Computer Science Department
Stanford University
Comprehensive Examination in Analysis of Algorithms
Autumn 1994

October 17, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in BLUE BOOKs. There are 5 problems in the exam. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

3. This exam is CLOSED BOOK.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers.

Comprehensive Exam: Analysis of Algorithms (60 points) **Autumn 1994**

1. **(5pts)** Determine the order of growth of the function $T(n)$, defined by the following recurrence:
   - $T(1) = 1$.
   - For $n \geq 2$, $T(n) = 3T(n/6) + n$.

2. **(5pts)** How many different irreflexive symmetric binary relations can be defined on a set of n elements ?

3. **(15pts)** Consider a weighted undirected graph with n nodes and m edges. Assume that you are given a minimum-weight spanning tree of this graph. How fast can you recompute the minimum spanning tree after the weight of one of the edges decreases by some given amount? Give a concise proof that your algorithm indeed produces a minimum-weight spanning tree.

4. **(15pts)** You are presented with a sequence of functions $f_1, f_2, \ldots$, where each function is defined on the interval from 1 to n. Function f_i is equal to 1 on the interval from 1 to some s_i and equal to 0 on the interval from $s_i + 1$ to n. The sequence is presented *one function at a time*. At any moment you might be given $x \in \{1 \ldots n\}$ and asked to evaluate at x the sum of all of the functions that were already revealed to you.
   Describe a data structure that stores the known functions and supports two operations: "evaluate $\sum_{i=1}^{k} f_i(x)$", where k is the number of known functions in the sequence, and "add a new function." One (inefficient) possibility is to maintain an array of size n. Initially, every element in the array is set to 0. Each time a new function (say, f_k) is revealed, we add 1 to all elements numbered 1 to s_k. In this case, adding a new function takes $O(n)$ operations and evaluating the sum takes $O(1)$ operations. Your goal is to design a data structure that is efficient both for computing the sum and for adding a new function to the set of known functions.

   Describe a data structure where each one of the above operations takes $O(\log n)$ time. Assume that you have $O(n)$ storage available.

5. **(20pts)** You have a computer that consistently trashes your disk each time you run more than k processes at the same time. You would like to find the value of k, while trashing your disk at most 2 times. Assume that you know that $k \leq n$.

   (a) **(5pts)** Describe a strategy that minimizes the number of "experiments" needed to find k, where an experiment consists of running some number of processes, running "fsck" afterwards to check whether the disk was trashed, and restoring the file system if it was.

   (b) **(5pts)** Generalize your strategy to any (constant) number of allowed crashes. Given a fixed number of crashes you are willing to sustain, what is the asymptotically maximum number of experiments used by your strategy ?

   (c) **(10pts)** Prove that, in the worst case, your strategy achieves asymptotically minimum possible number of experiments in the case where you are allowed only 2 crashes.

SOLUTIONS:
Algorithms

Comprehensive Exam: Analysis of Algorithms (60 points) **Autumn 1994**

Problem 1

First we compute $\alpha = \log_6 3$ and compare n^α it with n. In our case, $n = \Omega(n^{\alpha+\epsilon})$ for sufficiently small constant ϵ. Moreover, the "regularity condition" is satisfied, *i.e.* $3n/6 < \beta n$ for a constant β (for example, $\beta = 3/4$ works). Hence, we can conclude that $T(n) = O(n)$.

Problem 2

Binary relations on n elements have 1-to-1 correspondence with 0/1 $n \times n$ matrices. Entry $A_{ij} = 1$ if and only if (i, j) pair belongs to the relation. Symmetric means that $A_{ij} = A_{ji}$, and irreflexive means that the diagonal of A is 0. Hence, in order to determine a relation we need to set $n(n-1)/2$ entries in the matrix. There are $2^{n(n-1)/2}$ possibilities.

Problem 3

If the edge with reduced weight is in the tree, then there is nothing to do. If it is outside the tree, then adding it to the tree creates a cycle. To construct a new tree, we delete the largest weight edge on this cycle. This can be implemented in $O(n)$ time.

