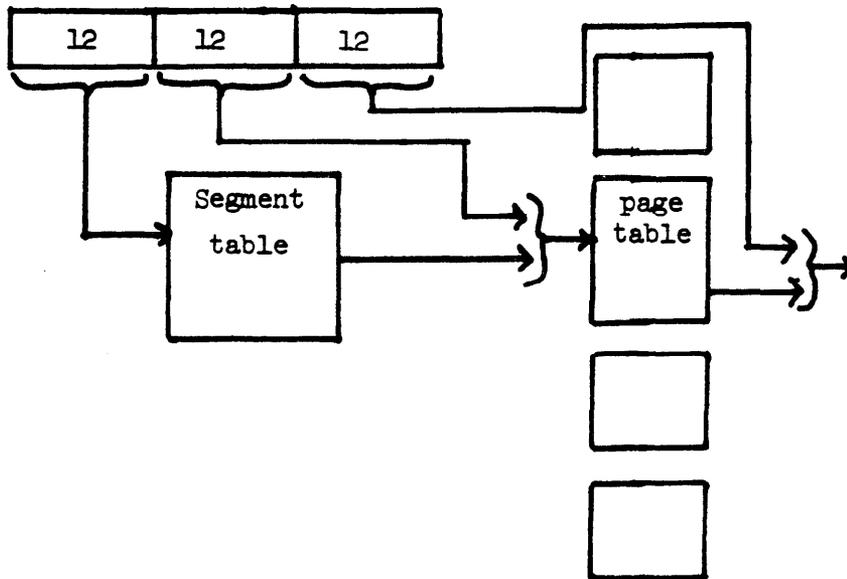
Assuming a miss takes  $T_A$  to be restored:

$$T_{avg} = h \cdot (T_A + T_{TLB}) + (1 - h) \cdot 2T_A$$

$$= hT_A + 2T_A - 2hT_A + h \cdot T_{TLB} = 2T_A - hT_A + h \cdot T_{TLB} \doteq (2 - h)T_A + h \cdot T_{TLB}.$$

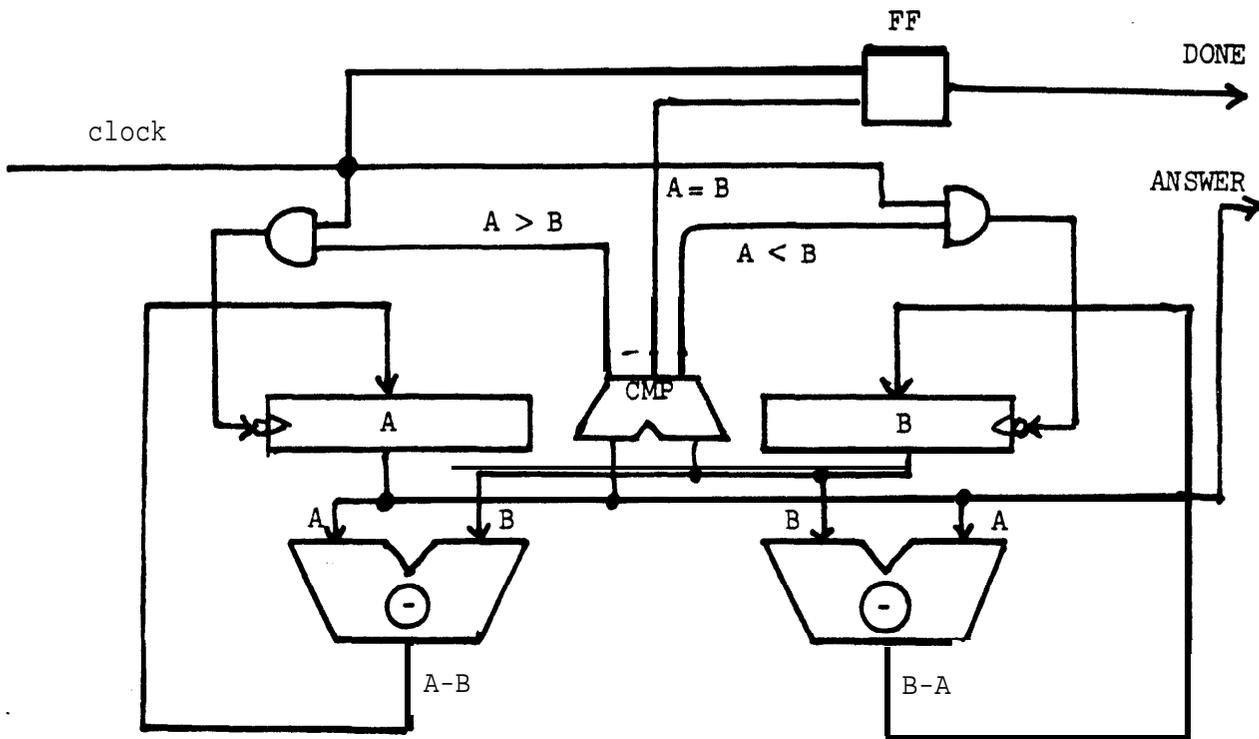
(c) Clearly, the translation tables (buffer) do not fit in main memory any more.

**Solution 1:** Two-step translation:

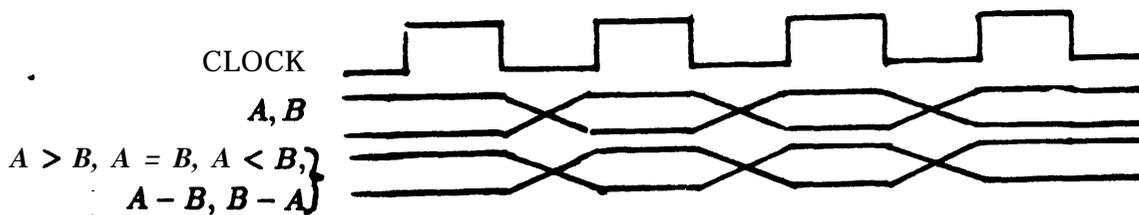


**Solution 3:** Imagine that the translation tables (buffer) are in *virtual* address space. Then, they can be paged-out of main memory. We just need to make sure that the translation tables for the first  $2^{24}$  words of virtual space **are always** in main memory.

**Problem 3.**



Both  $(A - B)$  and  $(B - A)$  are computed "continuously". Also,  $A$  and  $B$  are compared. On every clock cycle during which  $A \neq B$ , one of the  $A$  or  $B$  register is loaded with  $(A - B)$  or  $(B - A)$ , according to the result of the comparison. If and when  $A$  and  $B$  become equal, then none of them can be loaded anymore, DONE is asserted, and  $A$  contains the answer.



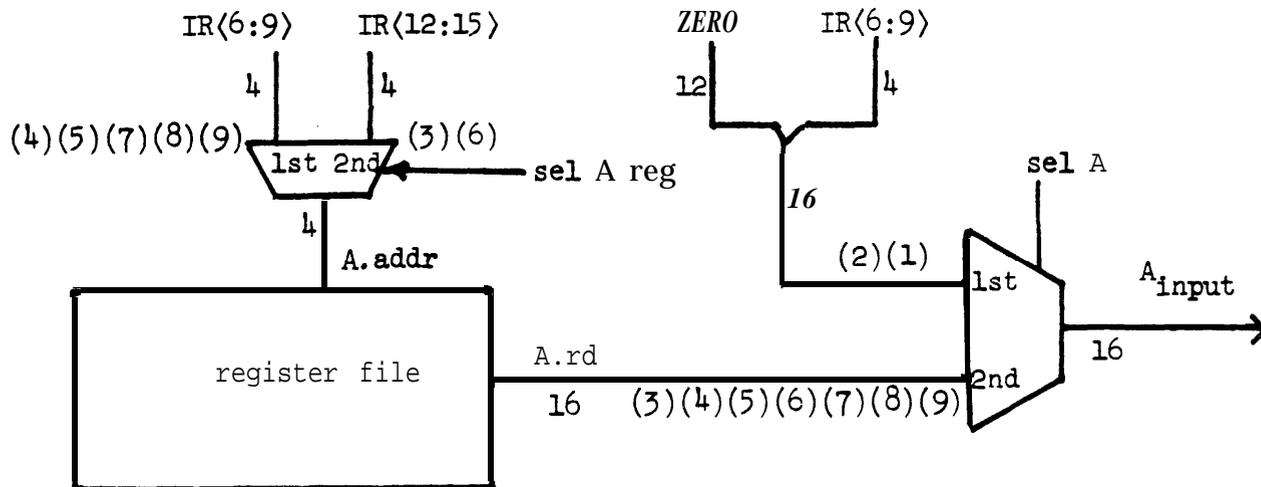
**Problem 3.**

(a) Case #	X	Y	ALU result	first "useful" .
(1)	$const_1$	$const_2$	$const_1 \text{ op } const_2$	
(2)	$const_1$	$r_2$	$const_1 \text{ op } r_2$	
(3)	$const$	$M[r_2 + d_2]$	$r_2 + d_2$	
(4)	$r_1$	$const_2$	$r_1 \text{ op } const_2$	
(5)	$r_1$	$r_2$	$r_1 \text{ op } r_2$	
(6)	$r$	$M[r_2 + d_2]$	$r_2 + d_2$	
(7)	$M[r_1 + d_1]$	$const$	$r_1 + d_1$	← same
(8)	$M[r_1 + d_1]$	$r$	$r_1 + d_1$	
(9)	$M[r_1 + d_1]$	$M[r + d]$	$r_1 + d_1$	

(b) Case #	A <sub>input</sub>	IR <sub>A</sub> ( )	B <sub>input</sub>	IR <sub>B</sub> ( )
(1)	$const_1$	(6:9)	$const_2$	(12:15)
(2)	$const_1$	(6:9)	$r_2$	r(12:15)
(3)(6)	$r_2$	r(12:15)	$d_2$	r(16:31)
(4)	$r_1$	r(6:9)	$const_2$	r(12:15)
(5)	$r_1$	r(6:9)	$r_2$	r(12:15)
(7)(8)(9)	$r_1$	r(6:9)	$d_1$	(10:25)

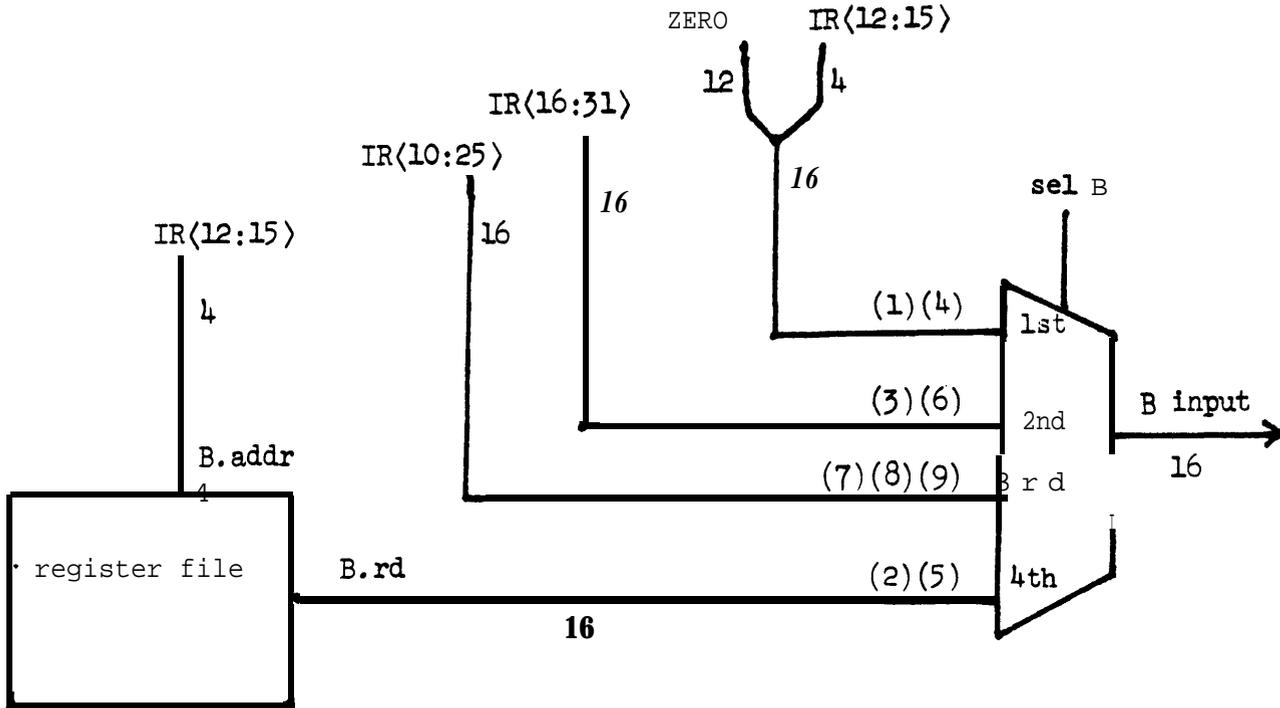
(c) A input:

Case #	A input
(1)(2)	(6:9) $const$
(4)(5)(7)(8)(9)	r(6:9) $reg.$
(3)(6)	r(12:15) $reg.$



**B** input:

Case	B input	
(1)(4)	(12 : 15)	const
(2)(5)	r(12 : 15)	reg.
(3)(6)	(16 : 31)	const
(7)(8)(9)	(10 : 25)	const



Multiplexer control for A and B inputs:

Case #	IR<4 : 5>	IR<10 : 11>	sel A	sel A reg	sel B
(1)	00	00	first	X	first
(2)	00	01	first	X	fourth
(3)	00	1x	second	second	second
(4)	01	00	second	first	first
(5)	01	01	second	first	fourth
(6)	01	1x	second	second	second
(7)(8)(9)	1X	x x	second	first	third

## Spring 1984 Comprehensive Solutions: Numerical Analysis

### Problem Ia.

```

# factor A into LU
  for i = 1, (n - 1)
    for j = i + 1, n
      begin
        A(j, i) = A(j, i)/A(i, i)
        for k = i + 1, n
          A(j, k) = A(j, k) - A(j, i) * A(i, k)
        end
      end
# forward solve Ly = b
  for i = 2, n
    for j = 1, (i - 1)
      b(i) = b(i) - A(i, j) * b(j)
# backward solve Ux = y
  for i = n, 1, -1
    begin
      for j = (i + 1), n
        b(i) = b(i) - A(i, j) * b(j)
      b(i) = b(i)/A(i, i)
    end

```

### Problem Ib.

$$\begin{aligned}
 \text{factor: } \quad \div & : \sum_{i=1}^{n-1} \sum_{j=i+1}^n 1 = \sum_{i=1}^{n-1} (n-i) = n(n-1) - \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \\
 \times & : \sum_{i=1}^{n+1} \sum_{j=i+1}^n \sum_{k=i+1}^n 1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^n (n-i) = \sum_{i=1}^{n-1} (n-i)^2 \\
 & = \sum_{i=1}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6} \\
 +/\- & : \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=i+1}^n 1 = \frac{(n-1)n(2n-1)}{6} \quad .
 \end{aligned}$$

$$\begin{aligned}
 \text{forward solve: } \quad \times & : \sum_{i=2}^n \sum_{j=1}^{i-1} 1 = \sum_{i=2}^n (i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \\
 +/\- & : \sum_{i=2}^n \sum_{j=1}^{i-1} 1 = \frac{n(n-1)}{2}
 \end{aligned}$$

$$\begin{aligned} \text{backward solve: } \div : \sum_{i=n}^1 i &= n \\ \times : \sum_{i=n}^1 \sum_{j=n}^{i+1} 1 &= \sum_{i=n}^1 (n-1) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \\ +/ -: \sum_{i=n}^1 \sum_{j=n}^{i+1} 1 &= \frac{n(n-1)}{2} \end{aligned}$$

$$\begin{aligned} \text{Totals: } \div : \frac{n(n-1)}{2} &= \frac{n^2}{2} + \frac{n}{2} = \frac{n^2}{2} + O(n) \\ \times : \frac{(n-1)n(2n-1)}{6} + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} &= \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} = \frac{n^3}{3} + O(n^2) \\ +/ -: \frac{n^3}{3} + \frac{n^3}{2} - \frac{5n}{6} &= \frac{n^3}{3} + O(n^2) \end{aligned}$$

(This was primarily a fact question.)

**Problem 1c.**

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

$\det(A) = -1$  so it is nonsingular yet the algorithm would attempt to divide by the zero in the  $a_{11}$  element.

At the  $i$ -th step of the factorization (eliminating the nonzeros below the diagonal in the  $i$ -th column), partial pivoting looks for the largest element (in absolute value) in the  $i$ -th column which is on or below the diagonal. The row containing this element and the  $i$ -th row are then interchanged before the elimination begins. This guarantees that the pivot element will be nonzero, since  $A$  is nonsingular.

For our example,  $PA \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$  and the algorithm works on this.

**Problem 1d.** (i) an iterative method which uses  $A$  pretty much unchanged (preserving its sparsity pattern). (ii) a sparse factorization technique which minimizes fill-in.

**Problem 1e.** The iterative technique only operates on the nonzeros in  $A$ , avoiding both the  $n^2$  storage (due to fill-in) and the  $n^3/3$  operation count which may result from naïve Gaussian elimination. If  $n$  is large,  $n^2$  storage may exceed the capacity of the computer system to handle.

**Problem 2a.** If  $x_0 > 2$  then  $x_1 = (2 - x_0)x_0 < 0$ , so we only need to look at the case  $x_i < 0$ . Then  $x_{i+1} = (2 - x_i)x_i$ , which is also  $< 0$ . Additionally,  $|x_{i+1}| = |2 - x_i||x_i| > 2|x_i|$ . So  $|x_{i+1}| > 2^{i+1}|x_0|$  for  $x_0 < 0$ , and  $|x_{i+1}| > 2^i|x_0| |x_0 - 2|$  for  $x_0 > 2$ . In both cases  $|x_i| \rightarrow \infty$  and  $x_i < 0, \forall i \geq 1$ , SO  $X_i \rightarrow -\infty$ .

**Problem 2b.** The fixed points are  $x = 0, 1$  since we want

$$x = 2x - x^2 \Rightarrow 0 = (1 - x)x.$$

(i) Let

$$\begin{aligned} e_n &= 1 - x_n \\ &= 1 - (2 - x_{n-1})x_{n-1} \\ &= 1 - 2x_{n-1} + x_{n-1}^2 = (1 - x_{n-1})^2 = e_{n-1}^2 \end{aligned}$$

If  $x_i \in (0, 2)$  then  $|1 - x| < 1$ , so  $e_{i+1} = e_i^2 < 1 \Rightarrow x_{i+1} \in (0, 2)$ . Therefore,  $x_i \in (0, 2) \forall i$  and  $e_i = e_0^{2^i} \rightarrow 0$  as  $i \rightarrow \infty$ , so  $x_i \rightarrow 1$ .

(ii) Alternative proof: If  $x_i \in (0, 3/4)$  then  $x_{i+1} \geq \frac{5}{4}x_i$ . This sequence increases monotonically and eventually  $x_{i+j} \geq (\frac{5}{4})^j x_i \geq 3/4$  (and  $\leq 15/16$ ), so  $\exists j$  such that  $x_{i+j} \in [\frac{3}{4}, 1)$  if  $x_i \in (0, \frac{3}{4})$ . If  $x_i \in (\frac{5}{4}, 2)$  then  $1 - x_{i+1} = 1 - 2x_i + x_i^2 > 0$ , therefore  $x_{i+1} \in (0, 1)$ . So eventually  $x_n \in [\frac{3}{4}, \frac{5}{4}]$  if  $x_i \in (0, \frac{3}{4}) \cup [\frac{5}{4}, 2)$ . Now if  $g(x) = (2 - x)x$  then

$$g : [\frac{3}{4}, \frac{5}{4}] = [\frac{15}{16}, 1] \Rightarrow g : [\frac{3}{4}, \frac{5}{4}] \subset [\frac{3}{4}, \frac{5}{4}]$$

and  $|g'| = |2(1 - x)| < 1/2$  on  $x \in [\frac{3}{4}, \frac{5}{4}]$ . Therefore the iteration converges to the fixed point.

**Problem 2c.** (i)  $e_{n+1} = e_n^2$ , so quadratically convergent. (ii)  $g'(x) = 2(1 - x) = 0$  at  $x = 1$ .  $g''(x) = -2 \neq 0$  at  $x = 1$ , so quadratically convergent.

**Problem 2d.** The idea is that for  $x_0 \ll 1$ ,  $x_1 = (2 - x_0)x_0 \approx 2x_0$  until  $x_i$  becomes moderately large, at which time there is a constant upper bound guaranteeing approximate convergence. For example, it takes  $\approx 11$  iterations to decrease  $e_n$  from .9 to .1 ( $e_n = e_{n-1}^2$ ,  $e_n = 1 - x_n$ ) and an additional 3 iterations to decrease  $e_n$  from .1 to  $10^{-8}$  ( $x_n = .99999998$ ).

Now for  $x_i < .1$  then  $x_{i+1} = (2 - x_i)x_i > (2 - .1)x_i$  and  $x_{i+1} < 2x_i$ , so

$$\begin{aligned} (2 - .1)^n x_0 \approx .1 &\Rightarrow n \approx \frac{-\log_2 10 - \log_2 x_0}{\log_2(2 - .1)} < \frac{4 - \log_2 x_0}{.9} \\ 2^n x_0 \approx .1 &\Rightarrow n \approx \frac{-\log_2 10 - \log_2 x_0}{1} \geq 3 - \log_2 x_0 \end{aligned}$$

so  $n \in [-\log_2 x_0 + 17, -1.2 \log_2 x_0 + 20]$ . The 1.2 coefficient can be dropped arbitrarily close to 1 if  $x_0$  small enough and by increasing the additive constant.

**Problem 3.** (i) On the second iteration  $x_{toi} := \frac{-10}{1} < 0$  and  $.0001 * \text{sum} = .0001$ , so this is a programming error, and the answer is no. (ii) No. If the test were  $\text{abs}(x_{toi}) > .0001 + \text{abs}(\text{sum})$  it would still not work.  $e^{-10}$  is  $< 10^{-4}$ , so an accuracy to .01% requires accurate digits to  $10^{-8}$ . Yet individual terms in the summation,  $x^i/i!$ , are as large as  $\frac{(-10)^{10}}{10!}$  ( $\approx 2700$ ) which does not have significant digits as small as  $10^{-8}$  (or even  $10^{-7}$ ) if we only have 10 digits. So accuracy in these digits is lost and never regained.

**Problem 4.**

$$\begin{aligned} E_n &= 1 - nE_{n-1} \\ &= 1 - n(1 - (n-1)E_{n-2}) = 1 - n + n(n-1)E_{n-2} \end{aligned}$$

$$= \sum_{i=0}^{n-1} \frac{(-1)^i n!}{(n-i)!} + n! E_0$$

so an error in  $E_0$  is magnified by  $n!$ , while  $E_n = \int_0^1 x^n e^{x-1} dx < 1$ . So this algorithm is not stable with respect to perturbations in the initial data.

**Problem 5. 0.** One theorem is:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} < \frac{\text{cond}(A)_\infty}{1 - \text{cond}(A)_\infty \frac{\|\delta A\|_\infty}{\|A\|_\infty}} [1.01(n^3 + 3n^2)\rho u]$$

where  $\rho$  is the growth factor  $\geq 1$  and  $u$  is the unit roundoff,  $= .5 \cdot 10^{-10}$ . So

$$\frac{\text{cond}(A)_\infty}{1 - \text{cond}(A)_\infty \frac{\|\delta A\|_\infty}{\|A\|_\infty}} [1.01(n^3 + 3n^2)\rho u] \geq \text{cond}(A)_\infty \doteq 10^2$$

and no correct digits are guaranteed.

Another appropriate theorem is:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq \|A\| \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}$$

where  $\delta b$  is the error in representing  $b$  in 10 significant digit. So even if the solution were exact we might have no significant digits.

**Problem 6.**

$$M\underline{x}^{n+1} = N\underline{x}^{(n)} + \underline{b}$$

$$M\underline{x} = N\underline{x} + \underline{b}$$

$$\Rightarrow M(\underline{x}^{n+1} - \underline{x}) = N(\underline{x}^{(n)} - \underline{x})$$

$$\Rightarrow Me^{(n+1)} = Ne^{(n)}$$

$$\Rightarrow e^{(n+1)} = M^{-1}Ne^{(n)}$$

We want

$$\begin{aligned} \|e^{(n)}\| &\leq \epsilon \\ e^{(n)} &= M^{-1}Ne^{n-1} = (M^{-1}N)^2 e^{(n-2)} \dots = (M^{-1}N)^n e^{(0)} \\ \Rightarrow \|e^{(n)}\| &\leq \|M^{-1}N\|^n \|e^{(0)}\| \\ \Rightarrow \|e^{(n)}\| &\leq \rho^n \|e^{(0)}\| \leq \epsilon \\ \Rightarrow n(\ln \rho) &\leq \ln \epsilon - \ln \|e^{(0)}\| \\ \Rightarrow n &\leq \frac{\ln \epsilon - \ln \|e^{(0)}\|}{\ln \rho} \end{aligned}$$

## Spring 1984 Comprehensive Solutions: Software

### Problem 1.

Note:  $-5 == (-5)$  except that  $-5 \text{ } ^{-}6$  is a legal array constant while  $(-5) \text{ } ^{-}6$  is not.

```

-/(10 20 30)×-(5 4 3)+1 ==
-/(10 20 30)×(¯6 -5 -4) ==
- /(-60 -100 -120) ==
-60 --100 -¯120 ==
-60 -20 ==
-80 .

```

### Problem 2.

The formula to get the address of an array element is:

$$\text{addr}(a \leftarrow i) = \text{addr}(a \leftarrow 0) + i * d,$$

where  $d$  is the size of each element of the array. **Thus:**

$$\begin{aligned} \text{addr}(a \leftarrow \pm k \leftarrow 1 * x \pm k \leftarrow 2) &= \text{addr}(a \leftarrow 0) + (fk \leftarrow 1 * x \pm k \leftarrow 2) * d \\ &= \text{addr}(a \leftarrow 0) \pm k \leftarrow 1 * x * d \pm k \leftarrow 2 * d \\ &= (\text{addr}(a \leftarrow 0) \pm k \leftarrow 2 * d) \pm (k \leftarrow 1 * d) * x \end{aligned}$$

In other words, the entire calculations can be done at compile-time, by just factoring the  $k \leftarrow i$  into the constants which are needed for the calculation anyway.

### Problem 3.

Field	Byte postions
<b>x</b>	<b>0..39</b>
name	0..14
hole	15
<b>age</b>	<b>16..19</b>
relatives	20..35
kind	20
<i>hole</i>	21
dep-a	22..29
mother	22..25
father	26..29
dep-b	22..35
num-children	22..25
children	26..35
idnum	36..39

The exact place where dep-a and dep-b start and end is a matter of definition.

**Problem 4.**

Usually, reference counting takes more time (since it has to be done on every assignment). However, it is spread out evenly over the entire computation, while (most) mark-scan algorithms cause a long pause at unpredictable times, which it may cause problems for real-time applications.

Reference counting cannot deal with cyclic structures.

With reference counting, you need to allocate space for the count, which in theory can become indefinitely large. Some mark-scan algorithms only require one or two extra bits per cell. (There is an optimization possible for reference counts, based on the observation that most reference counts are one. This is to just use a bit to indicate non-one reference counts, which means that these have to be looked up in a hash table.)

**Problem 5.**

```
procedure S;
begin
  if  $F \uparrow =$  "if" then
    begin
      Get(F);
      C;
      if  $F \uparrow \neq$  "then" Error;
      Get(F);
      S;
    end
  else A;
end;

procedure X;
begin
  if  $F \uparrow =$  "else" then
    begin
      Get(F);
      S;
    end;
end;
```

**Problem 6.**

Translate the productions

$$\text{stat} \rightarrow \text{if cond then stat} \mid \text{substat}$$
$$\text{substat} \rightarrow \text{if cond then substat else stat} \mid \text{simplest\&}$$

to the productions

$$S \rightarrow AS \mid T$$
$$T \rightarrow ATBS \mid R$$

where

$A = \mathbf{if\ cond\ then}$

$S = \mathbf{stat}$

$T = \mathbf{substat}$

$B = \mathbf{else}$

$R = \mathbf{simplestat}$

and the statement

$\mathbf{if\ cond_1\ then\ if\ cond_2\ then\ simplestat_3\ else\ if\ cond_4\ then\ simplestat_5\ else\ simplestat_6}$

transforms to

$AARBARRB$ .

Show the language is ambiguous by giving 2 parses for the sentence above.

Parse 1.

$S \rightarrow AS$   
 $A(S) \rightarrow A(T)$   
 $A(T) \rightarrow A(ATBS)$   
 $AA(T)BS \rightarrow AA(R)BS$   
 $AARB(S) \rightarrow AARB(T)$   
 $AARB(T) \rightarrow AARB(ATBS)$   
 $AARBA(T)BS \rightarrow AARBA(R)BS$   
 $AARBARRB(S) \rightarrow AARBARRB(T)$   
 $AARBARRB(T) \rightarrow AARBARRB$

parentheses are used to indicate the non-terminal being transformed on this application.

Parse 2.

$S \rightarrow T$   
 $T \rightarrow ATBS$   
 $A(T)BS \rightarrow A(ATBS)BS$   
 $AA(T)BSBS \rightarrow AA(R)BSBS$   
 $AARB(S)BS \rightarrow AARB(AS)BS$   
 $AARBA(S)BS \rightarrow AARBA(T)BS$   
 $AARBA(T)BS \rightarrow AARBA(R)BS$   
 $AARBARRB(S) \rightarrow AARBARRB(T)$   
 $AARBARRB(T) \rightarrow AARBARRB$

**Problem 7.**

Pure LISP is a functional, side-effect free language. The “state” of a program is therefore not meaningful, so neither variables or a program counter are defined. Therefore it does not contain assignments ( **SETQs** ), or the low-level control structures ( **PROG**, **GO** ).

**Problem 8.**

All strings which begin with either the pair **aa** or the letter **b**, and whose last character is **9**.

**Problem 9.**

- (a) It introduces cycles in the graph. Can be eliminated, for example, by disallowing aliases to directories.
- (b) contiguous, linked, indexed  
contiguous, indexed, linked  
linked, indexed, contiguous
- (c) Might degrade performance for random access – reading unnecessary blocks. Can also impair reliability by not allowing a client to flush blocks to disk when he wishes.

**Problem 10.**

- (a) Minimal states: ready, running, blocked.

Minimal transitions:

dispatch:	ready → running
preempt:	running → ready
wait for I/O, etc.:	running → blocked
I/O complete, etc.:	blocked → ready

- (b) Assuming a reasonable mix of jobs, good response requires preemptive scheduling. Preemptive scheduling introduces additional overhead at the expense of throughput. Conversely, highest throughput is achieved with FCFS scheduling. In that case, a long CPU-bound job will keep short interactive jobs from running, thereby degrading response.
- (c) In general, feedback queues are the best since a “long” job will drop to a lower-priority queue after one quantum; any remaining short jobs will no longer have it to contend with. Round-robin has the latter problem, but is better than FCFS, which doesn’t favor anyone. Naturally, in a pathological case where all jobs are the same length, FCFS is just fine.

**Problem 11.**

- (a) System flow control mechanisms take care of blocking a sender (producer) who has too many messages outstanding to a particular receiver (consumer). So, *CardReader* can read input as fast as possible and send the lines to *Compute*. Flow control takes care of blocking *CardReader* when it was reading too fast for *Compute*. Similarly, *Compute* processes the lines as fast as possible, and sends them on to *LinePrinter*, with similar flow control considerations. Thus, the solution would reduce to something like:

```
process CardReader;
    var msg : Message;
begin
    repeat
        (read input line into msg) ;
        Send ( ( Compute) , msg) ;
    until eof;
end;
process Compute;
    var msg : Message;
begin
    repeat
        Receive(msg);
        (compute) ;
        Send( (LinePrinter), msg);
    until eof;
end;
process LinePrinter;      var msg : Message;
begin
    repeat
        Receive (msg) ;
        (output msg) ;
    until eof;
end;
```

- (b) Two “helper” processes are needed to act as buffers. *CardReader* will now send to a *ComputeHelper* which simply accepts the message, buffers it locally and *Replies*. *Compute* then *Sends* to *ComputeHelper* to get the next line to process. Similarly, *Compute* and *LinePrinter* now interact via *LinePrinterHelper*.

N.B. You could use just one helper process, which would maintain two sets of buffers.

**Problem 12.**

- (a) Segmentation is primarily a means of organizing virtual memory. Segments are usually allocated to individual objects and are typically variable length. They are frequently allocated by the compiler. Segments can be directly mapped to the hardware, leading to segmented memory, but this is frequently done through paging. If hardware segmentation is used, external fragmentation can result. Moreover, swapping (large) segments in and out of main memory can be expensive, especially if relatively little of the segment is referenced.

Paging is primarily a means of implementing virtual memory. It solves the problems inherent in contiguous allocation of segments by providing fixed-size page frames into which any virtual page may be swapped. This results in less (external) fragmentation, but internal fragmentation results from larger mapping tables and wasted space in the last page. Performance can be worse than segmentation in the event that many I/Os are needed to swap in the same amount of memory, or better in the event that only those pages that are needed to swap in the same amount of memory, or better in the event that only those pages that are needed are swapped in.

- (b)
1. Limit the number of jobs accepted in the system.
  2. Monitor the page fault rate.
  3. Shed jobs.

**Spring 1984 Comprehensive Solutions: Mathematical Theory of Computation**

**Problem 1.** Lisp.

Definition of *suffix*:

$$\mathit{suffix}(u, v) \leftarrow (u = v \vee \neg \mathbf{n}v \wedge \mathit{suffix}(u, \mathbf{d}v)).$$

The form of this definition guarantees that *suffix* always terminates. We will use this property extensively when doing Boolean algebra (we will never deal with I).

*Reflexivity*:

$$\begin{aligned} \mathit{suffix}(u, u) &\equiv (u = u \vee \neg \mathbf{n}u \wedge \mathit{suffix}(u, \mathbf{d}u)) \\ &\equiv \mathbf{T} && \text{by reflexivity of } =. \end{aligned}$$

*Transitivity*: we want to show

$$\forall uvw (\mathit{suffix}(u, v) \wedge \mathit{suffix}(v, w) \rightarrow \mathit{suffix}(u, w)).$$

For arbitrary  $u, v$ , we will prove  $\forall w \Phi(w)$ , where

$$\Phi(w) \equiv (\mathit{suffix}(u, v) \wedge \mathit{suffix}(v, w) \rightarrow \mathit{suffix}(u, w)).$$

The proof is by list-induction.

i)  $\Phi(\mathit{nil})$ :

$$\begin{aligned} \mathit{suffix}(u, v) \wedge \mathit{suffix}(v, \mathit{nil}) &\rightarrow \mathit{suffix}(u, v) \wedge v = \mathit{nil} && \text{by the definition of } \mathit{suffix} \\ &\rightarrow \mathit{suffix}(u, \mathit{nil}) && \text{by substitution of equals.} \end{aligned}$$

ii)  $\forall xw (\Phi(w) \rightarrow \Phi(x.w))$ :

$$\text{Lemma: } \forall xuu' (\mathit{suffix}(u, x.u') \equiv (u = x.u' \vee \mathit{suffix}(u, u'))).$$

Proof: by definition of *suffix* and lisp axioms.

Now, for arbitrary  $x, u$ , assume  $\Phi(w)$  to prove  $\Phi(x.w)$ :

$$\begin{aligned} \mathit{suffix}(u, v) \wedge \mathit{suffix}(v, x.w) &\rightarrow \mathit{suffix}(u, v) \wedge (v = x.w \vee \mathit{suffix}(v, w)) && \text{by Lemma} \\ &\rightarrow \mathit{suffix}(u, x.w) \vee (\mathit{suffix}(u, v) \wedge \mathit{suffix}(v, w)) \\ &\rightarrow \mathit{suffix}(u, x.w) \vee \mathit{suffix}(u, w) && \text{by Inductive Hypothesis} \\ &\rightarrow \mathit{suffix}(u, x.w) && \text{by Lemma.} \end{aligned}$$

**Problem 2.** Logic.

Suppose that  $\alpha \models \beta$ ,  $\not\models \neg\alpha$ , and  $\not\models \beta$ . Assume  $\alpha$  and  $\beta$  have no proposition symbol in common. Since  $\not\models \neg\alpha$ , there is some truth assignment  $\theta$  such that  $\theta(\alpha)$  is true. Similarly, there is some truth assignment  $\vartheta$  such that  $\vartheta(\beta)$  is false. We “glue together” these two interpretations:

$$\eta(p_i) = \begin{cases} \theta(p_i) & \text{if } p_i \text{ occurs in } \alpha, \\ \vartheta(p_i) & \text{if } p_i \text{ occurs in } \beta. \end{cases}$$

This definition is meaningful because we assumed that  $\alpha$  and  $\beta$  shared no proposition symbol.

Since  $\eta(\alpha)$  is true and  $\eta(\beta)$  is false, we just contradicted  $\alpha \models \beta$ . Hence, our assumption must be false.

**Problem 3.** Program Logics.

Every time the loop is about to be entered, the invariant  $s = (z + 1)^2 \wedge t = 2z + 3 \wedge z \in \mathbf{N}$  holds.

The program terminates because

- since  $z \in \mathbf{N}$ ,  $s \in \mathbf{N}$  (by the loop invariant);
- $z$  increases each time through the loop; by the invariant,  $s$  does too;
- the loop eventually stops with  $s \not\leq c$ , since  $s$  is an integer and it keeps increasing.

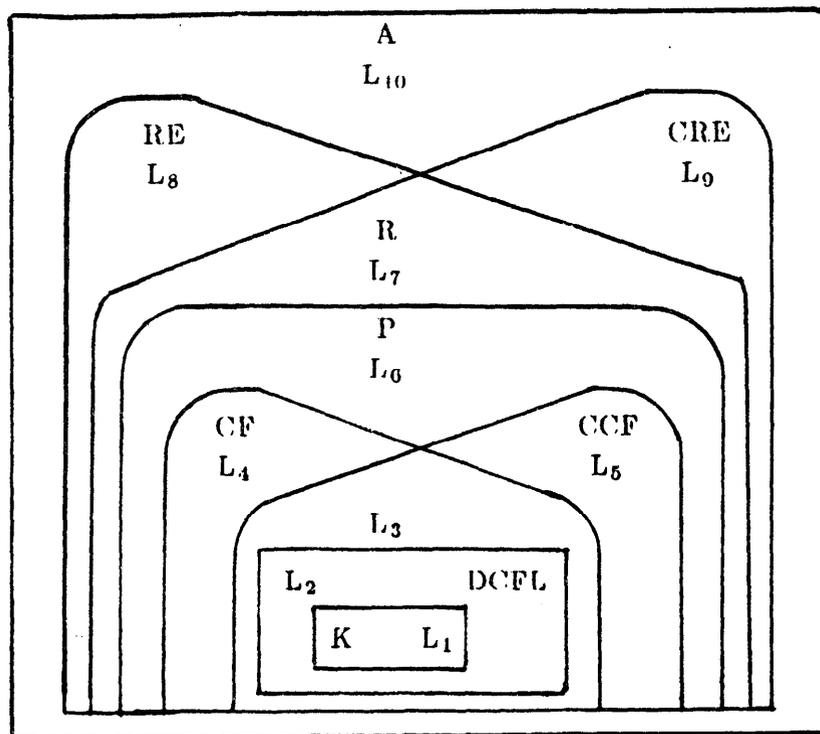
**Problem 4.** Formal Languages.

A few remarks:

- a) all regular languages are (deterministic) context free
- b) the class of deterministic context free languages is closed under complement
- c) all context free languages are recognizable in cubic time
- d) the class of polynomially recognizable languages is closed under complement
- e) the class of recursive languages is the intersection of the classes of recursively enumerable and co-recursively enumerable languages

$$\begin{aligned} L_1 &= \emptyset \\ L_2 &= \{a^n b^n : n \geq 0\} \\ L_3 &= \{a^m b^n : n = 2m \text{ or } n = m, m \geq 0\} \\ L^* &= \Sigma^* - L_5 \\ L_5 &= \{a^n b^n c^n : n \geq 0\} \\ L_6 &= \{(M, w) : M \text{ halts on } w \text{ within } |w|^4 \text{ steps}\} \\ L_7 &= \{(M, w) : M \text{ halts on } w \text{ within } 2^{|w|} \text{ steps}\} \\ L_8 &= \{(M, w) : M \text{ halts on } w\} \\ L_9 &= \Sigma^* - L_8 \\ L_{10} &= b\{(M, w, M', w') : A4 \text{ halts on } w \text{ and } M' \text{ does not halt on } w'\} \end{aligned}$$

Note: actually, one of  $L_2$ ,  $L_3$ , and one of  $L_6, L_7$ , was needed.



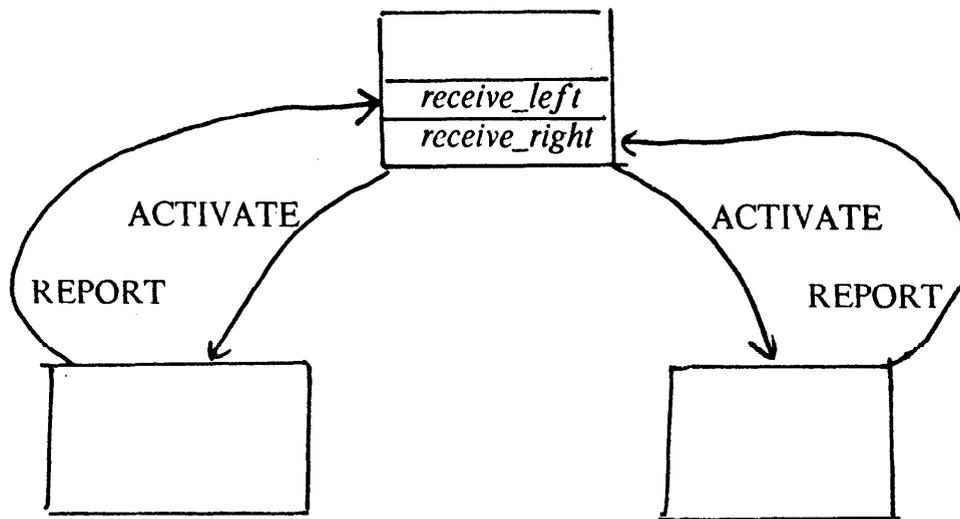


### ANALYSIS OF ALGORITHMS

Each problem is worth 15 points. For the purpose of problems 1 and 3, a complete binary tree is a binary tree in which all leaves have the same depth and all interior nodes have both a left and right child.