It remains to prove that the resulting tree is MST. We need a bit of notation. Let w denote the original edge weights and w' denote the new edge weights. Denote the deleted edge by e_1 and the new edge by e_2. Let T denote the original MST and let T' denote the computed MST, i.e. $T' = T - e_1 + e_2$.

Assume there exists an MST T'' where $w'(T'') < w'(T')$. Observe that T'' has to use e_2, otherwise T is not MST. Consider the cycle that includes e_1 and e_2 in $T + e_2$. One of the edges, say e_3, on this cycle is not in T''. Add it to T'' and remove e_2. The weight of the resulting tree is at most

$$\begin{aligned} w'(T'') + w'(e_3) - w'(e_2) &\leq w'(T'') + w(e_1) - w'(e_2) \\ &< w'(T') + w(e_1) - w'(e_2) \\ &= w(T) \end{aligned}$$

Thus, we got a tree that does not use e_2, but which is cheaper than T, which is a contradiction.

Problem 4

Our data structure will be a complete binary tree with $2^{\lceil \log_2 n \rceil}$ leaves. Each node of the tree holds a single number in addition to the pointers to the parent and its children, and hence the total storage requirements are $O(n)$.

Given a point $x \in [1 \ldots n]$, to compute the value of the sum of the already revealed functions at x, we start at the leaf that corresponds to x and traverse the tree up, adding all the numbers that we encounter up to the root. Clearly, this takes $O(\text{depth})$ of the tree, *i.e.* $O(\log n)$.

Assume we are given a new function that is 1 from 1 to s and 0 from $s+1$ to n. Represent s as a binary number with $\lceil \log_2 n \rceil$ digits. Traverse the tree starting at the root, going left if the digit is 0 and going right if a digit is 1. Each time we go right, we add 1 to the number stored at the left child of the current node. This takes $O(\text{depth})$ of the tree, *i.e.* $O(\log n)$.

Problem 5

Observe that the question corresponds to probing an array of n elements where all elements up to k are 0 and the rest are 1. A crash corresponds to probe returning "1".

5a Probe at $i\sqrt{n}$ for $i = 1, 2, \ldots$, until the first time a probe returns 1. This identifies a range $i^*\sqrt{n} \leq k \leq (i^*+1)\sqrt{n}$ for some i^*. Explore this range by linearly probing $i^*\sqrt{n}, i^*\sqrt{n}+1, \ldots$. Both stages of this strategy use at most $O(\sqrt{n})$ probes.

5b Assume we are allowed q crashes. Probe the array at $O(n^{1/q})$ equally spaced places, starting from 1, until the first time a probe returns 1. This identifies a range of size $O(n^{1-1/q})$ where k might be. Again, probe at most $O(n^{1/q})$ equally spaced places, starting at the smallest index of the identified range, until the first 1 is found. This identifies a range of size $O(n^{1-2/q})$. Continue until the size of the range is $O(n^{1/q})$. Search this range linearly.

At each one of the q stages of this strategy we used $O(n^{1/q})$ probes, and thus we used $O(qn^{1/q})$ probes total.

5c We will use the "adversary" argument. In other words, the adversary will let the algorithm probe, and decide on the value of k as it goes along. The only constraint is that this value should be consistent with all the answers to the probes.

Initially, k is in the range of 1 to n. The algorithm has to issue probes that force the adversary to pick a specific k. The algorithm can not terminate as long as the adversary has any freedom in picking k, since in this case the value of k returned by the algorithm will not be correct.

We say that a probe is of "type 1" if it is closer than $\sqrt{n}$ to the largest index probed up until now. Probes below the maximum-probed index do not bring any new information and hence can be discarded. We classify the rest of the probes as "type 2".

As long as the algorithm uses "type 1" probes, the adversary returns 0. Note that each one of these probes can decrease the range of possible values for k by at most $\sqrt{n}$, and hence the algorithm needs at least at least $O(\sqrt{n})$ type 1 probes to decrease the range of possible values for k to below $\sqrt{n}$.