1) Suppose that at each node  $n$  of a complete binary tree there is a processor  $P_n$  and a value  $X_n$ . The values are stored in no particular fashion. Each processor can activate the processors at its left or right children by passing a particular value  $Y$  to be searched for. The command  $\text{ACTIVATE}(m, Y)$  is used to activate the processor at node  $m$ .

Each processor can report a boolean value to its parent; the output of the tree is the value reported by the root. Command  $\text{REPORT}(V)$  reports value  $V$  and halts the processor. Each processor  $P_n$  has local variables  $n.\text{receive\_left}$  and  $n.\text{receive\_right}$  to receive the values reported by its left and right children (denoted  $n.\text{leftchild}$  and  $n.\text{rightchild}$ ), respectively. A diagram of communication between processors appears below.



The following code is used to search for a value  $Y$  in the tree by calling  $\text{SEARCH}(\text{root})$ . It is executed at each processor, when that processor is activated.

```

function SEARCH( n : NODE, Y : VALUE ) : boolean:
begin
1)  if n = NIL then REPORT(FALSE)
2)  else if  $X_n = Y$  then REPORT(TRUE)
    else begin
3)      n.receive_Left := UNDEFINED;
4)      n.receive_right := UNDEFINED;
5)      ACTIVATE(n.leftchild, Y);
6)      ACTIVATE(n.rightchild, Y);
    repeat forever
7)          if (n.receive_Left = TRUE) OR (n.receive_right = TRUE)
            then REPORT(TRUE)
8)          else if (n.receive_Left = FALSE) AND (n.receive_right = FALSE)
            then REPORT(FALSE)
    end
end
end

```

Write a recurrence for  $T(h)$ , the maximum time taken by SEARCH when applied to a node  $n$  of height  $h$ , in terms of  $T(i)$  for one or more values of  $i \leq h$ . Assume that lines (1)-(2) together take time  $a$  and lines (3)-(6) together take time  $b$ . Also assume that the body of the loop of lines (7)-(8) is executed instantaneously, so that as soon as one or both of  $n$ 's children have reported values to  $P_n$ , such that the condition of line (7) or that of line (8) is met,  $P_n$  reports to its parent. Also assume that ACTIVATE and REPORT messages are passed instantaneously. **Explain your reasoning.**

2) Give tight big-oh and big-omega upper and lower bounds on the function  $T(n)$  defined by:

$$T(1) = 1$$

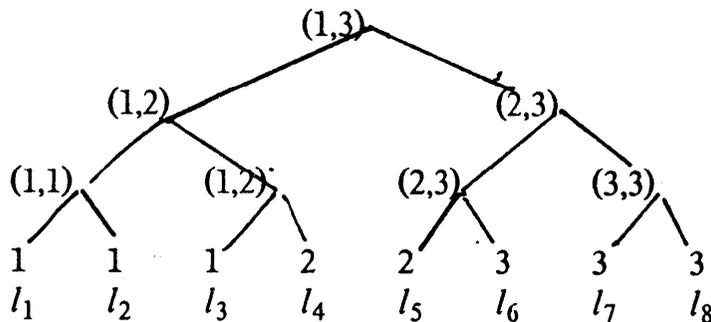
$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \text{ for } n = 2, 4, 8, \dots$$

Your bounds need apply only when  $n$  is a power of 2, i.e., you may assume that for all  $m$ ,  $T(m) = T(2^k)$ , where  $2^k$  is the smallest power of 2 greater than or equal to  $m$ . **Show your reasoning.**

3) Suppose we store integers at the leaves of a complete binary tree. Integers are in sorted order, from left to right, and duplicates are allowed. At interior nodes, the following fields are found:

<i>parent</i>	pointer to parent (NIL if root)
<i>lc</i>	pointer to left child (NIL if a leaf)
<i>rc</i>	pointer to right child (NIL if a leaf)
<i>low</i>	the smallest integer at a descendant leaf
<i>high</i>	the largest integer at a descendant leaf

For example, low and *high* values are indicated by pairs (*low*, *high*) at all the interior nodes of the following tree.



Your problem is to write a function  $RM(p)$  that takes a pointer  $p$  to a leaf and returns a pointer to the rightmost leaf with the same value. For example, if  $p$  points to  $l_4$ , a pointer to  $l_5$  is returned, and if  $p$  points to  $l_3$  a copy of  $p$  is returned.

Your program should be written in Pascal or a reasonable facsimile. It should visit as few nodes as possible. In particular, it should not immediately find the root of the tree by following parent pointers unless it turns out to be necessary to go through the root to find the desired leaf.

4) The partition problem (defined below) is NP-complete. You are to sketch a proof that the restricted partition problem (also defined below) is NP-complete.

*Hint:* It is easiest to use the fact that the partition problem is NP-complete. In transforming an instance of one problem to another, you may find it useful to transform the weights and add new weights.

*Given:* Finite set  $A = \{a_1, \dots, a_k\}$  with positive (strictly greater than zero) integer weights. The weight of  $a_i$  is denoted  $w(a_i)$ .

*Partition:* Is there a subset  $A' \subseteq A$  for which  $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$ ? i.e., can  $A$  be partitioned into two disjoint sets the sum of whose weights is the same?

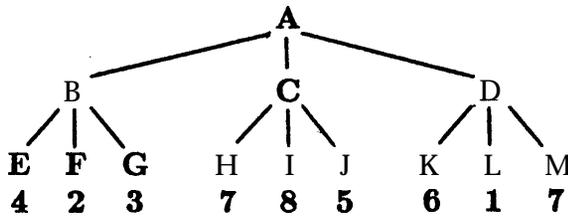
*Restricted Partition:* Is there a subset  $A' \subseteq A$  containing exactly  $k/2$  members (i.e., half the elements of  $A$ ) such that  $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$ ?

Artificial Intelligence

**1. (9 points)** For each of the following problems, would a forward or a backward chaining search be better? For each problem, give a very brief description of how the search proceeds in a forward or backward manner.

- A. (3 points) Proving a geometry theorem.
- B. (3 points) Understanding a line drawing of a blocks world scene.
- C. (3 points) Determining the molecular structure of a chemical from a mass spectrum.

**2. (10 points)** Consider the following game tree:



The numbers indicate the values of the leaf nodes. Assume that the first player tries to maximize the outcome, while the second player tries to minimize it.

**A. (2 points)** What moves will be taken if both players play optimally?

**B. (4 points)** If an alpha-beta search is used, not all of the leaf nodes need to be examined to determine the best move. The actual number of nodes examined depends by the order that the alpha-beta search first visits nodes. For some search order that visits the fewest leaf nodes possible, list which leaf nodes are visited.

**C. (4 points)** Why must the search proceed in a forward direction from the starting position, rather than backward from a goal position?

**3. (10 points)** Briefly describe the key ideas of the blackboard architecture (viz problem solving framework). Hearsay II is one example of a system built this way, but answer this question in a way that transcends the speech understanding application.

**4. (16 points)** Consider the problem of whether to put off doing an assignment on a given day. You have the following knowledge:

- An assignment can usually be put off.
- If the assignment is due tomorrow and the due date can't be postponed, then the assignment can't be put off.
- Due dates usually can't be postponed.
- If the assignment is a final paper, then the due date usually can be postponed.
- An assignment is usually not due tomorrow.
- An assignment usually is due tomorrow if you also have to take a test tomorrow

Using the convention below, write a TMS (Truth Maintenance System) style database that represents these facts. For each justification, list the node it supports, and the nodes on its *in* list and *out list*.

The convention: to make grading easier, please use the following letters (optionally preceded by a "¬" for negation) to represent nodes in your database:

- A: The assignment can be put off
- D: The <assignment is due tomorrow
- P: The due date can be put off
- F: The assignment is a final paper
- T: You have to take a test tomorrow

**5. (5 points)** Constraint propagation is a powerful problem solving method in AI. One of the best known applications is in the Waltz line labeling procedure for scene understanding. What is the reason that constraints were so effective in Waltz's case study?

**6. (10 points)** Lenat's AM program discovers concepts in the domain of elementary mathematics concepts. Its overall structure can be described as an application of "classical" AI ideas and methods. Here is your opportunity to so describe it for 10 points. (We're only looking for 10 points worth of answer here, just the essential ideas and methods.)

Winter 1985 - Hardware

Problem #1 (Parity) [10 Points]:

A byte of data is represented by 8 data bits -- **D0, D1, ... D7** -- and a parity bit **P** that is equal to the odd parity of the 8 data bits.

(i) [7 Points] Design a network to generate **P** from the data bits. Use 2-input exclusive-or gates. Use as few gates as possible.

(ii) [3 Points] Suppose a fault changes some of the bits to incorrect values. Which patterns of incorrect bit values will cause the parity to be incorrect? Identify all such patterns.

Problem #2 (Caches) [15 Points]:

Many computer systems place a cache memory between the CPU and main memory in order to increase the effective speed of main memory access.

(i) [8 Points] Describe the 3 most common mapping schemes:

[6 Points]

- a) Associative mapping
- b) Direct mapping
- c) Set-Associative mapping

[1 Point] Why might one choose Direct mapping over Associative mapping?

[1 Point] Why might one choose Set-Associative mapping over Direct mapping?

(ii) [5 Points] Given a Set-Associative Write-Back cache, describe in detail the actions that take place upon a memory read access and a memory write access.

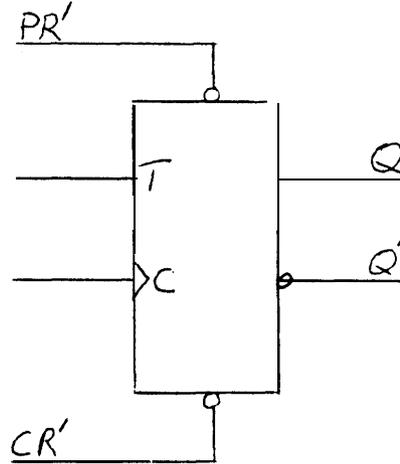
(iii) [2 Points] Give one advantage of a **Write-Back** cache over a Write-through cache. Give one advantage of a Write-through cache over a Write-Back cache.

**Problem #3 (Counters) [20 Points]:**

This is about counters -- circuits with one input that cycle through a fixed number of states in response to pulses on the input. In this problem the particular state sequence that the counters follow is not important -- a sequence of states corresponding to the binary number sequence is not required but is also not forbidden.

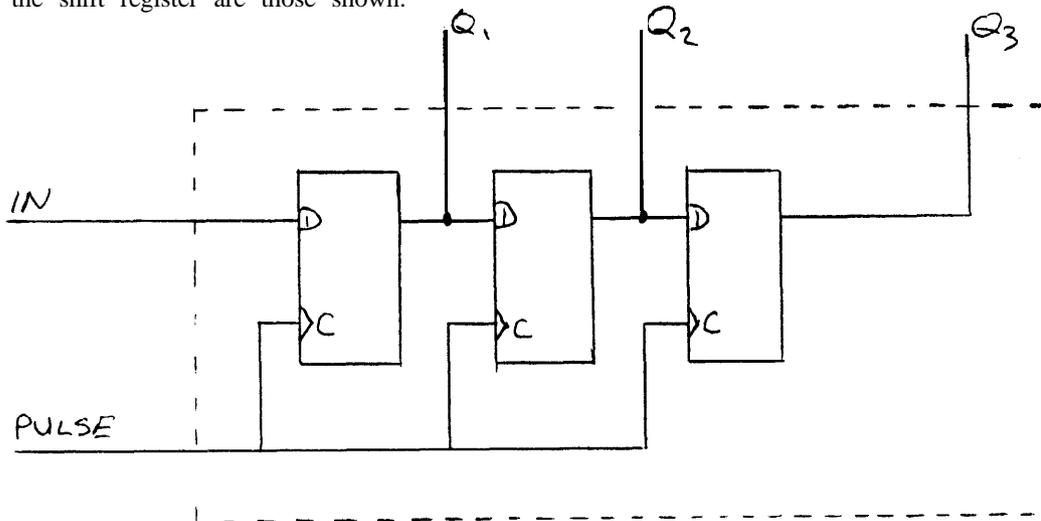
(i) **[5 Points]** Design a modulo-8 counter using three T flip-flops such as shown below and as few additional **NAND** gates as possible. Use no other component types. The counter has one signal input - **PULSE** - and is to increment one state for each pulse on **PULSE**. After 8 pulses it should return to the same state.

- $PR' = 0 \rightarrow Q = 1$
- $CR' = 0 \rightarrow Q = 0$
- C pulse causes Q to toggle
- if  $T = 1$ , Q to be unchanged
- if  $T = 0$ .



(ii) **[5 Points]** Design a modulo-5 counter using the same rules as in (i).

(iii) **[10 Points]** Design a modulo-5 counter using the 3-stage shift register shown below and as few additional NAND gates as possible. Use no other component types. The only available I/O for the shift register are those shown.



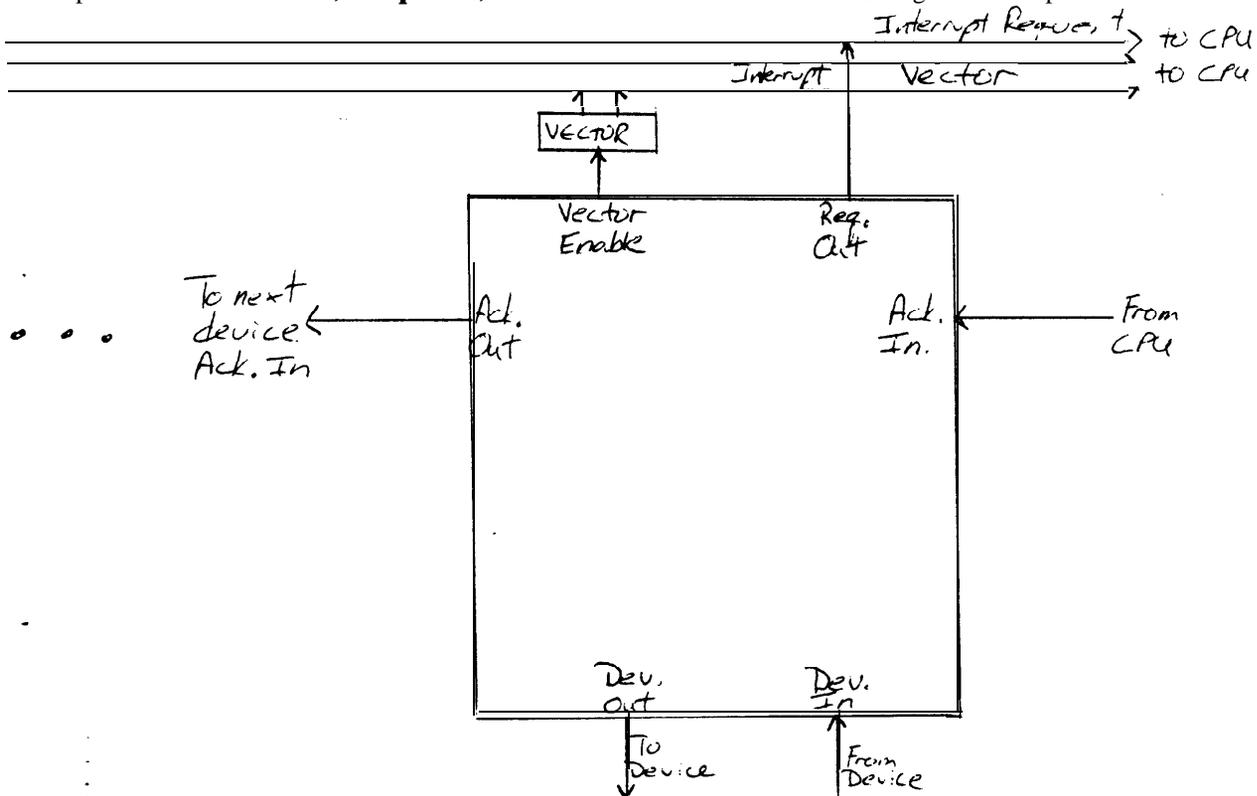
Problem #4 (I/O Buses) [15 Points]:

You are to design the interface logic for a "daisy-chained" interrupt structure I/O bus. This means that the interface interrupt priority is determined by physical placement of the interface in the "daisy-chain" (see figure below).

An interrupt transaction proceeds as follows:

When the device attached to your interface wishes to interrupt the CPU, it places a **1** on its "device interrupt request" (**Dev. In**) line and may return the line to 0 at any time before the interrupt transaction is complete. The interface must then place a **1** on the "CPU interrupt request" (**Req. Out**) line. This line is to be kept high until an acknowledgement is received from the CPU (through the "daisy-chain"). Once the acknowledgement is received, the interface must raise the "vector enable" line and keep it high until the acknowledge goes low. At this point the interrupt transaction is complete and the device must be signaled with the "device interrupt done" (**Dev. Out**) line.

[12 Points] You are to design the logic that takes the **Ack. In** and **Dev. In** signals as input and produces the **Ack. Out**, **Req. Out**, **Vector Enable** and **Dev. Out** signals as output.



[3 Points] This I/O bus has a design flaw (apparent when a high priority interface becomes active during a lower priority interrupt transaction). What changes would you make to the I/O bus design to correct this flaw?

Numerical **Analysis**"

1. (10 points) The following questions should be answered with either "True" or "False". The scoring is 2 points for a correct answer, -1 points for a wrong answer, and 0 points for no answer.
- True or False: **A** large condition number for a given problem means that the problem is well conditioned.
  - True or False: The condition number, with respect to inversion, of a matrix  $A$  is equal to  $\|A^{-1}\|$ .
  - True or False: **A** non-singular tridiagonal system of order  $n$  can be solved in  $O(n)$  operations.
  - True or False: The iteration  $\mathbf{x}_{n+1} = A\mathbf{x}_n + \mathbf{b}$  converges for any  $\mathbf{x}_0$  if  $\|A\| < 1$ .
  - True or False: The rate of convergence of the Bisection method is greater than that of Newton's method, when they both converge.

2. (3 points.) Let  $a$ ,  $b$  and  $c$  be three distinct numbers. What is the lowest degree of a polynomial which can be fitted to any given values of  $f(a)$ ,  $f'(a)$ ,  $f(b)$ ,  $f'(b)$ ,  $f(c)$ ,  $f'(c)$ ?

3. (4 points.) Give an example on a three decimal digit machine that shows that  $(a+b)+c$  can be different from  $a + (b + c)$  when floating point arithmetic **is used**.

4. (8 points.)

- a) (5 points.) Determine a positive number  $\xi$  so that the integration formula

$$I(f) = h[f(\xi h) + f(-\xi h)] \tag{†}$$

gives

$$I(p) = \int_{-h}^h p(x) dx$$

exactly for all **cubic polynomials**  $p(x)$ .

- b) (3 points.) Show that the magnitude of the error for  $f(z) = z^4$  is equal to  $\frac{8}{45}h^5$ .

5. (18 points.) Let  $\{x_i\}_{i=0}^{\infty}$  be the sequence obtained when the Newton method is applied to the equation  $x^5 - x = 0$  with starting value  $x_0$  a real number. Let  $r$  be the largest number such that  $x_n$  converges to zero (0) whenever  $|x_0| < r$ . In the following questions assume that  $|x_0| < r$ .
- (2 points.) What is the order of convergence of the Newton method in general?
  - (4 points.) What is the order of convergence in this particular example?
  - (5 points.) Assume that  $x_0 \neq 0$  and  $|x_0| < r < 1/\sqrt[4]{5}$ . Is the sequence  $x_0, x_1, x_2, \dots$ , monotonic in this example? Prove your answer.
  - (7 points.) Determine  $r$ . Hint: Use the result of c.
- 

6. (17 points.) Let

$$y_n = \int_0^1 \frac{x^n}{x+4} dx, \quad n \geq 0$$

- a. (2 points.) Show that

$$y_n + 4y_{n-1} = \frac{1}{n}. \quad (\dagger)$$

- b. (7 points.) Show that  $y_{11} < y_{10} < y_9$  and that  $\frac{1}{55} < y_{10} < \frac{1}{50}$ .
- c. (8 points.) Professor Staff claims that by choosing  $y_{10} = \frac{1}{50}$  and running the recurrence  $(\dagger)$  backwards (with sufficiently high precision), i.e.,

$$y_{n-1} = \frac{1}{4} \left( \frac{1}{n} - y_n \right), \quad n = 10, 9, 8, \dots, 1$$

he can obtain  $y_0$  with an error whose magnitude is smaller than  $2 \times 10^{-8}$ . Is he correct? Explain why or why not. (You may use the result in b, even if you have not proved it.)

## Software Systems

**Problem 1.** [10 points]

Consider the following grammar where  $S$  is the start symbol,  $\epsilon$  is the null string, lower case letters are terminal symbols and upper case letters are non-terminal symbols.

$$\begin{aligned} S &\rightarrow ABC & C &\rightarrow cm \\ A &\rightarrow Aa & c &\rightarrow cn \\ A &\rightarrow a \\ \mathbf{B} &\rightarrow cB \\ B &\rightarrow \epsilon \end{aligned}$$

- (a) [3 points] This grammar is not LL(1). Explain why by giving the non-LL(1) features.
- (b) [2 point] Why isn't it LR(1)?
- (c) [5 points] Give an LL(1) grammar with a minimum number of productions for the same language.

**Problem 2.** [9 points]

The parts of this question refer to the following simple computer instruction set. *Opnd* is an identifier (i.e. a memory location) and  $reg_i$  is either  $R_1$  or  $R_2$ .

$M_1$           A two register machine

ADD <i>opnd,reg<sub>1</sub></i>	$reg_1 \leftarrow reg_1 + opnd$
ADD <i>reg<sub>1</sub>,reg<sub>2</sub></i>	$reg_2 \leftarrow reg_2 + reg_1$
SUB <i>opnd,reg<sub>1</sub></i>	$reg_1 \leftarrow reg_1 - opnd$
SUB <i>reg<sub>1</sub>,reg<sub>2</sub></i>	$reg_2 \leftarrow reg_2 - reg_1$
MUL <i>opnd,reg<sub>1</sub></i>	$reg_1 \leftarrow reg_1 * opnd$
MUL <i>reg<sub>1</sub>,reg<sub>2</sub></i>	$reg_2 \leftarrow reg_2 * reg_1$
ST <i>reg<sub>1</sub>,opnd</i>	$opnd \leftarrow reg_1$
LD <i>opnd,reg<sub>1</sub></i>	$reg_1 \leftarrow opnd$

- (a) [2 points] Draw the DAG (Directed Acyclic Graph) for the following set of quadruples:

$$\begin{aligned} T_1 &= A + B \\ T_2 &= C * D \\ T_3 &= E - T_2 \\ X &= T_1 - T_3 \end{aligned}$$

- (b) [3 points] For the DAG in (a), give an  $M_1$  code sequence that would perform arithmetic operations in the same order as the quadruples. *The  $T_i$ 's are temporaries. You should eliminate unnecessary stores in to temporaries.*
- (c) (4 points] Give a shorter code sequence for Mr. What did you do to make the code sequence shorter?

**Problem 3.** [7 points]

- (a) [3 points] Give three ways you can cause aliasing in PASCAL.
- (b) [4 points] In many implementations of LISP, one can also cause aliasing. Yet, the problems one encounters with aliasing in PASCAL will not occur if one uses only the pure LISP features. Why?

**Problem 4.** [9 points]

Programming languages differ in the time at which various attributes are bound to identifiers. One such attribute is the size of an array. Give three times at which array size can be bound. For each binding time, give an example of a language that uses it and state in a few words how space for arrays can be allocated in that language.

**Problem 5.** [10 points]

In certain scientific programs the amount of time spent in each part of a loop body is much the same from iteration to iteration. Furthermore, most variables are accessed only once in a loop body. Call such programs *regular*. We will consider how demand paging can be adapted to perform well for regular programs.

- (a) [2 points] The Least-Recently-Used scheme for page replacement is based on an assumption about the pattern of page accesses in a program. What is this assumption?
- (b) [8 points] Suppose one had an estimate, for each memory location, of the average amount of time between accesses to that location and the amount of time since it was last accessed. Define a page replacement strategy that uses this information in a manner suitable for regular programs, and explain why you think this strategy should perform well for regular programs.

**Problem 6.** [15 points]

In many cases a program that uses monitors can be translated into a program that uses message-passing, and vice versa. Consider the following, where  $S_1$ ,  $S_2$ , and  $S_3$  represent sequences of statements.

```
monitor M;
  var x: integer;
      s: condition;
  procedure A (var a: integer);
    begin
       $S_1$  ; {  $S_1$  may modify a and x }
      s.signal;
    end;
  procedure B;
    begin
      while x < 0 do s.wait;
       $S_2$ ; {  $S_2$  may modify x }
    end;
   $S_3$  { initialization }
```

**You** are to give the code for a message-passing implementation of the monitor and the code a process executes to get the effect of the call to monitor procedure A. Assume that a message contains the id of the process sending the message and a variable-length data field. (Don't spend time on the details of how message information is encoded.) The procedures for message-passing are

- |                    |  |
|--------------------|--|
| send (P, message). | <i>Send a message to P. The sending process is not blocked.</i>  |
| receive (message). | <i>Receive a message from any process. The receiving process is blocked until a message arrives if one is not immediately available.</i> |

**You** may also assume that you are given procedures for operating on a queue.

## Mathematical Theory of Computation

**Problem 1:** (20 points)

WC use the following notation:

- \* represents string concatenation
- A represents the **empty** string
- u, v** represent characters (strings of length 1)
- x, y** are general strings

We **define** the unary function **rev** and the binary function **rev2** by the following set of axioms:

- |   |   |
|---|---|
| <ol style="list-style-type: none"> <li>1. <b>rev(A) = A</b></li> <li>2. <b>rev(u) = u</b></li> <li>3. <b>rev(u * x * v) = v * rev(x) * u</b></li> </ol> | <ol style="list-style-type: none"> <li>4. <b>rev2(A, y) = y</b></li> <li>5. <b>rev2(u * x, y) = rev2(x, u * y)</b></li> </ol> |
|---|---|

Prove the following facts about **rev** and **rev2**:

- (a) (8 points) **rev(x \* u) = u \* rev(x).**
- (b) (12 points) **rev2(x, A) = rev(x)** [Hint: Prove a more general result!]

Be **brief!** But be sure to indicate explicitly your inductive sentences and the inductive principles used. You may use without proof facts about \* and A. You may *not* assume **any** facts about **rev** or **rev2**.

**Problem 2:** (14 points)

Give a **model** for or refute (prove **unsatisfiable**) by resolution **each** of the following two sentences:

- (a)  $\forall x \exists y [p(x, y) \wedge \forall x \neg p(x, x)]$
- (b)  $[\forall x q(x)] \equiv [\exists x \neg q(x)]$

**Problem 3:** (6 points)

Give a decidable language  $\mathcal{L}$  such that  $\mathcal{L} \notin \text{NP}$  and  $\mathcal{L} \notin \text{CO-NP}$ .

You may assume that some very plausible conjectures are true (e.g.  $P \neq \text{NP}$ ), but if you do, make your assumptions explicit.

**Problem 4:** (8 points, 2 points for each part)

What, class of languages is accepted by each of the following classes of **machines**:

[N.B.: A language is *accepted* by a given machine if that language is exactly the set of strings on which the machine halts in a “yes” state.]

- (a) Nondeterministic Turing machines.

- (b) Deterministic finite state machines augmented by three counters. [A counter is an auxiliary variable that can be incremented by one, decremented by one, and tested to see if it is equal to 0.]
- (c) Deterministic Turing machines with an oracle for propositional satisfiability.
- (d) Deterministic finite state machines with one queue. [A queue is an auxiliary string in which characters can be added to the end of the string, and characters can be read and/or removed from the front of the string. The string can also be tested to see if it is empty.]

**Problem 5:** (12 points, 3 points for each part)

- (a) Which of the following languages over the alphabet  $\Sigma = \{0, 1\}$  are regular. Prove your answer:
  - i) The set of all strings that contain neither three consecutive 0's nor three consecutive 1's.
  - ii) The set of all strings that contain an equal number of 0's and 1's.
- (b) Prove that the following languages over the alphabet  $\Sigma = \{0, 1\}$  are context free. You need not necessarily give a grammar.
  - i)  $\mathcal{L} = (\mathcal{L}')^*$ , where  $\mathcal{L}' = \{ww^R \mid w \in \Sigma^+\}$  and  $w^R$  denotes the reverse of the string  $w$ .
  - ii)  $\{w \mid \text{every prefix of } w \text{ has at least as many 0's as 1's}\}$ .

**Problem 6: (0 points)**

Show proficiency in one of the following: Greek, Hebrew.

⋮  
⋮  
⋮

## Analysis of Algorithms Solutions

### Problem 1.

Let  $T(h)$  be the maximum time to determine whether the value  $Y$  is in the processor tree. If  $Y$  is not in the tree, all nodes must be queried, giving the maximum time; assume that  $Y$  is not in the tree.

The contributions to time  $T(h)$  are  $a$  from lines (1)-(2), then  $b$  from lines (3)-(6), and an additional  $T(h - 1)$  time units as control passes in parallel to the next level of the tree. With the base case that time  $T(0) = a$  (for it takes only  $a$  units to return FALSE from a NIL node) we have

$$T(h) = a + b + T(h - 1), \quad T(0) = a.$$

### Problem 2.

Expand by using the equation for  $T(n)$  to substitute for  $T(n/2)$ , then  $T(n/4)$ , and so on. That is, the second equation can, if we substitute  $n/2^i$  for  $n$  be written as

$$T(n/2^i) = 4T(n/2^{i+1}) + n^2/2^{2i}.$$

Thus,

$$\begin{aligned} T(n) &= 4[4T(n/4) + n^2/4] + n^2 \\ &= 16T(n/4) + 2n^2 \\ &= 16[4T(n/8) + n^2/16] + 2n^2 \\ &= 64T(n/8) + 3n^2 \end{aligned}$$

and in general,

$$T(n) = 4^i T(n/2^i) + in^2.$$

Let  $i = \log_2 n$ , so  $T(n/2^i) = 1$ , to get

$$\begin{aligned} T(n) &= 4^{\log_2 n} + n^2 \log_2 n \\ &= n^2(\log_2 n + 1). \end{aligned}$$

Thus,  $T(n)$  is  $O(n^2 \log n)$  and  $\Omega(n^2 \log n)$ .

### Problem 3.

The general idea is to first find the lowest point that is an ancestor of the desired leaf, and then descend to that leaf by binary search. To go up, we have to be careful not to fall off the root, if the lowest ancestor turns out to be the root. We may sketch it as:

```

function rm(p);
begin
  ancestor := goup(p, p↑.value);
  rm := godown(ancestor, p↑.value)
end;

```

The functions **goup** and **godown** can be sketched:

```

function goup(q, v);
begin
  if q↑.parent = nil then goup := q
  else if q↑.high > v then { no nodes with value v to right of q }
    goup := q
  else goup := goup(q↑.parent, v)
end;

function godown(q, v);
begin
  if q↑.rc = nil then { q is a leaf }
    godown := q
  else if q↑.rc↑.low > v then { no nodes with value v to the right }
    godown := godown(qf.lc, v)
  else godown := godown(qf.rc, v)
end;

```

#### Problem 4.

Let PP stand for “partition problem” and RPP mean “restricted partition problem” on the set  $A$ . We need to show that RPP is NP-complete. Clearly RPP is in NP, for it takes time  $|A|$  to sum the weights  $\sum_{a \in A'} w(a)$  and  $\sum_{a \in A - A'} w(a)$ , with constant time to verify equality.

We next exhibit a polynomial reduction of PP to RPP. First add 1 to all weights, defining  $w'(a) = 1 + w(a)$  for all  $a \in A$ . Then for  $0 \leq i \leq |A|$  (in order), apply RPP to the set  $A$  augmented with  $i$  elements of weight 1. We claim the least value of  $i$  that generates an instance of RPP also generates an instance of PP. If there is no instance of RPP, then there is no PP. Furthermore, this reduction is polynomial in nature.

The reduction takes at most  $|A| + 1$  applications of RPP on a set no larger than  $2|A|$ , clearly a polynomial in  $|A|$ . It remains to prove that PP is equivalent to the above procedure.

Let  $n = |A|$ . At level  $i$ , the existence of a partition implies that  $(n+i)/2$  elements have the same weight sum as some  $(n-i)/2$  other elements. This implies a true partition, for all of the weight 1 elements must be in either  $A'$  or  $A - A'$ .

If no partition is determined by the above method, then for  $0 \leq i \leq n$ , there is no partition of  $(n+i)/2$  versus  $(n-i)/2$  elements. This means there is no partition at all. This completes the polynomial reduction of PP to RPP, proving that RPP is NP-complete.

Note: Several students added new elements with weight 0 and performed a similar analysis. Partial credit was given even though the problem statement expressly forbids weights  $w(a) \leq 0$ .

### Artificial Intelligence

**1. (9 points)** For each of the following problems, would a forward or a backward chaining search be better? For each problem, give a very brief description of how the search proceeds in a forward or backward manner.

**A. (3 points)** Proving a geometry theorem.

Backward. Search from the theorem to other theorems that could prove it until axioms are reached.

**B. (3 points)** Understanding a line drawing of a blocks world scene.

Forward. Reason from the lines to **successively** larger scale information.

**C. (3 points)** **Determining the molecular structure of a chemical** from a **mass spectrum**.

Forward. Reason from the spectrum toward larger parts of the compound

2. **(10 points)** Consider the Following game tree:

A									
B			C			D			
E	F	G	H	I	J	K	L	M	
4	2	3	7	8	5	6	1	7	

The numbers indicate the **values of the leaf nodes**. Assume **that** the first player tries to maximize the outcome, while the second player tries to minimize it.

**A. (2 points)** What moves will be taken if both players play optimally?

A to C to **J**

**B. (4 points)** If an alpha-beta search is used, not all of the leaf nodes need to be examined to determine the best move. The actual number of nodes examined depends by the order that the **alpha-beta** search first visits nodes. For some search order that visits the fewest leaf nodes possible, list which leaf **nodes are visited**.

A, C, H, I, **J**, B, G, D, L (Alpha-beta performs optimally when what turns out to be the best move is searched first)

**C. (4 points)** Why must the search proceed in a forward direction from the starting position, rather than backward from a **goal position**?

This is an adversary search; there is no single goal **from** which to search, because all opponent's moves must be considered.

**3. (10 points)** Briefly describe the key ideas of the blackboard architecture (viz problem solving framework). Hearsay II is one example of a system built this way, but answer this question in a way that transcends the speech understanding application.

The blackboard is a common place to **post** information. It is divided into layers for different kinds of information. Knowledge sources interact with the blackboard by noticing and posting information. Each knowledge source is independent; they interact only through the blackboard. Control is done opportunistically.

4. (16 points) Consider the problem of whether to put off doing an assignment on a given day. You have the following knowledge:

- An assignment can usually be put off.
- If the assignment is due tomorrow and the due date can't be postponed, then the assignment can't be put off.
- Due dates usually can't be postponed.
- If the assignment is a final paper, then the due date usually can be postponed.
- An assignment is usually not due tomorrow.
- An assignment usually is due tomorrow if you also have to take a test tomorrow

Using the convention below, write a TMS (Truth Maintenance System) style database that represents these facts. For each justification, list the node it supports, and the nodes on its *in list* and *out list*.

The convention: to make grading easier, please use the following letters (optionally preceded by a "¬" for negation) to represent nodes in your database:

- A: The assignment can be put off
- D: The assignment is due tomorrow
- P: The due date can be put off
- F: The assignment is a final paper
- T: You have to take a test tomorrow

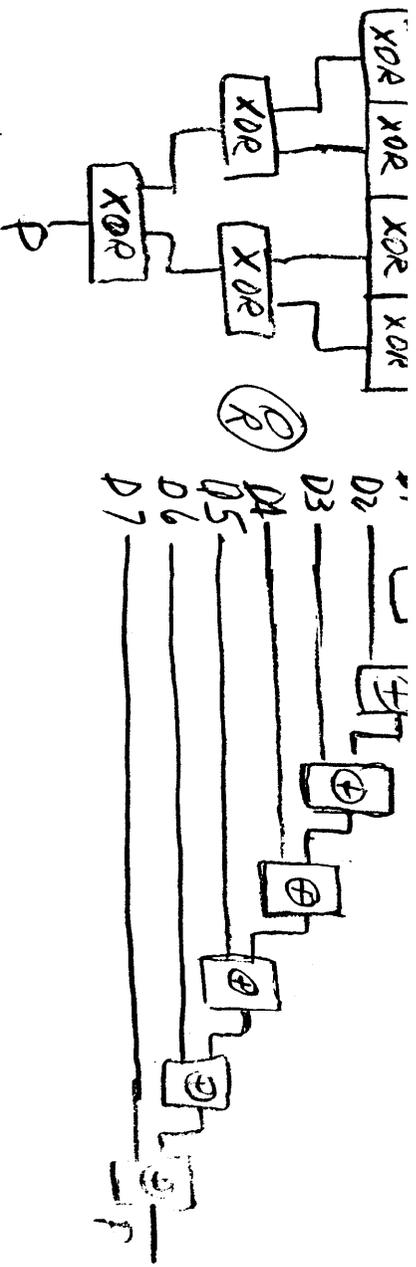
node	in-list	out-list
A		¬A
¬A	D ¬P	
¬P		P
P	F	
¬T		T
D	T	

**5. (5 points)** Constraint propagation is a powerful problem solving method in RI. One of the best known applications is in the Waltz line labeling procedure for scene understanding. What is the reason that constraints were so effective in Waltz's case study?

Because the world works that way. More specifically, the constraints on the possible interpretations of lines that are a consequence of the geometry of vision are sufficient to quickly limit the possible interpretations, and to quickly propagate information. Another way of saying it is that the number of possible labelings under the constraints are very few compared with the number that is combinatorially possible.

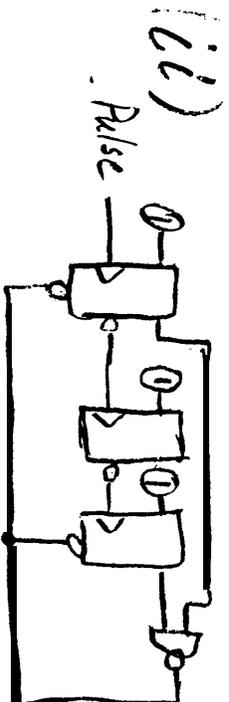
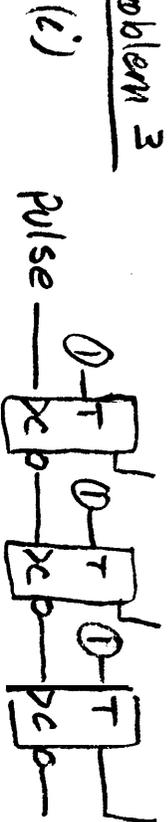
**6. (10 points)** Lenat's AM program discovers concepts in the domain of elementary mathematics concepts. Its overall structure can be described as an application of "classical" AI ideas and methods. Here is your opportunity to so describe it for 10 points. (We're only looking for 10 points worth of answer here, just the essential ideas and methods.)

AM can be described as a classical heuristic search, in this case, looking for "interesting" ideas. It proceeds by the generate and test paradigm. It uses frames to represent its knowledge. Finally, it uses an agenda as its control mechanism. (One student's answer included that AM used lots of CPU time, a common AI technique. By the grace of the examiners, no points were deducted.)

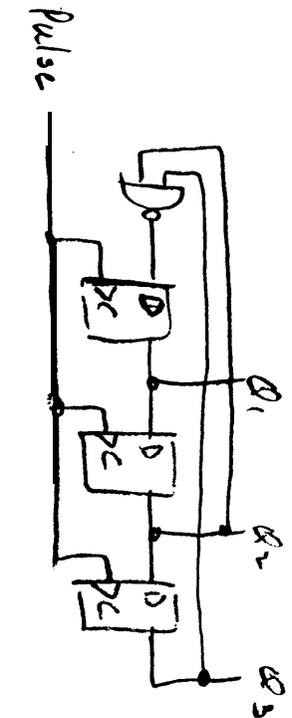


(ii) Any odd number of incorrect bits cause bad parity

Problem 3



iii



Standard  
odd-cycle  
twisted ring  
counter

Caches: *Problem #2*  
-----

(i) a) Associative Mapping -

A Data/Tag pair may occupy ANY slot in the cache. For an "n" slot fully associative cache to operate quickly, it is necessary to have hardware to do "n" tag comparisons in parallel.

b) Direct Mapping -

A Data/Tag pair may only occupy a designated slot in the cache, as determined by some hashing function (usually some low-order bits of the tag field).

c) Set-Associative Mapping -

A Data/Tag pair may occupy some (usually small) set of slots in the cache. A typical implementation uses "n" direct mapping caches (for an "n" set cache). Typically, when a cache slot must have its contents discarded to make room for a new Data/Tag pair, the set is chosen randomly.

One might choose Direct Mapping of Associative Mapping because Direct Mapping requires considerably less hardware.