Thus, if the number of probes is below $\Omega(\sqrt{n})$, the algorithm makes at least one type 2 probe. As a response to the first such probe, the adversary returns 1. At this moment, the only restriction on k that the adversary has is that k is larger than the largest probed index that returned 0 and smaller than the index probed by the type 2 probe. From now on the only valid algorithm is to linearly probe this range. This is true because if the algorithm misses an index, say i, then the adversary sets k to i and returns 1 at the next probe. Hence, in this case, the total number of probes is $O(\sqrt{n})$.

Computer Science Department
Stanford University
Comprehensive Examination in Artificial Intelligence
Autumn 1994

October 18, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 4 pages.

3. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why *you* think something is true when *we* think it is actually false. But no partial credit can be given if you write nothing.

1. LOGIC and AUTOMATED REASONING [20 Points]

(a) i. Transform the following sentence into clausal form
$S : \forall x.\forall y.(P(x,y) \wedge Q(y,x)) \Rightarrow \forall x.\exists y.(R(x,y) \vee \forall z.S(y,z))$ [3 Points]

ii. In general, does the procedure that you use in part i above result in clauses that preserve the validity of a sentence?
Does it preserve the satisfiability of a sentence? [2 Points]

(b) Find a most general unifier for each of the following sets, if one exists; otherwise, explain why such a unifier does not exist. [6 Points]
(Assume that u, w, x, y, z are variables, A, B are constants, and f, g are function symbols.)

i. $\{P(x,y,z), P(w,u,y), P(u,A,u)\}$
ii. $\{P(z, f(x,y)), P(f(A,B), x)\}$
iii. $\{P(f(x,y,z)), P(f(g(y), f(z,A,B), g(A)))\}$

(c) Everyone is a skier or a mountain climber. All mountain climbers hate rain, and all skiers like snow. Mike likes everything that Tom does not like, and Tom likes everything that Mike does not like. Tom likes rain and snow. Therefore there is a mountain climber that is not a skier.

The clausal form for this set of sentences is the following:
(Assume that x is a variable, and Tom, $Mike$, $Rain$, and $Snow$ are constants.)

C_1. $\{Skier(x), Climber(x)\}$
C_2. $\{\neg Likes(x, Rain), \neg Climber(x)\}$
C_3. $\{\neg Skier(x), Likes(x, Snow)\}$
C_4. $\{\neg Likes(Mike, x), \neg Likes(Tom, x)\}$
C_5. $\{Likes(Mike, x), Likes(Tom, x)\}$
C_6. $\{Likes(Tom, Rain)\}$
C_7. $\{Likes(Tom, Snow)\}$
C_8. $\{\neg Climber(x), Skier(x)\}$

Use resolution with set of support to derive the empty clause, where $\{C_8\}$ is the set of support. [9 Points]

2. LEARNING

15 Points

(a) Consider the problem of concept formation in books in a book store. The specific relations for the books are

- type (which can be an element of { fiction, non-fiction }),
- binding (which can be an element of { Paperback, Hardcover }), and
- subject (which can be an element of { AI, Geology, Air-Travel }).

Our language restricts the space of possible concepts to be conjunctions of positive literals.

The following instances are in the target set (i.e. are positive instances)

	SPHERE	GÖDEL-ESCHER-BACH	LFAI
TYPE:	FICTION	NON-FICTION	NON-FICTION
BINDING:	PAPERBACK	PAPERBACK	PAPERBACK
SUBJECT:	AI	AI	AI

The following instances are not in the target set (i.e. are negative instances)

	Terminal Man	AIRPORT
TYPE:	FICTION	NON-FICTION
BINDING:	HARDCOVER	PAPERBACK
SUBJECT:	AI	AIR-TRAVEL

i. What is the version graph for the target set? [5 Points]

ii. What are the most general and the most specific concepts consistent with this set? [4 Points]