One might choose Set-Associative Mapping over Direct Mapping because although Set-Associative Mapping requires somewhat more hardware, it has much less trouble with contention for cache slots.

(ii) Read Access:

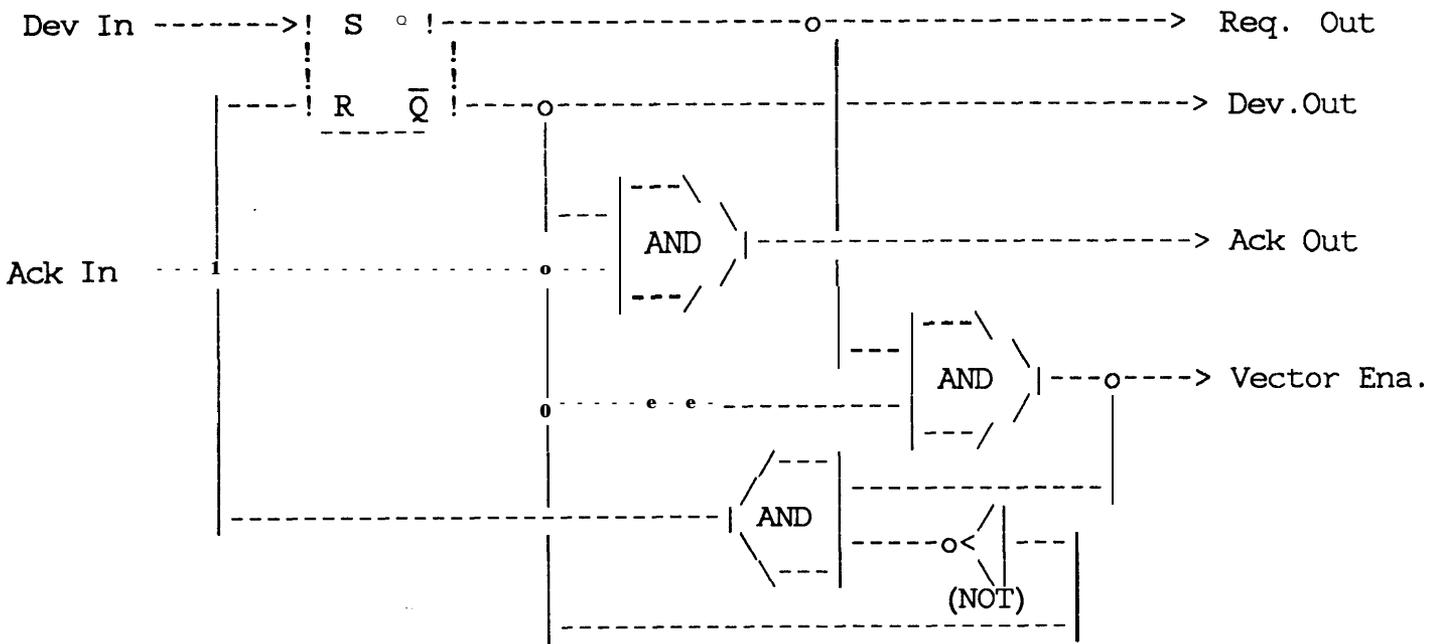
The address of the requested data is used as the input to a hashing function (typically some low order bits of the address) to calculate the slot number in the cache. All sets in the cache are checked in parallel. If the tag part of the entry matches the address of the requested data (in ANY of the sets) the data is returned as the result of the read access. If there is no match, a set is selected for replacement (usually at random). If the selected entry is dirty (i.e. must be written back to main memory) it is written back to main memory. The data is read from main memory and the data and tag are placed in the cache slot, marked as not dirty.

Write Access:

The cache is checked (as above) to see if the data is already there. If a match occurs, the data part of the cache slot is replaced with the write data and is marked dirty. If there is no match, a slot is made free (as above) and the data and tag are written into this now free slot and marked dirty.

(iii) A write-back cache requires fewer main memory write cycles than a write through cache. With a write-through cache there is no problem in maintaining consistency of main memory when multiple processors or DMA Input/Output are used.

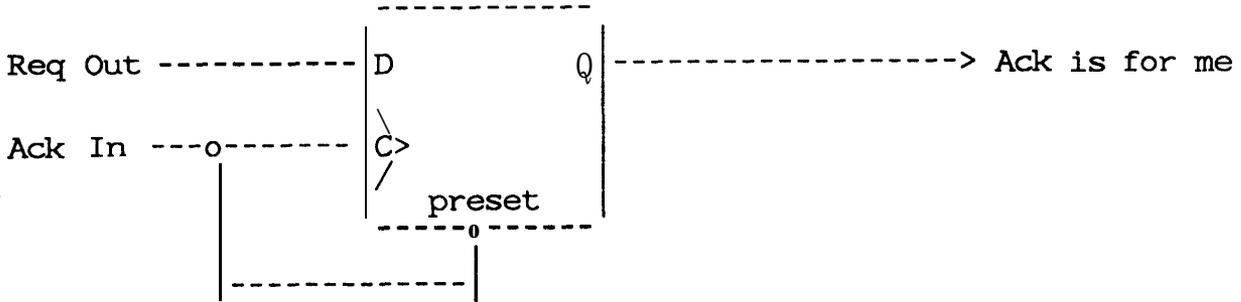
Winter 1985 - Hardware (Solutions)  
 I/O Busses: *Problem #4*



The problem is that higher priority devices (i.e. those closer to the CPU in the daisy chain) may mistakenly interpret an "Ack In" signal as being destined for them even though a lower priority device is in the middle of an interrupt transaction.

One solution is to provide a BUSY line to indicate that a interrupt transaction is in progress.

Another solution is to use edge triggered flip-flops to look for the rising edge of Ack In:



**Numerical Analysis**

1. a) False, b) False, c) True, d) True, e) False.

2. 5.

3.

$$\begin{aligned}(10^{-3} + 1) - 1 &= \mathbf{0} \\ 10^{-3} + (1 - 1) &= \mathbf{10^{-3}}\end{aligned}$$

4. a) There are three ways to do this.

1: Put  $p(x) = ax^3 + bx^2 + cx + d$  into  $I(p)$  and the actual integral and determine  $\xi$ .

2: Put  $p(x) = \text{constant}$ ,  $p(x) = x$ ,  $p(x) = x^2$ , and  $p(x) = x^3$  individually into  $I(p)$  and the actual integral and determine  $\xi$ .

3: The easiest way is to note that i) the interval of integration is symmetric about 0; ii)  $I(f) = 0$  for any odd function  $f$ . iii)  $\int_{-h}^h f(x) dx = 0$  for any odd function  $f$ . So, we only need try  $p(x) = x^2$ .

$$I(x^2) = 2h^3\xi^2, \quad \int_{-h}^h x^2 dx = \left. \frac{x^3}{3} \right|_{-h}^h = \frac{2h^3}{3}$$

now match the two answers

$$2h^3\xi^2 = 2h^3\frac{1}{3} \implies \xi^2 = \frac{1}{3} \implies \xi = \frac{1}{\sqrt{3}}$$

b)

$$I(x^4) = h\left[\frac{2}{9}h^4\right] = \frac{2}{9}h^5, \quad \int_{-h}^h x^4 dx = \frac{2}{5}h^5.$$

The error is

$$\frac{2}{5}h^5 - \frac{2}{9}h^5 = \frac{8}{45}h^5$$

5. a) Quadratic (order 2).

b)

$$f(x) = x^5 - x, \quad f'(x) = 5x^4 - 1$$

**Newton's iteration is**

$$\begin{aligned}
 x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \\
 x_{i+1} &= x_i - \frac{x_i^5 - x_i}{5x_i^4 - 1} \\
 x_{i+1} &= \frac{4x_i^5}{5x_i^4 - 1} = g(x_i)
 \end{aligned} \tag{1}$$

Clearly,  $g(x)$  has a root at 0 of multiplicity 5. This implies that  $g^{(i)}(0) = 0$  for  $i = 0, 1, 2, 3, 4$  (where  $g^{(1)} = g'$ , etc.) and  $g^{(5)}(0) \neq 0$ . The order is 5.

c) Multiply equation (1) above by  $x_i$ :

$$x_i x_{i+1} = \frac{4x_i^6}{5x_i^4 - 1}$$

and you see that as long as  $|x_i| < \sqrt[4]{5}$ , the sign of  $x_i$  is different than the sign of  $x_{i+1}$ . So the convergence is not monotonic.

d) We need to find the point  $r$  where  $-r = g(r)$  (or, equivalently,  $r = g(-r)$ ). By setting  $-r = g(r)$  we obtain  $r = 1/\sqrt[3]{3}$ .

6. a)

$$\int_0^1 \frac{x^n}{x+4} + \frac{4x^{n-1}}{x+4} dx = \int_0^1 x^{n-1} dx = \frac{1}{n}.$$

b) We can show  $y_{n+1} < y_n$ , or  $y_n - y_{n+1} > 0$ :

$$y_n - y_{n+1} = \int_0^1 \frac{x^n(1-x)}{x+4} dx$$

This is an integral over a positive interval with a positive integrand, hence it is positive.

Now, to get  $1/55 < y_{10} < 1/50$  we use the recurrence relation for  $n = 10$

$$y_{10} + 4y_9 = 1/10 \implies y_{10} < 1/50$$

and again for  $n = 11$

$$y_{11} + 4y_{10} = 1/11 \implies y_{10} > 1/55$$

c) The maximum initial error  $|e_0| = |1/55 - 1/50| = 1/550$ . So we have the following iterates:

error in  $y_{10} = e_0$

error in  $y_9 = \frac{1}{4}e_0$

error in  $y_8 = \frac{1}{16}e_0$

error in  $y_0 = \frac{1}{4^{10}}e_0$

and  $4^{10} = 2^{20} \approx 10^6$  so the error in  $y_0$  is approximately  $10^{-6} \times 1/550 \approx 2 \times 10^{-9}$ .

Solutions: **Software Systems**  
 Susan Owicki and **Steve Tjiang**

**Problem 1.**

(a) The productions  $\{A \rightarrow Aa, A \rightarrow a\}$  left **recursively** defines **A**;  $\{C \rightarrow cm, C \rightarrow cn\}$  have a **common left factor**;  $FOLLOW(B) \cap FIRST(B) \neq \{\}$

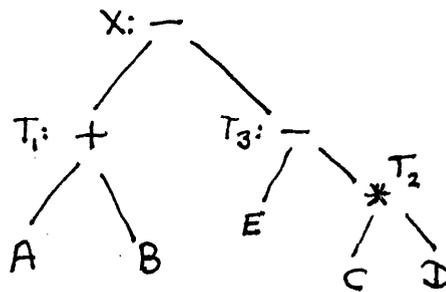
(b) **It is not possible to decide which of the two productions**  $\{B \rightarrow cB, B \rightarrow \epsilon\}$  **to use until one of the strings cm, cn, and cc is seen. This requires a lookahead of two.**

(c) **The language produced by the grammar is**  $a^+c^+\{m|n\}$ . **A five -production grammar for this language is:**

$s \rightarrow aS$   
 $S \rightarrow acL$   
 $L \rightarrow CL$   
 $L \rightarrow m$   
 $L \rightarrow n$

**Problem 2.**

(a)



(b)

ld a,r1  
**a d d** b,r1  
 st r1,t1  
 ld c,r1  
 mul d,r1  
 sub r1,r2  
 ld t1,r1  
 sub r2,r1  
 st r1,x

(c) If the right subtree of the DAG is evaluated first then one can eliminate all uses of temporaries.

```
ld    c,r1
mul   d,r1
ld    e,r2
sub  r1,r2
ld   a,r1
a d d b,r1
sub   r2,r1
st    r1,x
```

An alternative is to recognize that the quadruples calculate the expression

$$x = a + b - (e - c * d).$$

This can be rewritten as

$$x = a + b - e + c * d$$

which yields the following code:

```
ld    a,r1
a d d b,r1
ld    c,r2
mul   d ,r2
add  r1,r2
sub  e,r2
st    r2,x
```

Either solutions were worth full credit although the first solution is the preferred solution.

Problem 3.

• a)

- Two pointers to the same object.
- Computed array subscripts, e.g. A[i] and A[j] are aliases if i = j.
- Passing the same actual parameter to two var parameters in the same procedure.
- Passing a global variable as the actual to a var parameter.

(b) In pure LISP, there is no assignment and no way to change the value bound to a variable. Thus aliasing can have no visible effect. It makes no difference whether two variables refer to the same object or to different copies of the same value.

Problem 4.

Compile time — Pascal and older versions of FORTRAN. FORTRAN can allocate space statically. In Pascal space is allocated on the stack at time of procedure invocation. Pascal cannot use static allocation because it; allows recursion.

**Block entry** — in Algol and Simula array bounds can be declared to be variables which are evaluated at block entry. Space is allocated on the stack.

**Assignment** — Variables in APL, LISP, and various other interpreted languages get the type of the value assigned to them, including the bounds if it is an array. This requires allocation on the heap.

Other answers are possible.

### Problem 5.

(a) Pages used in the recent past are likely to be used again in the near future (temporal locality).

(b) There are two cases to consider. First, a page may be a good candidate for replacement because the program has left the loop where it was being used. If there are no pages meeting this criterion, then the page to be replaced should be the one with the longest time until it can be expected to be used again.

Let  $\text{avg}(i)$  and  $\text{last}(i)$  be, respectively, the average time between accessing location  $i$ , and the time since the last access to  $i$ . The first criterion is satisfied for page  $p$  if

$$\forall i \in p (\text{last}(i) > \text{avg}(i) + \epsilon)$$

If a page has not been used for a time significantly longer than its average, the program has probably left the loop where it was being used. Here  $\epsilon$  is a constant to allow for small variations in the time of executing a loop.

If no page satisfies the first criterion, pick a page with the maximum value of

$$\min_{i \in p} (\text{avg}(i) - \text{last}(i))$$

The expression  $\text{avg}(i) - \text{last}(i)$  is a good predictor of when location  $i$  will be accessed next. Taking the minimum over all locations on a page indicates when that page will be accessed next. So we choose the page that is likely to be accessed furthest in the future.

### Problem 6.

The monitor provides storage for shared variables, mutual exclusion, and the wait/signal primitives. In a message-passing implementation, a process can be used to provide the same facilities. The code for the procedures can be part of the process or can be executed in the calling processes. The first, alternative is used here.

To call procedure **A**, a process sends a message to **M**, the "monitor" process, passing the name of the procedure to be executed and the parameter. It then waits to receive a message indicating that the procedure has been executed; this message contains the output value of the var parameter **a**.

send(M,["A",a])

receive([P,a])

Process M executes an infinite loop in which it waits to **receive** messages and act upon them. A queue is **used** to keep **track of** processes that are waiting **in procedure B**. The version below is **based on the semantics** in which a signal awakens **all processes** waiting on a condition **variable, but** other semantics **are acceptable**. The components of **message m** are accessed as follows

```

m.sender: the process that sent the message
m.choice: the procedure to be invoked
m.data: parameter value (for calls to A only)
var    q: queue;
        x: integer;
        m: message;
s3;
while true do
    while notEmpty(q) and x ≥ 0 do
        remove(q,m);
        S2;
        send(m.sender,[]);
    end {while};
    receive(m);
    if m.choice := "A";
        then a := m.param;
        S1;
        send(m.sender,[a]);
    else if m.choice = "B"
        then put(q,m);
    end {while}

```

**Mathematical Theory of Computation**

**Problem 1: (rev and rev2)**

(a) We prove this statement using complete induction. In order to prove  $\mathbf{rev}(\mathbf{x} * u) = \mathbf{u} * \mathbf{rev}(\mathbf{x})$ , we inductively assume the following hypothesis:

$$\begin{aligned} & \text{(for all string } x') \\ & \text{(for all character } u) \left[ x' < x \supset \mathbf{rev}(x' * u) = u * \mathbf{rev}(x') \right]. \end{aligned}$$

Here,  $x' < x$  can either mean  $x'$  is a proper substring of  $x$  or  $x'$  is shorter in length than  $x$ . Both are well-founded relations. Note that quantification over  $u$  in the induction hypothesis is essential.

There are three cases:

$x = \Lambda$ :

$$\mathbf{rev}(\Lambda * u) = \mathbf{rev}(u) = u = u * \Lambda = u * \mathbf{rev}(\Lambda)$$

$x = a$  is a character:

$$\mathbf{rev}(a * u) = \mathbf{rev}(a * \Lambda * u) = u * \mathbf{rev}(\Lambda) a = u * \Lambda * a = u * a = u * \mathbf{rev}(a)$$

$x = a * x'' * b$ :

$$\begin{aligned} \mathbf{rev}(a * x'' * b * u) &= u * \mathbf{rev}(x'' * b) * a \\ &= u * b * \mathbf{rev}(x'') * a \quad [\text{Induction hypothesis } (u \leftarrow b)] \\ &= u * \mathbf{rev}(a * x'' * b) \end{aligned}$$

(b) We prove the stronger result  $\mathbf{rev2}(x, y) = \mathbf{rev}(x) * y$ . That  $\mathbf{rev2}(x, \Lambda) = \mathbf{rev}(x)$  follows immediately.

In order to prove  $\mathbf{rev2}(x, y) = \mathbf{rev}(x) * y$ , we inductively assume the following:

$$\begin{aligned} & \text{(for all string } x') \\ & \text{(for all string } y) \left[ x' < x \supset \mathbf{rev2}(x', y) = \mathbf{rev}(x') * y \right]. \end{aligned}$$

Again,  $<$  denotes either of the two well-founded orderings mentioned above.

The proof has three cases:

$x = \Lambda$ :

$$\mathbf{rev2}(\Lambda, y) = y = \Lambda * y = \mathbf{rev}(\Lambda) * y$$

$x = a$  is a character:

$$\mathbf{rev2}(a, y) = \mathbf{rev2}(a * \Lambda, y) = \mathbf{rev2}(\Lambda, a * y) = a * y = \mathbf{rev}(a) * y$$

$x = a * x'' * b$ :

$$\begin{aligned} \mathbf{rev2}(a * x'' * b, y) &= \mathbf{rev2}(x'' * b, a * y) \\ &= \mathbf{rev}(x'' * b) * a * y \quad [\text{Induction hypothesis } (y \leftarrow a * y, x'' \text{ t } x'' * b)] \\ &= b * \mathbf{rev}(x'') * a * y \quad [\text{by Part (a)}] \\ &= \mathbf{rev}(a * x'' * b) * y = \mathbf{rev}(x) * y \end{aligned}$$

Winter 1985 - Mathematical Theory of Computation (Solutions)

**Problem 2:** (Satisfiability and Resolution)

- (a) Possible model: Let the domain be the integers and  $p(x, y)$  be  $x < y$ .  
 (b) First, reduce the proposition into clausal form:

$$\begin{array}{ll} \forall xq(x) \equiv \exists x\neg q(x) & \text{sentence} \\ \forall xq(x) \equiv \exists y\neg q(y) & \text{rename variables} \\ (\neg\forall xq(x) \vee \exists y\neg q(y)) \wedge (\forall xq(x) \vee \neg\exists y\neg q(y)) & \text{eliminate } \equiv \\ (\exists x\neg q(x) \vee \exists y\neg q(y)) \wedge (\forall xq(x) \vee \forall yq(y)) & \text{push } \neg\text{'s inward} \\ (\neg q(a) \vee \neg q(b)) \wedge (q(x) \vee q(y)) & \text{clausal form} \end{array}$$

Then resolve:

- 1)  $\neg q(a) \vee \neg q(b)$  given
- 2)  $q(x) \vee q(y)$  given
- 3)  $\neg q(b)$  from (1), (2),  $\{x \leftarrow a, y \leftarrow a\}$
- 4)  $\square$  from (2), (3),  $\{x \leftarrow b, y \leftarrow b\}$

**Problem 3:** (Harder than P and NP)

Any decidable language *much harder* than both NP and co-NP would do. My favorite is the set of valid statements of Presburger arithmetic. No assumptions are needed.

**Problem 4:** (Classes of machines)

All four machines are equivalent in power to a Turing machine, and therefore accept the recursively enumerable (r.e.) languages.

- (a) Nondeterminism adds no power to a Turing machine. See [Hopcroft and Ullman p. 164].
- (b) Two counters are sufficient to give a finite state machine the power of a Turing machine; therefore three counters are more than sufficient. See [Hopcroft and Ullman, p. 172].
- (c) Propositional satisfiability is decidable. The Turing machine therefore gains no additional power.
- (d) This is just a Post machine, which is equivalent in power to a Turing machine. See [Manna p. 27].

**Problem 5:** (Languages)

- (a) (i) The set of all languages containing either three consecutive 1's or 0's is regular since it can be written as the regular expression:

$$(0 + 1)^*(000 + 111)(0 + 1)^*$$

Regular languages are closed under complementation.

- (ii) Assume the language, call it  $\mathcal{L}$ , is regular. Then  $\mathcal{L} \cap 0^*1^*$  would also be regular. But that is just the language  $\{0^n1^n \mid n \geq 1\}$ , which is known not to be regular.

(b) (i) The language  $\{ww^R \mid w \in \Sigma^+\}$  is obviously context free. Context free languages are closed under Kleene star.

There is also a fairly simple grammar for this language.

(ii) This language is easily recognized by a finite state machine with one stack. The machine first pushes a \$ on the stack as an end marker. Every time a 0 is seen, the machine pushes an  $a$  onto the stack. If a 1 is seen, the top of the stack is checked. If the top symbol is a \$, (i.e. the stack is empty), the machine fails; otherwise it pops a symbol off the stack.

**Problem 6:** (Language proficiency)

Ξέρω να γράφω Ελληνικά

אני יודע עברית



## ANALYSIS OF ALGORITHMS

### 1. Data Structures (24 points total)

You want to design a data structure that maintains a large set of records with fixed length keys (i.e., assume keys can be compared and/or hashed in constant time), and supports the operations:

**INSERT:** insert a new record, with duplicate keys allowed

**INSERT-UNIQUE:** insert a record only if its key is not in the set

**DELETE:** delete one record with the given key, if there is one .

**FIND:** locate a record in the set, with the given key

**FIND-NEXT:** locate the record in the set with the next higher key than a given key value

**A. (5 points)** Fill the following table with the average times required by these operations on the standard data structures shown, expressed as a function of  $n$ , the number of records. You can ignore constants and omit the  $O(\ )$ 's, as they are understood in this context.

	(1) Insert	(2) Insert-Unique	(3) Delete	(4) Find	(5) Find-Next
<b>(a) Unordered Linked List</b>					
<b>(b) Binary Search Tree</b>					
<b>(c) Ordered Array</b>					
<b>(d) Hashing with Chained Overflows</b>					

**B. (5 points)** Suppose that in your application there are going to be  $n$  INSERTs with unique keys to create the set of records, followed by  $n$  FINDs in random order. However, the FINDs will reference only  $\log \log n$  distinct keys, not known in advance.

Sketch procedures using and/or adapting standard data structures that minimize (within reason) the average total time for an entire run. State your time in  $O(\ )$  form; you should do better than  $n \log n$ .

**C. (6 points)** Like Part B, except that after the  $n$  INSERTs you are given  $\log \log n$  non-overlapping intervals  $[\ell_i, u_i]$ . After being given all the intervals, you then need to find all records whose keys fall into any of the intervals.

**D. (8 points)** Like Part B, except that after the  $n$  INSERTs you are given  $\log \log n$  non-overlapping intervals  $[\ell_i, u_i]$ . After each interval is given you need to find all records whose keys fall that interval, before you are given the next interval.

**2. Algorithm Efficiency (12 points total)**

The following program computes the  $n$ th "Tribonacci" number:

```
function trib(n);  
begin  
  if n>1 then  
    t := trib(n-1) + trib(n-2) + trib(n-3)  
  else if n=1 then  
    t := 1  
  else  
    t := 0;  
  trib := t; {i.e., return t}  
end.
```

Assume the recurrence equation for the running time  $T(n)$  of this program on input  $n$  is:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + T(n-3) + c && \text{For } n > 1 \\ T(n) &= C && \text{For } n \leq 1 \end{aligned}$$

where  $C$  is a positive constant.

**A. (5 points)** Estimate  $T(n)$  for large  $n$  by obtaining a  $O(\ )$  (upper) bound if  $T(n)$  is polynomial in  $n$ , or by obtaining a  $\Omega(\ )$  (lower) bound if  $T(n)$  is super-polynomial (i.e., exponential) in  $n$ .

**Hint:** Use the fact that  $T(n)$  is a monotonically increasing function to obtain a simpler recurrence that bounds  $T(n)$  in the direction you want.

**B. (7 points)** Describe an algorithm for the same problem that has much better running time, and show the running time in  $O(\ )$  notation.

**C. (0 points)** Was Fibonacci the same person as Leonardo di Pisa?

Spring 1985 - Analysis of Algorithms

3. NP-Completeness (14 points total, 2 points per part)

Place each of the statements A through G below into one of the categories numbered (1) through (5) below:

- (1) true
- (2) true if and only if  $\mathcal{P} \neq \mathcal{NP}$
- (3) true if and only if  $\mathcal{P} = \mathcal{NP}$
- (4) false
- (5) not known to be in one of the above.

- A. Any problem that can be reduced in polynomial time to SATISFIABILITY requires exponential deterministic time.
- B. Any NP-complete problem can be reduced in polynomial time to SATISFIABILITY.
- C. If any problem in  $\mathcal{NP}$  requires exponential deterministic time, then all NP-complete problems do.
- D. Deciding whether a graph has a Hamiltonian Cycle (i.e., a simple cycle that goes through all vertices) is not in  $\mathcal{P}$ .
- E. Deciding whether a graph has a subgraph that has a Hamiltonian cycle is not in  $\mathcal{P}$ .
- F. Deciding whether a propositional (i.e., Boolean) formula is a tautology is in  $\mathcal{P}$ .
- G. Deciding whether a propositional (i.e., Boolean) formula is a tautology is in  $\mathcal{NP}$ .

4. Mathematics (10 points total)

- A. (2 points) How many permutations of the five letters a, b, c, d and e have c followed immediately by d?
- B. (3 points) Let  $f(n) = (\log n)^{\log n}$ . Show (briefly and informally) whether or not  $f(n)$  is polynomially bounded.
- C. (5 points) Evaluate

$$\sum_{j=1}^{\infty} \sum_{k=1}^j k p^{k+j}$$

where  $0 \leq p < 1$ . Recall that, for  $n \geq 0$  and  $0 \leq r < 1$ ,

$$(a) \quad \sum_{i=n}^{\infty} r^i = \frac{r^n}{1-r}$$

$$(b) \quad \sum_{i=1}^{\infty} i r^i = \frac{r}{(1-r)^2}$$

Spring 1985

**Artificial Intelligence      Time: one hour      Total points: 60**

**1. Natural language and knowledge representation (25 pts).**

**a) (13 pts)**

Show an ATN that accepts the following sentences:

Jim is a cognitive scientist.

Cognitive scientists drive sports cars.

A Porsche is an expensive sports car.

An Impala is a sedan.

A sedan is a family car.

If a person drives a family car then he does not drive a sports car

You need **not** bother with **enumerating** the terminal symbols. Your ATN should not accept ill-formed sentences such as:

Jim cognitive scientist.

If a car is a family **car**.

Jim **are** cognitive scientists.

b) (2 pts) Show the parse **tree generated** by your ATN for the **sentence** "Jim drives an old green Impala".

c) (5 pts) A human can **integrate the information in all** the sentences from a) and b) sensibly. Why is **the same not true of a system that uses first-order predicate calculus?** In **what** kind of logic can they be represented?

d) (5 pts) Represent the meaning of the sentences in a) and b) in a semantic network. **What** feature of your network avoids the problem with first-order predicate calculus?

**2. Search (12 pts).**

Describe the behavior of someone driving around a city looking for a **drugstore** (without asking for directions) using :

a) breadth-first search

b) depth-first search

c) hill-climbing

d) best-first search.

Include in your descriptions the (discrete) **states, operators and evaluation functions** as appropriate. Also indicate the pathological behaviors that might arise.

A description of the person's behavior as "desperate" will not suffice.

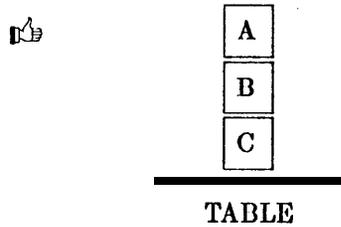
### 3. Blocks world planning (23 pts).

Here we will treat the blocks-world planning problem entirely within first-order predicate calculus with equality. The following relations will be used to describe a situation  $s$ :

- $on(x, y, s)$  Block  $x$  is on block  $y$
- $ontable(x, s)$  Block  $x$  is on the table.
- $clear(x, s)$  The top of block  $x$  is clear.
- $holding(x, a)$  The hand is holding block  $x$ .
- empty hand ( $s$ ) The hand is empty.
- $block(x)$   $x$  is a block.

(In your answers you may abbreviate the above predicates by on, ot, c, h, ch, b respectively).

a) (3 pts) Use these relations (and the symbols  $\forall \exists \wedge \vee \neg \Rightarrow \Leftrightarrow =$ ) to describe the following situation  $s_0$ :



b) (4 pts) Write axioms (with quantifiers) to express

- i) the relationship between **holding** and **emptyhand**;
- ii) the relationship between **on** and **clear**;
- iii) the fact that only one block can be on another.

The actions **pickup(x)**, **puton(x, y)** (put  $x$  down on  $y$ ) and **putontable(x)** are related to situations by the result relation: **result(a, s)** refers to the situation resulting from performing action  $a$  in situation  $s$ .

c) (6 pts) Write axioms defining the three actions in terms of their effects. Bear in mind that the hand only holds one block and can't pick something up unless it's clear, and that a block is on something or in the hand but not both.

d) (2 pt) What is meant by a **frame axiom**? Write one such axiom for this case.

Suppose we now want to use a backward-chaining theorem-prover to find a plan for achieving some condition. The theorem-prover accepts a query in the form of a single proposition (i.e. with no connectives except  $\neg$ ) and attempts to prove that the query can be satisfied using backward reasoning on the available rules; it returns the variable bindings that satisfy the query. You may assume that all the above axioms (and a full set of frame axioms) are available in the correct form.

e) (4 pts) What query could we give the theorem prover in order to find a sequence of actions to achieve the condition that C be on top of B, starting from the situation  $s_0$  shown above? What would the returned term be for the shortest such sequence?

f) (2 pts) Is there an intrinsic difference in the efficiency of forward and backward chaining for a blocks world planner trying to find a sequence of actions to invert the stack in  $s_0$ ? Why (not)?

g) (2 pts) How do actual planning systems avoid the use of frame axioms? What happens to the knowledge contained in those axioms in such systems?

**HARDWARE EXAM.**

**Problem 1 (10 points):**

- i) (4 points)** Diagram and label the fields of a typical floating point number.
- ii) (2 points)** Define “normalized” for floating point numbers.
- iii) (4 points)** In what ways does normalization contribute to precision?

**Problem 2 (15 points):** The VOL-4 is a fictitious X-bit word-addressed machine with registers R0-R7. Instruction operands may be addressed in any of the eight following modes:

Syntax	Mode	Effect
Rn	Register	Rn
(Rn)	Register Indirect	M[Rn]
d (Rn)	Indexed	M[d + Rn]
@d(Rn)	Indexed Indirect	M[M[d+Rn]]
(Rn) +	Autoincrement	M[Rn]; Rn ← Rn+1
@(Rn) +	Autoincrement Indirect	M[M[Rn]]; Rn ← Rn+1
-(Rn)	Autodecrement	Rn ← Rn-1; M[Rn]
@-(Rn)	Autodecrement Indirect	Rn ← Rn-1; M[M[Rn]]

- i) (3 points)** How many bits of an instruction must an operand consume? (Do not count the d in indexed operands.)
- ii) (4 points)** R7 is also the Program Counter (PC), which during execution of an instruction points to the next instruction. We may therefore derive new modes by using R7 in one of the above modes. Give the syntax of the above mode appropriate for each of the following derived modes:
  - a)** Immediate - Operand is the next word (after the instruction).
  - b)** Absolute - Operand is addressed by the next word.
  - c)** PC- Relative - Operand is at PC+d.
  - d)** PC-Relative Indirect - Operand is addressed by the word at PC+d. (Take care not to execute operands.)

There are four instruction types: op; op x; op a; op a,b. Here op is an operation, x is an 8-bit constant, and a and b are operands addressed as above. Each instruction consists of an initial word plus an additional word for the cl of each indexed-mode operand.

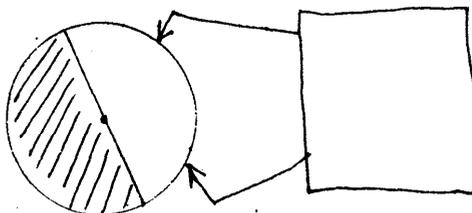
- iii) (4 points)** Propose a layout for the initial word of each type. Show all fields and their widths.
- iv) (4 points)** How many operations of each type do these layouts allow?

**Problem 3 (10 points):**

- i) (3 points)** Define Gray code.
- ii) (4 points)** Give in order the 8 values of a 3-bit Gray code.
- iii) (3 points)** What problem does Gray code solve?

**Problem 4 (10 points):**

- i) (2 points) Define "edge-triggered" for flipflops.
- ii) (8 points) Two detectors are placed 90 degrees apart near the edge of a rotating disk a semicircle of which is black. Each detector outputs logical 1 for black and otherwise 0.



Using nothing but an edge-triggered flipflop, give the circuit diagram for a device that outputs 1 when the disk is rotating clockwise and 0 for counterclockwise. It is acceptable for your output to be delayed by up to one disk revolution.

**Problem 5 (15 points):** Register transfer language (RTL) has the following constructs:

- PC                    program counter
- M[SP]                memory location accessed by the stack pointer
- PSW                   program status word
- 0,1,...                constants
- $X \leftarrow Y$         store Y in X
- $X \leftarrow X \text{ op } Y$    store X op Y in X (op is + or -)

**i) (9 points)** For a machine similar to a PDP-11 or a 68000, describe using RTL the operation of: (a) JSR a (subroutine jump to address a); (b) TRAP v (trap instruction via vector address v); (c) A vectored interrupt with vector address v.

**ii) (6 points)** What distinguishes subroutine calls from traps, traps from interrupts, and interrupts from subroutine calls? Illustrate these distinctions with the intended applications of these capabilities.



4. (15 points) Suppose you have a binary computer with no division available and you want to devise an *iterative* procedure to approximate  $1/a$  for  $a \neq 0$ .
- a) (5 points) Since you will be using an iterative procedure you naturally worry about convergence and the thought of writing code which will converge for every machine-representable number is unpleasant. However, you only need an algorithm that converges for  $a \in [d, 1]$ . The inverse of  $a \notin [d, 1]$  can be easily obtained (with no iteration) from the inverse of some number in  $[d, 1]$ . **Explain why this is so and determine  $d$ .**
- b) (10 points) Describe an iterative algorithm to approximate  $1/a$ ,  $a \in [d, 1]$ , which has quadratic convergence.
- 

5. (15 points) Suppose the roots of the quadratic equation  $ax^2 + bx + c = 0$  are well separated and are representable as machine numbers on your computer. The **direct** application of the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

is generally not a good procedure for obtaining both roots for the following two reasons:

- 1: cancellation,
- 2: overflow.

Both of these deficiencies can be avoided, in this situation. **Show where these deficiencies occur and present a method without these deficiencies.**

**Software Systems**

60 Minutes

**Problem 1.** [15 points]

There are many ways of representing set values in Pascal. One representation is an unsorted list of the elements with no duplicates; another is to use a bit vector where a bit is on if and only if the corresponding element is in the set. In this question, consider only sets of integer subranges.

- (a) [5 points] What are the time tradeoffs between the two representations with respect to the set operations of intersection, union, testing for an element, and testing for equality? Consider also the case of using a list with duplicates.
- (b) [8 points] In a hypothetical Pascal implementation on a 16-bit computer, set values of type “**set of  $l..u$** ” are represented as bit vectors such that  $i \in s$  if and only if  $l \leq i \leq u$  and the  $(i - l + 1)$ 'th bit is set in the bit vector of  $s$ .

Consider the following declarations:

```

var   V1 : set of  $l_1..u_1$ ;
        V2 :-set of  $l_2..u_2$ ;
        V3 : set of  $\min(l_1, l_2).. \max(u_1, u_2)$ ;

```

followed by the assignment

$$\mathbf{v3} := \mathbf{V2} + \mathbf{V1} \quad (*)$$

Assume that our machine has four registers and an instruction set like that described in Chapter 15 of *Principles of Compiler Design*. You may, in fact, assume the existence of any operation codes you like, as long as they do not manipulate more than two words at a **time**.

Give a description of what the code generated by the Pascal compiler for executing (\*) must look like. Your description must be specific and detailed, but need not go to the **extreme** of actually giving the assembler-equivalent of the compiled code.

- (c) [2 points] Modify the set representation so that set union becomes much easier. **Explain** why the new representation has this benefit.

**Problem 2.** [8 points]

Consider the following program written in a hypothetical Pascal extended with procedure variables.

```

program main(input, output);
var   PV: procedure var;
      x: integer;
      procedure P1;
      var   x: integer;
          procedure P2(y: integer);
          begin
              x := y+1
          end;
      begin
          PV := P2
      end;
begin
    P1;
    PV(3)
end.

```

- (a) [3 points] Since Pascal uses static scoping what trouble might one foresee when PV is called?
- (b) [5 points] Give two ways of dealing with this problem by using examples from other languages.

**Problem 3.** [4 points]

Consider the following grammar where S is the start symbol,  $\epsilon$  is the null string, and lower case letters are terminal symbols.

$$\begin{aligned}
 S &\rightarrow aSa \\
 S &\rightarrow bSb \\
 S &\rightarrow \epsilon
 \end{aligned}$$

The above grammar is not LR(1). Construct enough LR(1) items to show that this is indeed the case.

**Problem 4.** [5 points]

Consider the following page reference string:

1	3	2	3	1	2	3	1	4	7	6	5	7	5	6
				↑					↑			↑	↑	
				a					b			c	d	

Suppose a working set strategy is being used with a window size of 5:

- (a-d) [2 points] What is the working set at a, b, c, and d?

- (c) [3 points] If only 4 page frames are available to the process at b, give at least two plausible alternative courses of action? and tell which strategy you think is best and why.

**Problem 5.** [5 points]

The “shortest job first” scheduling strategy for a single processor is optimal in that it gives the minimum average waiting time. Why is the corresponding disk scheduling strategy “shortest seek time first” unwise? Could the same problem apply to processor scheduling? In what circumstances?

**Problem 6.** [8 points]

Consider a multiprogramming system which has dynamic page relocation hardware. Suppose many different programs run at a given time, and that each of these includes in its executable code copies of the library routines it uses. It is desired that the redundancy be eliminated by having just one copy of the entire library in memory, to be shared by all of the programs.

- (a) [2 points] Describe how this might be done. Indicate what restrictions, if any, must be placed on the library routines.
- (b) [3 points] Does your solution allow for changes in the library which may move some of the entry points, without requiring the programs to be run through the link-editor again? If not, how would you modify it to do so?
- (c) [3 points] Does your (modified if necessary) solution suffer from a potential drawback? If so, what is it (i.e., what is the trade-off), and if not, explain why. Recall that some library routines may be very small, or called very frequently.

**Problem 7.** [5 points]

Why is it that many systems provide neither prevention nor avoidance of deadlock for user processes? Be specific about what mechanisms would be necessary and why it may be impractical to provide them.

**Problem 8.** [10 points]

Suppose we replace the **wait/signal** construct of monitors with a single construct **await**(  $B$ ), where  $B$  is a general boolean expression which causes the process executing the **await** to wait until  $B$  becomes true.

- (a) [4 points] Explain why this construct cannot be efficiently implemented in general.
- (b) [6 points] Describe how to translate code using this construct into code using the standard **wait/signal** construct, if  $B$  is restricted to be a local boolean variable  $v$ . Don't forget that the only operations on condition variables are **wait** and **signal** (that is, such variables cannot be examined). You may assume that  $v$  is never passed by reference to any procedure, and cannot be aliased.

**MATHEMATICAL THEORY 0% COMPUTATION**

**1. Language Theory (20 points total)**

Consider the four classes of languages and the four operations on languages shown in the following table:

	(1) Union	(2) Complementation	(3) Kleene Star,	(4) Intersection with a Regular Language
(a) Regular				
(b) Context-Free				
(c) Deterministic Polynomial Time Recognizable				
(d) Recursively Enumerable				

**A. (10 points)** Fill in the above table with “yes” and “no” depicting which classes of languages are closed under which operations.

**B. (10 points)** For each of the following four entries of the table, if it is a “yes” entry give a (very brief) outline of the proof; if it is a “no” entry, give a counterexample (without proof).

(a-2) Regular Sets under Complementation

(b-3) Context-Free under Intersection with a Regular Language

(c-3) Deterministic Polynomial Time Recognizable under Kleene Star

(d-2) Recursively Enumerable under Complementation

## 2. Program Verification (18 points total)

The following program, given integers  $n \neq 0$  and  $m \geq 0$ , computes  $n^m$ :

```

function power(n, in);
begin
  p := 1;
  q := n;
  while m  $\neq$  0 do
    begin
      if odd(m) then p := p * q;
      q := q * q;
      m := m div 2;
    end;
  power := p; {i.e., return p}
end.

```

Here div is integer division (e.g.,  $13 \text{ div } 5 = 2$ ).

- A. (9 points)** State a loop invariant that is suitable for proving partial correctness, and prove that it is invariant.
- B. (6 points)** Prove the loop **terminates**.
- C. (3 points)** Argue that the total correctness of the program follows from (A) and (B).