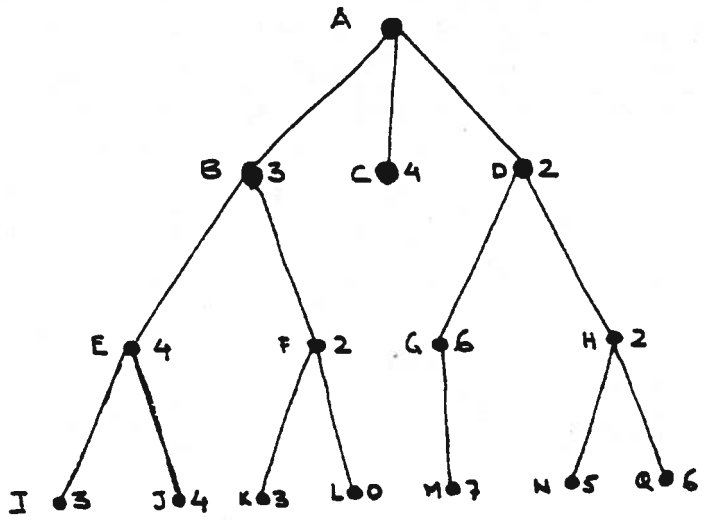
(b) Find the smallest decision tree to represent the concept

$(P_1(x) \wedge P_2(x)) \vee (P_3(x) \wedge P_4(x) \wedge P_5(x))$ [6 Points]

3. SEARCH

15 Points

(a) Consider the following search space:



List the order in which nodes are visited in the tree above for each of the following search strategies. The values next to the node labels represent the (heuristic) cost to get to the goal node, arc traversal is a unit cost operation, and L is the goal node.

i. A* Search. [6 Points]

ii. IDA* Search (assuming an initial depth cutoff of 1). [5 Points]

(b) Briefly explain the differences between hill climbing and best-first search. [4 Points]

4. PROBABILISTIC REASONING [10 Points]

Consider the three sentences, H: John's car battery is low, E_1: John has difficulty starting his car, E_2: John's car has dimmed headlights. Suppose $p(E_1|H) = 0.8$. $p(E_1|\neg H) = 0.2$. $p(E_2|H) = 0.75$. $p(E_2|\neg H) = 0.6$, and $p(H) = 0.1$.

Assume the Odds of an event A given B to be

$O(A|B) = p(A|B)/p(\neg A|B)$

(a) What are the odds that John's car battery is low given that he has difficulty starting his car? [5 Points]

(b) What are the odds that John's car battery is low given that he has difficulty starting his car *and* has dimmed headlights? (for this part assume that given H (or $\neg H$). E_1 and E_2 are independent.) [5 Points]

SOLUTIONS:
Artificial Intelligence

1. (a) i. The clausal form for this sentence is the following:
$\{\neg P(S_1, S_2), \neg Q(S_2, S_1), R(x, f(x)), S(f(x), z)\}$
where S_1, S_2 are skolem constants and f is a skolem function.

ii. In general, does the procedure that you use in part i above result in clauses that preserve the validity of a sentence?
No. The clausal form conversion procedure described in Manna and Waldinger's *The Deductive foundations of Computer Programming* is a validity preserving transformation; it is based on the notion of the *existential closure* of a sentence.
Does it preserve the satisfiability of a sentence? [2 Points]
Yes. To prove the unsatisfiability of a set of sentences, we prove the unsatisfiability of the sentences in clausal form.
Note that if you chose the procedure described in Manna and Waldinger's book, the answer to the first part of the question would be yes, and that to the second part would be no. Our answers here assume that you use the procedure described in LFAI.

(b) i. Since u has to be made equal to A, all the variables have to be made equal to A. An mgu then is: $\{u \leftarrow A, w \leftarrow A, x \leftarrow A, y \leftarrow A, z \leftarrow A\}$.

ii. This set is not unifiable, since x would have to be made equal to $F(x, y)$. and the *occurs check* fails.

iii. An mgu for this set is:
$\{z \leftarrow G(A), y \leftarrow F(G(A), A, B), x \leftarrow G(F(G(A), A, B))\}$

(c) C_1. $\{Skier(x), Climber(x)\}$
C_2. $\{\neg Likes(x, Rain), \neg Climber(x)\}$
C_3. $\{\neg Skier(x), Likes(x, Snow)\}$
C_4. $\{\neg Likes(Mike, x), \neg Likes(Tom, x)\}$
C_5. $\{Likes(Mike, x), Likes(Tom, x)\}$
C_6. $\{Likes(Tom, Rain)\}$
C_7. $\{Likes(Tom, Snow)\}$
C_8. $\{\neg Climber(x), Skier(x)\}$

A resolution with set of support refutation for this set of clauses, where $\{C_8\}$ is the set of support, is the following.

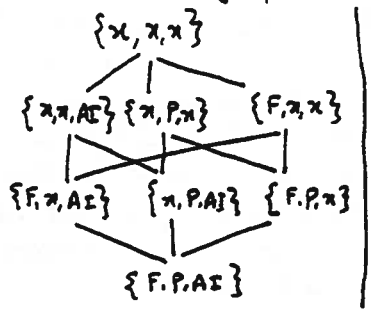
C_9: $\{Skier(x)\}$	C_8, C_1, Factoring.
C_{10}: $\{Likes(x, Snow)\}$	C_9. C_3.
C_{11}: $\{\neg Likes(Tom, Snow)\}$	C_{10}. C_4.
C_{12}: $\{\}$	C_{11}. C_7.

2. LEARNING

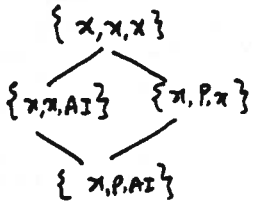
15 Points

a] [i] We start with the version graph with Sphere as the sole +ve instance.

This graph is:

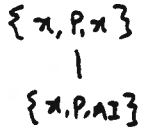


. Next, with GEB as another +ve instance, we prune the relations restricted to F. The graph now is:



With LFAI there's no change to this graph.

Next, with Terminal Man as a −ve instance, we remove those nodes inconsistent with it so we are left with.

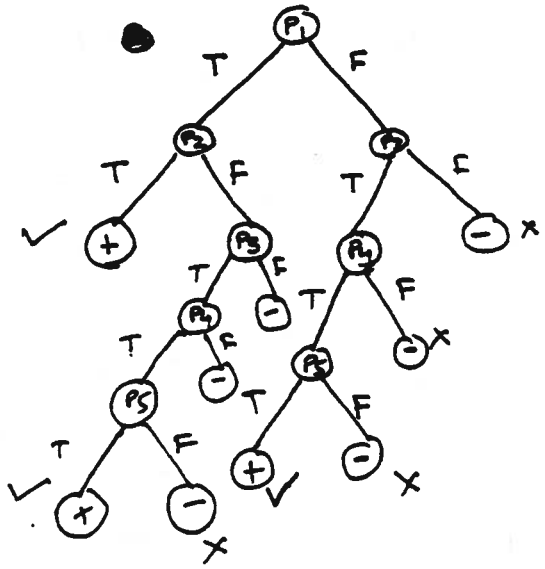


. Finally, with Airport, we remove the node inconsistent with subject AirTra

& our final version graph is → {x, P, AI}

[ii] In this case the candidate elimination algorithm converges to a single concept. Both the most general and the most specific concepts consistent with this set are {x, P, AI}, i.e. all AI papers

b] One such decision tree is:



Any tree rooted on either P_3, P_4 or P_5 must strictly be larger than this tree, because the subtrees to test for $\{P_1(x) \wedge$ must be a child of each of P_3, P_4 & P_5 nodes.

3. SEARCH

15 Points

(a) i. A, D, B, H, F, L.

ii.
- For cutoff = 1, 2, only expand A.
- For cutoff = 3, expand A and D.
- For cutoff = 4, expand A, D, B, H, F, L.

(b) In hill climbing search we examine only the nodes that are directly reachable, i.e. one step away, from the last examined node. This gives hill climbing a depth first flavor. In best-first search, the unexplored nodes that were encountered earlier in the search are considered as well. This gives best-first search a breadth-first flavor.