## 3. Logic (22 points total)

**A. (12 points)** For each of the following two formulae, state whether it is unsatisfiable, satisfiable but not logically valid, or logically valid. If it is either unsatisfiable or logically valid, prove so by resolution. If it is satisfiable but not logically valid, give a model for it **and one for its negation**.

$$\left( \forall x A(x) \Rightarrow \forall x B(x) \right) \Rightarrow \forall x \left( A(x) \Rightarrow B(x) \right) \quad (a)$$

$$\forall x \left( A(x) \Rightarrow B(x) \right) \Rightarrow \left( \forall x A(x) \Rightarrow \forall x B(x) \right) \quad (b)$$

**B. (10 points)** State Gödel's Completeness Theorem and Incompleteness Theorem. You **do not** have to define the terms involved. **If you prefer, state them informally (but precisely).**

## ANALYSIS OF ALGORITHMS COMPREHENSIVE EXAM SAMPLE SOLUTIONS, MAY 1985

### 0. Preface

The solutions below are just what the title says, samples of full credit solutions. They are written in the same detail that we accepted and expected for full credit. (In a few cases, a little *more* detail than we required is given.) Additional explanations, not part of the sample solutions, are given in *slanted* type. We hope this will help takers of future exams to judge how much detail to write in their answers.

WC tried to construct this exam section to test these three basic areas:

- (1) Design of algorithms and use of appropriate data structures to solve particular problems (problems 1B, C, D; 2B)
- (2) Estimation of running times, including recurrence equations, simple asymptotics, and “big-oh” and “big-omega” concepts (problems 1B, C, D; 2A, 4B)
- (3) Knowledge of the elements of NP-completeness (problem 3)

Exam takers who showed little or no competence in these three areas would not achieve the “minimum competence” requirement in AA. The above mentioned problems covered 45 points, and anyone who got half of these points was comfortably above the minimum, while those who got less than one third of these points were in trouble. (The half and third figures are after-the-fact observations, not a policy of this or future committees. Hard and fast rules for minimum competence will probably never be drawn.)

### 1. Data Structures

#### 1A.

	(1) Insert	(2) Insert-Unique	(3) Delete	(4) Find	(5) Find-Next
(a) Unordered Linked List	1	$n$	$n$	$n$	$n$
(b) Binary Search Tree	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$
(c) Ordered Array	$n$	$n$	$n$	$\log n$	$\log n$
(d) Hashing with Chained Overflows	1	1	1	1	$n$

*Remark: Some people thought Ordered Array meant some kind of direct indexing; but they ignored the possibility that the actual keys might not be dense in their range, and that some kind of initialization would be necessary to distinguish data from garbage. Another common mistake was to think FIND-NEXT could assume you had already paid to find the previous key. The problem said “a given key value” and not “a key in the set of records.”*

**1B.** Use a hash table with chained overflows.  $n$  records are inserted in  $O(n)$  time. Each FIND takes  $O(1)$  average time for  $O(n)$  total. Total for job:  $O(n)$ .

For extra efficiency, set a mark bit to 0 in each record as it is inserted, and set that bit to 1 on the first FIND for it; also on the first FIND for the record, move it to the front

of the chain. This all takes  $O(1)$  per FIND, helps avoid worst case disasters, and should give a constant factor improvement when the table is moderately full.

*Remark: Almost full credit for a simple linked list of inserted records with a "move to front" scheme for retrievals, avoiding the  $n \log n$  cost of sorting the set. As was hinted, the main idea of parts B, C, and D was to beat the cost of sorting the whole set; however, some part credit was given for  $n \log n$  schemes. Some people forgot that they had to pay for inserts as well as retrievals, and some overlooked that a total of  $n$  retrievals were to be done in part B.*

**1C.** Let  $m = \log \log n$ . On INSERTs, just store the records, say in a linked list, or unordered array, in time  $O(n)$ . Build a binary search tree of the requested intervals in time  $O(m \log m)$ . (The intervals have a natural ordering.) For worst-case insurance, you can make it balanced, or use an ordered array. Finally run through the set of records once, and for each record, search it down in the interval-tree, in time  $O(\log m)$  per record. Output it iff it is in a requested interval. Total time for the retrieval dominates the job, and is  $O(n \log m)$ .

*Remark: Half credit for putting the requests in a list instead of a BST or ordered array, obtaining  $O(nm)$  time.*

**1D.** Let  $m = \log \log n$ . On INSERTs, just store the records in a linked list. This takes time  $O(n)$ .

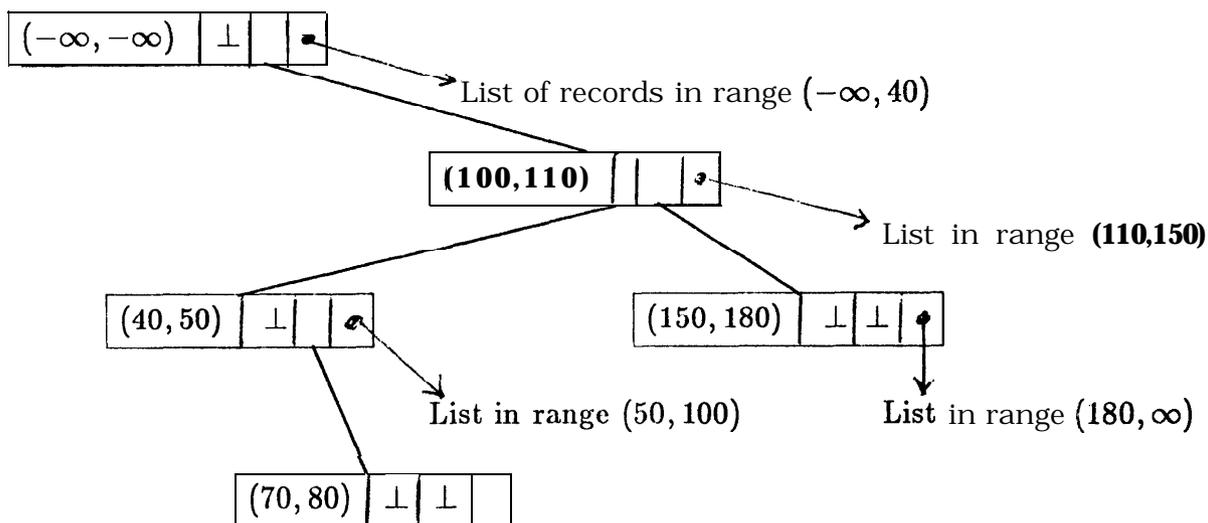
Start a binary search tree with a root node having interval  $(-\infty, -\infty)$ , null left and right node pointers, and a list pointer pointing to the head of the above list of all records. A general node in the BST will have an interval entry, left and right node pointers, and its list pointer. The idea is to maintain the list pointer so that it points to a sub-list that contains just the records higher than this node's interval, and lower than the next higher interval requested so far.

As each interval request arrives, insert it into the BST and locate the one sub-list of records that might fall into this new interval. The list pointer of the node immediately to the left of the new node points to the sub-list we want. Run through this sub-list, putting records in the interval, and building two new sub-lists of records above the interval and below the interval. The "above" sub-list becomes the new node's list pointer and the "below" sub-list replaces the original sub-list in the node to the left.

The example on the next page shows the situation after a new node for interval  $(70, 80)$  has been inserted, but before the list in range  $(50, 100)$  has been run through and subdivided. When the run-through is finished the  $(40, 50)$  node will point to a  $(50, 70)$  sub-list and the  $(70, 80)$  node will point to a  $(80, 100)$  sub-list (and of course, records in the range  $(70, 80)$  will have been returned as output).

The run-through takes time proportional to the size of the sub-list. But the  $k$ -th request will be processed against one of  $k$  sublists whose sum of lengths is no greater than  $n$ . Therefore the average time for the  $k$ -th request is  $O(n/k)$ . The average for all  $m$  requests is  $O(nH_m)$ , where

$$H_m = \sum_{k=1}^m \frac{1}{k} = O(\log m)$$



This dominates the time for the job, so the total time is  $O(n \log m)$ .

*Remark:* This was one of the two hardest questions on the exam, and no one got the full idea. A considerably less verbose answer would receive full credit if it had the correct ideas. If you got  $O(n \log m)$  in part C, then half credit for running through the whole list of  $n$  records on each request, obtaining  $O(nm)$  time; however, only one quarter credit for this essentially trivial solution if you also gave the trivial solution (or  $O(n)$  equivalent) for part C that was mentioned above.

## 2. Algorithm Efficiency

A.  $T(n) \geq 3T(n-3) + C$  for  $n > 1$ , so  $T(n) \geq U_n$ , where

$$\begin{aligned} U_n &= 3U_{n-3} + C && \text{for } n > 1 \\ U_n &= c && \text{for } n \leq 1 \end{aligned}$$

We know  $U_n > 3^{\lfloor \frac{n}{3} \rfloor} C$ , so  $T(n) = \Omega(3^{\frac{n}{3}})$ .

*Remark:* The two most common errors were getting an upper bound instead of a lower bound, and replacing  $\Omega(3^{\frac{n}{3}})$  by  $\Omega(3^n)$ .

B. When  $n > 1$ , initialize  $(t, u, v)$  to  $(1, 0, 0)$ . Then for  $i = 1$  thru  $n-1$  "simultaneously" update

$$(t, u, v) := (t + u + v, t, u)$$

That is, maintain the three most recent values of trib in  $(t, u, v)$ . At the end,  $t$  holds  $\text{trib}(n)$ .

Each cycle in the loop takes constant time, so the procedure takes  $O(n)$ .

*Remark:* Almost everyone got this. We took off 1 point for using an array, as you don't know how big  $n$  might be.

C. Fibonacci was the same person as Leonardo di Pisa.

### 3. NP-Completeness

A. 4 (false)

Remark: *Reductions* go the *other* way around.

B. 1 (true)

Remark: In fact, so can **any problem** in NP.

C. 1 (true)

Remark: *This* is what “complete” *means in* complexity.

D. 2 (iff  $\mathcal{P} \neq \mathcal{NP}$ )

Remark: *I.e., Hamiltonian Cycle* is *a* hardest problem in NP.

E. 4 (false)

Remark: *This just* means, *does the graph have* a cycle *at all?*

F. 3 (iff  $\mathcal{P} = \mathcal{NP}$ )

Remark: Recall that “tautology” is the complement of “satisfiability,” hence is in co-NP. If we have a polynomial time algorithm that runs in  $T(n)$  time for either problem, we can solve the complement by running the same algorithm for time  $T(n) + 1$ ; if it hasn’t succeeded yet, the complement is true.

G. 5 (not known)

Remark: One of the hardest problems on the exam. If “tautology” were in JVP, then co-NP would be the same class as NP. But recall that to be in NP the problem must be polynomially solvable with a polynomial number of “lucky guesses.” For tautology the guesses would be the correct proof steps. But soznc problems (i.e., families of tautologies) have no known proofs of polynomial length, even with lucky guessing. On the other hand, no one has proven that such proofs can’t exist, so the problem is open.

### 4. Mathematics

A.  $4! = 24$ .

B.  $\log f(n) = (\log n) \log(\log n)$ . I.e.,  $f(n) = n^{\log \log n}$ . Since  $\log \log n$  is not bounded by any constant K,  $f(n)$  is not polynomially bounded.

C. Interchange the order of summation in order to make both upper limits  $\infty$ , and match the given formulas.

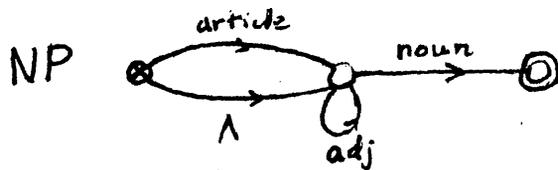
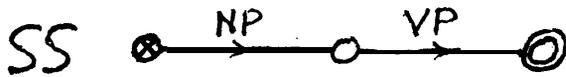
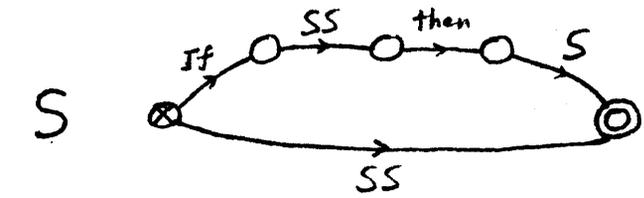
$$\sum_{j=1}^{\infty} \sum_{k=1}^j k p^{k+j} = \sum_{k=1}^{\infty} k p^k \sum_{j=k}^{\infty} p^j = \sum_{k=1}^{\infty} k p^k \left( \frac{p^k}{1-p} \right) = \frac{1}{(1-p)} \sum_{k=1}^{\infty} k (p^2)^k = \frac{1}{(1-p)(1-p^2)^2}$$

Remark: The answer is actually  $\frac{p^2}{(1-p)(1-p^2)^2}$  because the second hint omitted an  $r$  in the numerator of its right side. As we said at the outset, these are samples of full credit solutions!

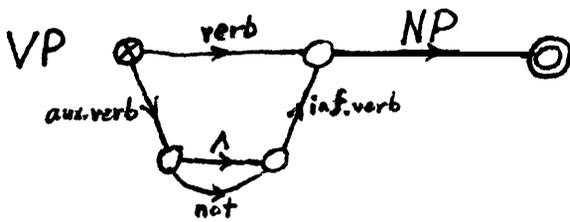
Artificial Intelligence Time: one hour Total points: 60

1. Natural language and knowledge representation (25 pts).

a)

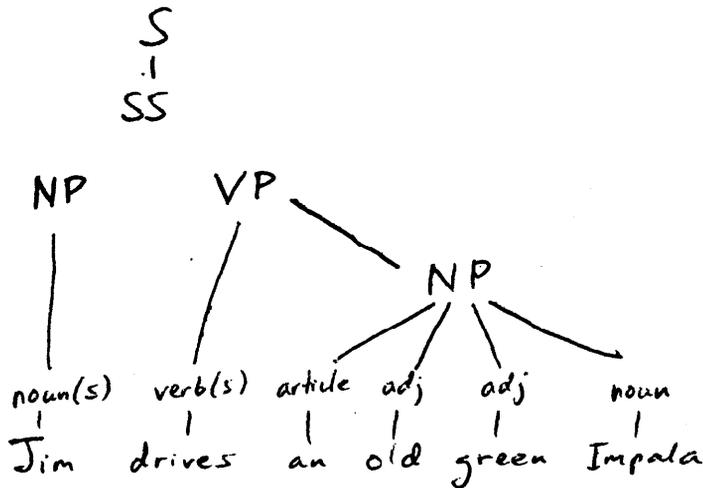


Noun arc sets register to s. or p.



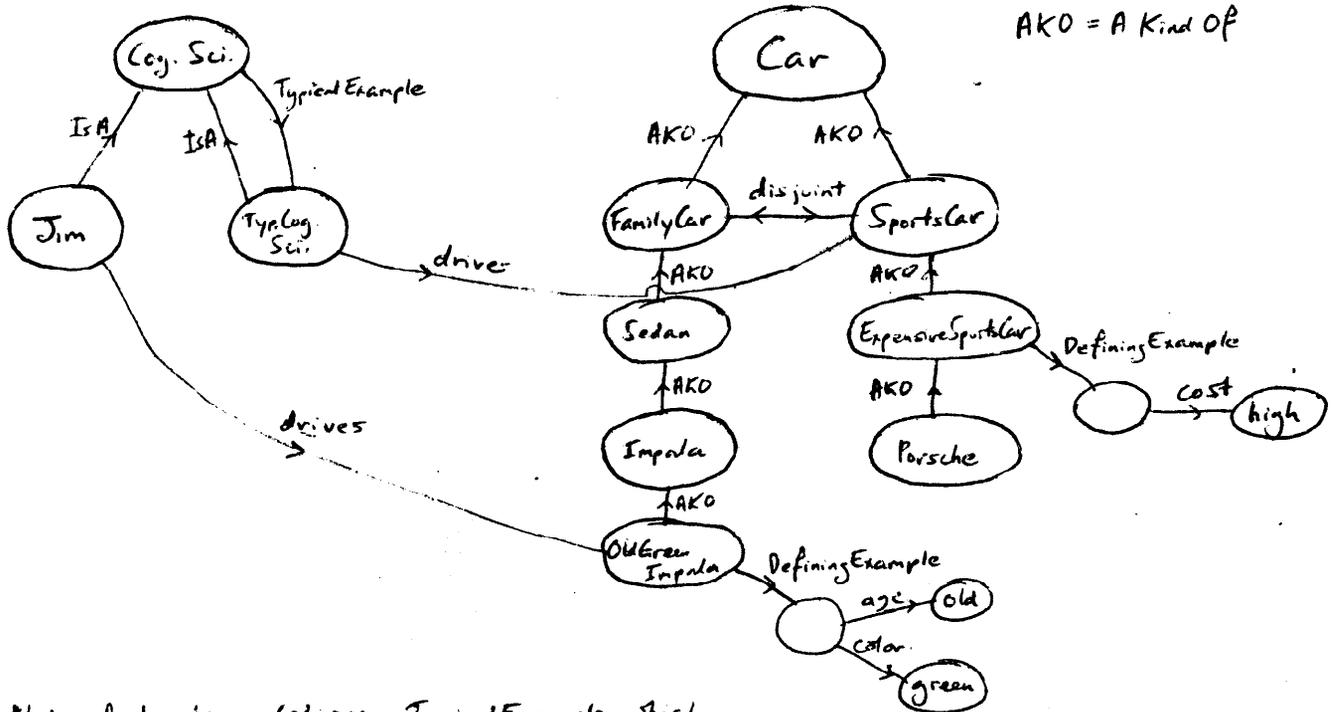
Verb/aux.verb arc tests own number for equality to register

b)



c) The sentence in b) is inconsistent with those in a). In a logical system, contradictions are difficult to handle; for example, in a resolution system they allow one to infer that every possible sentence is true. A human usually treats such sentences as "Cognitive scientists drive sports cars" as giving the normal, or default state of affairs. This is adequately described using default or non-monotonic logic, which allows one to give rules to the effect that unless we know otherwise, any cognitive scientist drives a sports car.

d)



Note distinction between TypicalExample, which has default properties, and DefiningExample, which has necessary properties.

The normal inheritance mechanism for finding out what kind of car Jim drives will go up to the containing class (cognitive scientists) and then take the *Drives* value from its prototype TypCogSci, which is a sports car. But when we add a *Drives* link specifically for Jim, the value retrieval will use it rather than try to inherit a value.

2. Search (12 pts).

The states of the search will be intersections and (on long stretches) intermediate points, chosen such that a search through all states is guaranteed to find the drugstore if there is one. The operators are the transitions from one state to another, and thus all we need is an operator that takes us from one intersection to an adjacent one. The start state is wherever the person starts, and the goal states are those intersections within one block of a drug store.

a) Breadth-first search

In a breadth-first search states are searched in increasing order of distance from the start state, i.e. we visit next the successors of the least recently visited unexpanded node. Thus we search all intersections within

one block, then all intersections within two blocks, and so on. A reasonable way to do this would be to drive in a spiral outward from the starting point. Being a blind, exhaustive search, this method has no applicable evaluation function. There are no pathological behaviours.

b) Depth-first search

Here we visit next the successors of the most recently visited unexpanded node. Depending on the order of visiting successors of a given node, which is not part of the definition of depth-first search, we would follow a spiral path (if the order of successors is, say, left before straight before right) just as in breadth-first, or go straight out to 'infinity' (if straight on takes precedence and we miss the drug store). Most depth-first searches use a depth bound (such as the edge of town) so we would reach the edge, back up a block and go left to the edge, back up again (two blocks if necessary) and so on until a whole quadrant was searched and we had backed up to the start, then go off again in a new direction all the way to the edge. There is no evaluation function involved, but without a depth bound we can easily drive straight off the map.

c) Hill-climbing

In hill-climbing we use an evaluation function to estimate the 'promise of a node, i.e., its likelihood of being close to a drug store. From each intersection, we go to the adjacent intersection with the highest evaluation, but only if higher than where we are (this may entail driving to each in turn to evaluate it). The precise behaviour thus depends on the evaluation function. A reasonable one would give weight to the number of nearby stores, closeness to a major road, shopping centre or housing complex and visibility of a drugstore sign. Thus starting at home in a residential area, we would drive toward an intersection with a major road, then along the road in the direction of increasing store density until a mall was reached, then hopefully we would be able to see a drugstore sign. Pathological behaviours include sitting in the middle of a homogeneous neighbourhood with no worthwhile direction to go (the 'plateau' effect) and getting stuck at a shopping centre without a drug store (the 'local maximum effect).

d) Best-first search

In a best-first search we always expand the unexpanded node with the highest evaluation of all the nodes we have visited. The same evaluation function as in c) could be used here. Because we always visit the *globally* best node, rather than the *locally* best, we avoid the problems with local maxima and the search also will keep going even when it is reduced to examining less promising nodes. For example, after reaching a shopping centre without a drug store (which we would be able to search thoroughly), we would go back to some promising side street we passed on the way and search that. The search can become pathological in the sense that it can hop all over town visiting one promising node here, another there rather than persevering locally.

Probably the best strategy would be to use a best-first search with an evaluation function that gave weight to closeness to recently visited nodes, thus helping to avoid inefficiently hopping around. Whether such context-sensitive evaluation functions are part of pure best-first search is a matter of taste.