4. PROBABILISTIC REASONING

[10 Points]

(a)

$$O(H|E_1) = \frac{p(H|E_1)}{P(\neg H|E_1)} = \frac{p(E_1|H)p(H)}{p(E_1|\neg H)p(\neg H)} = \frac{(.8)(.1)}{(.2)(.9)} = \frac{4}{9}$$

(b)

$$O(H|E_1 \wedge E_2) = \frac{p(H|E_1 \wedge E_2)}{P(\neg H|E_1 \wedge E_2)} = \frac{p(E_1 \wedge E_2|H)p(H)}{p(E_1 \wedge E_2|\neg H)p(\neg H)} = \frac{p(E_1|H)p(E_2|H)p(H)}{p(E_1|\neg H)p(E_2|\neg H)p(\neg H)}$$

$$= \frac{p(E_2|H)}{p(E_2|\neg H)} O(H|E_1) = \frac{.75}{.6} \times \frac{4}{9} = \frac{5}{9}$$

Computer Science Department
Stanford University
Comprehensive Examination in Automata and Formal Languages
Autumn 1994

October 17, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in BLUE BOOKS. There are three problems in the exam. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. The exam takes 30 minutes.

3. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers.

Comprehensive Exam:

Autumn 1994

Automata and Formal Languages (30 points)

Instructions: You are expected to *sketch* the main ideas in your solutions, but be very brief and avoid unnecessary detail. You are permitted to invoke any result proved in the Hopcroft-Ullman book provided you include the appropriate citation.

1. (8 points) Consider the following context-free grammar G.

$$\begin{aligned} S &\rightarrow bABaa \mid Sa \mid a \\ A &\rightarrow aB \\ B &\rightarrow baB \mid \epsilon \end{aligned}$$

Let L_A and L_B be the languages consisting of the terminal strings that can be derived from the variables A and B, respectively.

(a) [4 points] Show that L_A and L_B are regular by providing regular expressions for these two languages.

(b) [4 points] Show that $L(G)$ is regular by providing a regular expression for it.

2. (10 points) A *monotone* 2-SAT formula is a 2-CNF boolean formula $F(x_1, \ldots, x_n)$ which does not contain negated variables. For example:

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3).$$

It is clear that there always exists a truth assignment for the variables $x_1, \ldots, x_n$ satisfying the formula F – simply set each variable to TRUE.

Consider the following problem called MONOTONE 2-SAT: given a monotone 2-SAT formula F and a positive integer k, determine whether there exists a truth assignment satisfying F such that the number of variables set to TRUE is *at most* k.

Show that the MONOTONE 2-SAT problem is NP-hard. (**Hint:** Think about the vertex cover problem.)

3. (12 points) Consider the following decision problem:

> Given a deterministic finite state automaton (DFA) M over the alphabet $\Sigma = \{0, 1\}$, does $L(M)$ contain at least 2 strings?

Is this problem decidable? Justify your answer. (**Hint:** Think about the decision problems of deciding emptiness and finiteness of regular languages.)

SOLUTIONS:
Automata

Comprehensive Exam: **Autumn 1994**

Automata and Formal Languages (Solutions)

Instructions: You are expected to *sketch* the main ideas in your solutions, but be very brief and avoid unnecessary detail. You are permitted to invoke any result proved in the Hopcroft-Ullman book provided you include the appropriate citation.

1. (8 points) Consider the following context-free grammar G.

$$\begin{aligned} S &\rightarrow bABaa \mid Sa \mid a \\ A &\rightarrow aB \\ B &\rightarrow baB \mid \epsilon \end{aligned}$$

Let L_A and L_B be the languages consisting of the terminal strings that can be derived from the variables A and B, respectively.

(a) [4 points] Show that L_A and L_B are regular by providing regular expressions for these two languages.

(b) [4 points] Show that $L(G)$ is regular by providing a regular expression for it.

Solution: By inspection, L_B has the regular expression $(ba)^*$, and L_A has the regular expression $a(ba)^*$.

Consider now the productions from S. We observe that the productions from A and B do not generate any sentential form containing S. Also the production $S \rightarrow bABaa$ is not recursive in S which implies that it can be used at most once in the any derivation from S. From these facts, we can conclude that the strings derived from S are of the form $(bABaa+a)a^*$. Thus, $L(G)$ has the regular expression $(ba(ba)^*(ba)^*+a)a^*$, which simplifies to $(ba(ba)^*+a)a^*$.

2. (10 points) A *monotone* 2-SAT formula is a 2-CNF boolean formula $F(x_1, \ldots, x_n)$ which does not contain negated variables. For example:

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3).$$

It is clear that there always exists a truth assignment for the variables $x_1, \ldots, x_n$ satisfying the formula F – simply set each variable to TRUE.

Consider the following problem called MONOTONE 2-SAT: given a monotone 2-SAT formula F and a positive integer k, determine whether there exists a truth assignment satisfying F such that the number of variables set to TRUE is *at most* k.

Show that the MONOTONE 2-SAT problem is NP-hard. (**Hint:** Think about the vertex cover problem.)

Solution: In a graph $G(V, E)$ with $V = \{1, \ldots, n\}$, a vertex cover is a set of vertices $C \subseteq V$ such that for each edge $(i, j) \in E$, $\{i, j\} \cap C \neq \emptyset$. The VC problem is the following: given a graph $G(V, E)$ and a positive integer k, does G contain a vertex cover of size at most k. We know that VC is NP-hard, and establish the NP-hardness of MONOTONE 2-SAT by reduction from VC.

The reduction starts with a VC instance $< G, k >$ and creates an instance $< F, k >$ of MONOTONE 2-SAT, where the monotone 2-CNF formula F is defined as follows: for each vertex $i \in V$, create a boolean variable x_i; for each edge $(i, j) \in E$, create a clause $x_i \vee x_j$. The reduction runs in linear time, but it remains to verify its correctness.

Suppose $G(V, E)$ has a vertex cover C of size at most k. Consider the truth assignment for the variables in F in which $x_i = \text{TRUE}$ if and only if $i \in C$; clearly, the number of TRUE variables is at most k. We claim that this is a satisfying truth assignment for F. To establish the claim, we show that each clause $x_i \vee x_j$ in F is satisfied. Since (i, j) must be an edge, and C must contain at least one of i and j, if follows that at least one of x_i and x_j is assigned TRUE and the clause is satisfied.

Suppose now that there is a satisfying truth assignment for F with no more than k variables set to TRUE. Consider the set of vertices $C = \{i \mid x_i = \text{TRUE}\}$; clearly, $|C| \leq k$. We claim that C is a vertex cover for G. To see this, focus on any one edge $(i, j) \in E$. Since F must have a clause $x_i \vee x_j$, and that clause is satisfied, at least one of x_i and x_j is assigned TRUE and so at least one end-point of the edge (i, j) belongs to C.

3. (12 points) Consider the following decision problem:

> Given a deterministic finite state automaton (DFA) M over the alphabet $\Sigma = \{0, 1\}$, does $L(M)$ contain at least 2 strings?

Is this problem decidable? Justify your answer. (**Hint:** Think about the decision problems of deciding emptiness and finiteness of regular languages.)

Solution: The problem is indeed decidable. The proof is similar to that for the decidability of emptiness or infiniteness of regular languages, as described in Theorem 3.7 (Hopcroft-Ullman, p. 63). We also make use of the fact that given any string x, it is possible to decide membership of x in $L(M)$.

Let n be the number of states in the DFA M. The decision procedure enumerates all strings $x \in \Sigma^*$ of length less than n, and checks for their membership in $L(M)$; let m_1 be the number of such strings in $L(M)$. Then, the decision procedure enumerates all strings $x \in \Sigma^*$ such that $n \leq |x| < 2n$ and checks their membership in $L(M)$; let m_2 be the number of such strings in $L(M)$.

By Theorem 3.7, if $m_2 > 0$ then $L(M)$ is infinite, and the procedure outputs YES. Assume now that $m_2 = 0$. By Theorem 3.7, or by a direct application of the Pumping Lemma, we have that if $L(M)$ contains a string of length at least $2n$ then it must contain a string of length between n and $2n - 1$, i.e., $m_2 > 0$. Since $m_2 = 0$, all strings in $L(M)$ are of length less than n implying that $|L(M)| = m_1$. Thus, the decision procedure now has to merely verify that $m_1 \geq 2$.