**3. Blocks world planning (23 pts).**

a)

$block(A) \wedge block(B) \wedge block(C) \wedge$   
 $on(A, B, s_0) \wedge on(B, C, s_0) \wedge ontable(C, s_0) \wedge$   
 $clear(A, s_0) \wedge emptyhand(s_0)$

b)

- i)  $\forall s[(\exists x.holding(x, s)) \Leftrightarrow \neg emptyhand(s)]$
- ii)  $\forall y, s[(\exists x.on(x, y, s)) \Leftrightarrow \neg clear(y, s)]$
- iii)  $\forall x, y, z, s[on(x, z, s) \wedge on(y, z, s) \Rightarrow x = y]$

c)

- i)  $\forall x, s[emptyhand(s) \wedge clear(x, s) \Rightarrow holding(x, result(pickup(x), s))]$
- ii)  $\forall x, y, s[holding(x, s) \wedge clear(y, s) \Rightarrow on(x, y, result(puton(x, y), s)) \wedge emptyhand(result(puton(x, y), s))]$
- iii)  $\forall s, s[holding(x, s) \Rightarrow ontable(x, result(putontable(x), s)) \wedge emptyhand(result(putontable(x), s))]$

To be more accurate, we could also add conjuncts to the preconditions of these actions stating that  $x$  and  $y$  are blocks.

d) A frame axiom states that some fact remains *unchanged* in the situation resulting from an action. For example, we want to say that when a block is put on the table, any block which is on another block stays there:

$$\forall x, y, z, s[on(x, y, s) \Rightarrow on(x, y, result(putontable(z), s))]$$

e) If we have just  $s_0$  in the database, we just need to ask what situation would have C on B, thus the query is  $on(C, B, s)$ ? where  $s$  is a free variable. From this query the system chains back through the action rules to see what situation could lead to the desired one, until we reach a situation which satisfies the description of  $s_0$ . The first such step would presumably use the *puton* rule to produce the goal of finding a situation  $s_1$  such that  $holding(C, s_1) \wedge clear(B, s_1)$ , where  $s = result(puton(C, B), s_1)$ . The returned binding for  $s$  would be

$$result(puton(C, B), \\ \quad result(pickup(C), \\ \quad \quad result(puton(B, A), \\ \quad \quad \quad result(pickup(B), \\ \quad \quad \quad \quad result(putontable(A), \\ result(pickup(A), s_0))))))$$

which shows the required actions in reverse order.

f) Since the state space is exactly symmetrical about a line drawn halfway between the two states, there can be no difference in efficiency. The forward search looks identical to the backward search with A and C reversed.

g) Most planners use the STRIPS formalism in which actions have add/delete lists which specify the changes to be made to the global state description. The frame axioms are now implicit in the procedure which calculates the new state, since it assumes that only the things given in the add/delete list actually change.

Spring 1985 - Hardware Systems (Solutions)

[10] 1. [4] (a) Diagram and label the fields of a typical floating point number.

-----  
|sign|exponent|mantissa|            - other similar formats exist  
-----

[2] (b) Define "normalized" for floating point numbers.

The mantissa is left justified.

[4] (c) In what ways does normalization contribute to precision?

Primarily by utilizing the maximum number of digits of the mantissa; secondarily in some formats by omitting the leading bit (radix 2 only).

[15] 2. The VOL-4...

[3] (i) How many bits of an instruction must an operand consume? (Do not count the d in indexed operands.)

6

[4] (ii) R7 is also the Program Counter (PC), which during execution of an instruction points to the next instruction. We may therefore derive new modes by using R7 in one of the above modes. Give the syntax of the above mode appropriate for each of the following derived modes:

- (R7)+            a) Immediate    - Operand is the next word (after the instruction).
- @(R7)+        b) Absolute     - Operand is addressed by the next word.
- d(R7)           c) PC-Relative - Operand is at PC+d.
- @d(R7)        d) PC-Relative Indirect - Operand is addressed by the word at PC+d.

[4] (iii) Propose a layout for the initial word of each type. Show all fields and their widths.

[4] (iv) How many operations of each type do these layouts allow?

Since "op a,b" is the format that will have the least number of opcodes, it is probably wise to attempt to maximize the number of opcodes for this format. A variable length type field does this:

Spring 1985 - Hardware Systems (Solutions)

-----  1 1 1 op(13)  -----	8192
1 0 op(6) x(8)  -----	64
1 1 0 op(7) a(6)  -----	128
0 op(3) a(6) b(6)  -----	a

Another possibility is to use a fixed-length type field:

-----  type(2) op(14)  -----	16384
type(2) op(6) x(8)  -----	64
type(2) op(8) a(6)  -----	256
type(2) op(2) a(6) b(6)  -----	4

[It is not necessary for the type fields all to be the same length, e.g. one might have lengths 4,2,3,1 in that order. It is not even necessary to have a type field per se, just so long as the limit of  $2^{16}$  possible initial words is not exceeded for all instructions combined.]

[10] 3. [3] (i) Define Gray code.

Gray code is reflected binary. More exactly, let  $G(n)$  denote the list of the  $2^n$   $n$ -bit words of Gray code in order, let  $R(L)$  and  $L@M$  denote the reverse and append operations on lists, and let  $x\#L$  denote the result of prepending  $x$  to every word in  $L$ . Then  $G(0) = (e)$  (a list with just the empty word  $e$ ) and  $G(n+1) = (0\#G(n))@(1\#R(G(n)))$ .

Equivalently, Gray code begins with a word of all 0's, and the XOR of consecutive words is obtained from the XOR of consecutive words in standard binary by zeroing all 1's save the leftmost in each word.

In view of the brevity accorded this topic in the references on the reading list, it was considered acceptable merely to define Gray code to be a code in which consecutive words differed in only 1 bit (sometimes called cyclic codes). Note however that while there are 144 cyclic codes of 3 bits there is only 3-bit Gray code.

[3] (ii) Give in order the 8 values of a 3-bit Gray code.

000 001 011 010 110 111 101 100

Spring 1985 - Hardware Systems (Solutions)

[4] (iii) What problem does Gray code solve?

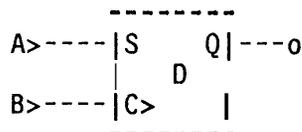
The problem is that for some codes skew produces meaningless intermediate values during transitions between consecutive values. Example: when 011 changes to 100 in ordinary binary the skew may be such that the left bit changes first, leading to a meaningless intermediate value of 111. This is important whenever sampling and transition are mutually asynchronous.

[10] 4. [2] (i) Define "edge-triggered" for flipflops.

The flipflop samples its inputs only at clock transitions.

[8] (ii) Two detectors are placed 90 degrees apart near the edge of a rotating disk a semicircle of which is black. Each detector outputs logical 1 for black and otherwise 0.

Using nothing but an edge-triggered flipflop, give the circuit diagram for a device that outputs 1 when the disk is rotating clockwise and 0 for counterclockwise. It is acceptable for your output to be delayed by up to one disk revolution.



[15] 5.

[9] (i) For a machine similar to a PDP-11 or a 68000, describe using RTL the operation of:

(a) JSR a (subroutine jump to address a);

```
SP <- SP-1
M[SP] <- PC
PC <- a
```

(b) TRAP v (trap instruction via vector address v);

```
SP <- SP-1
M[SP] <- PSW
SP <- SP-1
M[SP] <- PC
PSW <- M[v+1] (or some way of modifying PSW)
PC <- M[v]
```

(c) A vectored interrupt with vector address v.

As for (b)

Spring 1985 - Hardware Systems (Solutions)

[6] (ii) What distinguishes subroutine calls from traps, traps from interrupts, and interrupts from subroutine calls? Illustrate these distinctions with the intended applications of these capabilities.

Subroutine calls differ from traps and interrupts by remaining in user state and permitting branching to a location of the programmer's choice; traps and interrupts (usually) enter system state, and branch to system-determined locations not under the programmer's control. Interrupts differ from traps and subroutine calls in that they are asynchronous: their timing is not linked to that of the suspended computation.

Subroutines provide for structured programming and library routines. Traps provide controlled access to privileged routines, both for system calls and for exception handling. Interrupts serve to temporarily divert the processor's attention to a device urgently needing servicing.



3. Set  $\mathbf{A}(h) - \mathbf{A} = E(h)$  where  $\mathbf{A}$  is the "exact" solution to whatever problem we are solving, then

$$\mathbf{A}(h_1) = \mathbf{A} + c_1 h_1^2 + \mathcal{O}(h^4)$$

$$\mathbf{A}(h_2) = \mathbf{A} + c_1 h_2^2 + \mathcal{O}(h^4)$$

and we are trying to get  $c_1$ . Subtract  $\mathbf{A}(h_1)$  from  $\mathbf{A}(h_2)$  and divide by  $h_2^2 - h_1^2$  to get

$$c_1 \approx \frac{\mathbf{A}(h_2) - \mathbf{A}(h_1)}{h_2^2 - h_1^2}$$

The accuracy of our estimate to  $c_1$  can obviously be expected to be no better than  $\mathcal{O}(h^4)$  and, in fact, is actually only  $\mathcal{O}(h^2)$  since we have to divide by something of order  $h^2$  to obtain  $c_1$ .

4.

- a) For the binary machine mentioned  $d = \frac{1}{2}$  since any floating point number is  $\pm m \times 2^e$  ( $m \in [\frac{1}{2}, 1]$ ). The inverse is  $\pm \frac{1}{m} \times 2^{-e}$ .
- b) Use Newton's method on  $f(x) = \frac{1}{x} - a$ . The iteration formula is

$$x_{i+1} = x_i(2 - ax_i)$$

which involves no division.

5.

- 1: Cancellation could occur in the calculation of the numerator of the formula. We should use the formulae

$$x_1 = -\frac{b + \text{sign}(b) \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$

The second formula follows from noting that  $ax^2 + bx + c = a(x - x_1)(x - x_2)$  implies  $ax_1x_2 = c$ .

- 2: Overflow could occur in the calculation of the discriminant. Even though  $d = \sqrt{b^2 - 4ac}$  may be representable the intermediate calculation of  $b^2 - 4ac$  may overflow. There are a few ways to solve the problem. One way is

$$d = \begin{cases} |b| \sqrt{1 - (4ac/b)^2}, & |4ac| \leq |b| \\ 4|ac| \sqrt{1 - (b/(4ac))^2}, & |b| \leq |4ac| \end{cases}$$

## Spring 1985—Software Systems (Solutions)

1(a)

	bit	list
intersection, union, equality	$O(l)$	$O(nm)$
testing for an element	$O(1)$	$O(n)$

where  $l$  is the length of the bit vector,  $n$  and  $m$  are the number of elements in the operand sets.

(b) Without loss of generality, one can assume that  $l_1 \leq l_2$ , and the zeroth bit of a word is the highest order bit. The the following procedure can be used.

1. Move and shift set V2 to the final result location **V3**. Let  $k$  be  $(l_2 - l_1) \text{ div } 16$  and  $r$  be  $(l_2 - l_1) \text{ mod } 16$ . Set the zeroth through the  $k - 1$ th words of V3 be zero. The  $k$ th word is the first word of V2 shifted to the right  $r$  bits. For  $k < i < (\max(u_1, u_2) - l_1 + 15) \text{ div } 16$ , the  $i$ th word of V3 is the lower order 16 bits of the  $r$  bit right shifted result of the 32 bit word formed from the  $(i - k)$ th and the  $(i - k + 1)$ th words of V2. We assume that when a word outside the extent of V2 is requested in the above calculation, the value of zero is used instead.

2. Now V1 is or'ed word by word with V3 to obtain the correct result. Again we assume that any words outside of the extent of V1 are zero.

(c) One possible simplification is to start all sets on multiples of sixteen. This would simplify the shifting operation above as only word moves would be required.

2(a) The  $x$  referenced in **P2** is in the scope of **P1** but when PV **is** called there are no activations of **P1**.

(b) In C, nesting of procedures are not allowed and in LISP, dynamic scoping is used.

3(a)

$I_0:$	$S \rightarrow .aSa, \uparrow$
	$S \rightarrow .bSb, \uparrow$
	$s \rightarrow .\epsilon, \uparrow$
$I_1 = \text{goto}(I_0, a):$	$S \rightarrow a.Sa, \uparrow$
	$S \rightarrow .aSa, a$
	$S \rightarrow .bSb, a$
	$S \rightarrow .\epsilon, a$

There is a shift/reduce conflict in the second and last productions of  $I_1$ .

## Spring 1985-Software Systems (Solutions)

4(a) {1, 3, 2}

(b) {2, 3, 1, 4, 7}

(c) {4, 7, 6, 5}

(d) {7, 6, 5}

(e) One alternative would be let the program run with only four page frames. An LRU strategy could be used to decide which frame to discard (frame 2 in this case). An alternative strategy is to suspend the program completely until enough page frames are available to accommodate the entire working set.

(Remember that we have no way of predicting the program's future behavior, other than its past behavior. We can't assume that the size of the working set is going to decrease soon, although that happens to be the case in the example we are looking at.) If we force the program to run without all of its working set available, we are likely to encounter "thrashing." The program will be making regular references to all the members of its working set, but one fifth of those references will generate faults, requiring the replacement of some other page in the set with the new page. The more the program faults, the lower the CPU utilization becomes, encouraging the system to run more programs and make even fewer page frames available. If we suspend the program entirely, however, then we avoid thrashing and keep the CPU utilization high. This is clearly the preferred strategy.

### Problem 5

Like all priority-based scheduling techniques, shortest-job-first and **shortest-seek-time-first** are potentially susceptible to starvation. Starvation occurs when a steady stream of high-priority requests is received, such that low-priority requests never get serviced.

In the case of disk scheduling, this scheduling technique can easily lead to starvation. Most disk request patterns show **locality of reference**, such that requests which are close to each other time-wise are also close to each other on the disk. This means that it is easy to generate a continuous stream of requests with a very short seek time, thereby starving other requests.

In the case of CPU scheduling, shortest-job-first is less likely to cause starvation unless a steady stream of jobs with short CPU requests enters the system. If a flood of very short jobs did arrive, then longer jobs would be starved.

**6(i)** Some portion of the user process address space must be reserved for the library, which will occupy the same virtual address in every user process (actually, if the processor supports completely relocatable code then the library can occupy different portions of different processes' address spaces). When user programs are linked, the linker substitutes a known address in the library space for references to library routines (or a known offset from the beginning of the library space if the library doesn't have a fixed location in the user address space).

The library routines must be pure (that is, they cannot modify any part of themselves) and reentrant (it must be possible for any number of processes to be running in a library routine). This might be done by making all data references refer to a per-process stack.

## Spring 1985-Software Systems (Solutions)

(b) The approach described above does not, because the binding from library routine reference to entry point is made by the linker. If the size of any module in the library were changed, all the programs would have to be relinked to reflect the new entry points.

There are two ways we might avoid this. One approach would be to set up at the very beginning of the library a table indicating the entry points of all the routines. Then library references would be resolved to point to a location in the table which would not change, and the actual entry to the library would be made by an indirect call. The second scheme would be to simulate a segmented architecture, by allocating enough space for each library module to accommodate its maximum possible size, effectively putting it in its own "segment." This amounts to fixing the entry points in the user address space, so changes do not cause them to move.

(c) Both of the above solutions suffer from potential drawbacks. The first approach, using indirection, adds a level indirection to every library routine reference from the user program. Since some routines may be called very frequently, this could noticeably affect performance. The second approach, simulating a segmented architecture (or using one if it's available) suffers from waste due to fragmentation. By allocating one "segment" to each module, we may rapidly use up segment name space, while at the same time wasting a lot of user address space.

### Problem 7

Deadlock prevention consists of disallowing at least one of the four essential conditions for deadlock: mutual exclusion, hold and wait, no preemption, and circular wait. It is easy to see why no practical system can eliminate mutual exclusion. Elimination of hold and wait is possible, but it generally results in poor resource utilization and is subject to starvation. Allowing preemption can be very expensive – usually this means some way to restart a process after its resources have been preempted. Also, some resources which require mutual exclusion cannot safely be preempted (for example, a tape drive). To eliminate circular wait, we must place arbitrary restrictions on the order in which resources may be requested. This will generally result in poor resource utilization, as resources will be requested long before they are used.

Deadlock avoidance, by contrast, does not eliminate any of the four conditions of deadlock. It relies, instead, on *a priori* knowledge about the future behavior of a program, in the form of maximum possible requests for each resource. This information must be supplied before the program makes any resource requests, and so cannot be dependent on the program's input. It would be difficult (or impossible) to automate the calculation of this information, and so it depends on accurate information being supplied by the user. Furthermore, deadlock avoidance adds overhead to resource allocation due to the safety checks that must be performed at the time of each request.

8(a) In general, the only way to determine if the boolean expression B is true is to evaluate it. This implies that `await(B)` must either be implemented with busy waiting or have the compiler generate code to reevaluate B everytime its value may change. The first option is inefficient and the second is impractical.

## Spring 1985—Software Systems (Solutions)

8(b) Await( $v$ ) may be translated as:

if not  $v$  then wait( $c_v$ );

where  $c_v$  is the condition variable associated with  $v$ . The code

if  $v$  then signal( $c_v$ );

must follow every place where  $v$  may change.

⋮

## MATHEMATICAL THEORY OF COMPUTATION

### 1. Language Theory (20 points total)

#### A.

	(1) Union	(2) Complementation	(3) Kleene Star	(4) Intersection with a Regular Language
(a) Regular	yes	yes	yes	yes
(b) Context-Free	yes	no	yes	yes
(c) Deterministic Polynomial Time Recognizable	yes	yes	yes	yes
(d) Recursively Enumerable	yes	no	yes	yes

#### B.

- (a-2) Regular languages are closed under Complementation: Just make the accepting states of the DETERMINISTIC finite state automaton non-accepting, and vice-versa.
- (b-3) Context-free languages are closed under intersection with a regular language: Take a pushdown automaton  $M$  that accepts the CFL  $L$ , and a finite automaton  $M'$  for the regular language  $L'$ . Construct a new pushdown automaton whose store moves are determined by the moves of  $M$ , and whose acceptance is determined by both that of  $M$  and of  $M'$ .
- (c-3) Deterministic polynomial time-recognizable languages are closed under Kleene star: Given a string  $x$ , to determine whether it is in  $L^*$  ( $L$  a language in P), determine by dynamic programming whether each of its  $|x|^2$  substrings are in  $L'$ .
- (d-2) Recursively enumerable languages are not closed under Complementation. Counterexample: The set of all theorems of First-Order Logic.

### 2. Program Verification

A: Loop Invariant: " $m \geq 0, pq^m = n^{m_0}$ " (where  $m_0$  is the input value of  $m$ ). It holds at the first execution. To show that it remains invariant from one execution of the loop to the next, consider two cases: If  $m$  is even, then  $m$  is halved and  $q$  is squared (O.K.); if  $m$  is odd, then  $m$  becomes  $(m-1)/2$ ,  $p$  becomes  $pq$  and  $q$  is squared (as a result  $pq^m$  stays the same). Also, integer division by two preserves nonnegativity.

B. The loop terminates because  $m$  "loses one bit" in each iteration, and so must reach zero.

C. If the loop terminates, then the invariant gives, with  $m = 0$ :  $n^{m_0} = pq^0$ ; thus  $p = n^{m_0}$ , as desired. Since the loop does terminate, the program is correct.

### 3. Logic

A.

$$\left( \forall x A(x) \Rightarrow \forall x B(x) \right) \Rightarrow \forall x \left( A(x) \Rightarrow B(x) \right) \quad (a)$$

This formula is satisfiable. For example, take the universe to be all people  $A(x)$  to mean that  $x$  has a brain, and  $B(x)$  that  $x$  has no wings. All subformulae are true, including the whole formula. However it is not valid: Take the same universe,  $A(x)$  to mean “ $x$  is female”,  $B(x)$  “ $x$  is male”. The lhs is true (since ITS lhs is false), but the rhs is false.

$$\forall x \left( A(x) \Rightarrow B(x) \right) \Rightarrow \left( \forall x A(x) \Rightarrow \forall x B(x) \right) \quad (b)$$

This is a valid formula, as can be shown by resolution. First negate it:

$$\neg \forall x \left( A(x) \Rightarrow B(x) \right) \Rightarrow \left( \forall x A(x) \Rightarrow \forall x B(x) \right)$$

and convert to clausal form:

$$\left( \neg A(x) \vee B(x) \right) \wedge A(y) \wedge \neg B(s_0)$$

\*Resolving the first two clauses gives  $B(x)$ , which clearly resolves with the last clause to give the empty clause.

B. Gödel's Completeness Theorem says that First-Order Logic (or Predicate Calculus) has a finite axiomatization, that is, all theorems in FOL can be proven starting from these axioms. Consequently (in fact, equivalently), FOL is recursively enumerable.

The Incompleteness Theorem says that any logical system powerful enough to encompass Number Theory (Arithmetic) is bound to be incomplete, that is, there are going to be true facts in that system that, are not provable within the system.

⋮