Computer Science Department
Stanford University
Comprehensive Examination in Compilers
Autumn 1994

October 20, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 2 pages.

3. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

CS Comprehensive Exam: Compilers (30 points)

1. (7 points)

   Comments in the C programming language are introduced by the characters /* and terminated by the characters */. They do not nest and they do not occur within string or character literals. For each of the following regular expressions, indicate whether or not the expression describes precisely the set of all valid C comments. For those that do not, provide a counterexample. Assume for convenience that the character set is {*,/,a}.

   (a) /*(/|a|*)**/

   (b) /*(a|/)**(*|a(a|/)**)*/

   (c) /*(a|/)**(*|a(a|/)**)**/

2. (8 points)

   The following grammar generates all regular expressions over the alphabet $\{a, b\}$:

   $$R \rightarrow R + R \mid RR \mid R^* \mid (R) \mid a \mid b$$

   where '+' denotes 'or'. Unary * has highest precedence, followed by concatenation; disjunction has the lowest precedence. All operators are left associative.

   (a) Show that the grammar is ambiguous.

   (b) Construct an equivalent unambiguous grammar that enforces the correct operator precedence and associativity.

   (c) Eliminate any left-recursive productions and productions with common prefixes from your grammar in (b).

3. (4 points) What is meant by *structural equivalence* and *name equivalence* of types? Is one more restrictive than the other? Is one easier for a compiler to check than the other?

4. (11 points)

   Suppose you are asked to extend the C programming language to allow overloading of user-defined functions, based on the number of formal parameters and their types.

   Answer the following questions as briefly, *yet clearly*, as you can. We are more interested in abstract operations (*e.g.*, "store x in a table, along with its size") than in implementation details (*e.g.*, "for performance, implement the table with a doubly-linked list of AVL trees").

   (a) How can a compiler keep track of the parameters and their types? Describe what should be recorded when a function declaration is processed.

(b) How can a compiler determine which function to invoke in a procedure call? Describe what and how information can be retrieved when resolving a procedure call.

(c) Describe type errors that can occur as a result of this language extension, and how a compiler can detect them.

SOLUTIONS:
Compilers

CS Comprehensive Exam: Compilers (30 points)

1. (7 points)

Comments in the C programming language are introduced by the characters /* and terminated by the characters */. They do not nest and they do not occur within string or character literals. Indicate whether or not each of the following regular expressions describes precisely the set of all valid C comments. For those that do not, provide a counterexample. Assume for convenience that the character set is {*,/,a}.

(a) /*(/|a|*)"*/

Ans: no, this expression matches /**/*/.

(b) /*(a|/)"*(*|a(a|/)"*)"/

Ans: yes, perhaps most easily seen by constructing a DFA.

(c) /*(a|/)"*(*|a(a|/)"*)"*/

Ans: no, this expression fails to match /**/.

2. (8 points)

The following grammar generates all regular expressions over the alphabet $\{a, b\}$:

$$R \rightarrow R + R \mid RR \mid R^* \mid (R) \mid a \mid b$$

where '+' denotes 'or'. Unary * has highest precedence, followed by concatenation; disjunction has the lowest precedence. All operators are left associative.

(a) Show that the grammar is ambiguous.

Two leftmost derivations of *aba* are:

$$R \Longrightarrow RR \Longrightarrow RRR \Longrightarrow aRR \Longrightarrow abR \Longrightarrow aba$$
$$R \Longrightarrow RR \Longrightarrow aR \Longrightarrow aRR \Longrightarrow abR \Longrightarrow aba$$

(b) Construct an equivalent unambiguous grammar that enforces the correct operator precedence and associativity.

$$\begin{aligned} R &\rightarrow R+T \\ R &\rightarrow T \\ T &\rightarrow TF \\ T &\rightarrow F \\ F &\rightarrow F^* \\ F &\rightarrow (E) \\ F &\rightarrow a \\ F &\rightarrow b \end{aligned}$$

(c) Eliminate any left-recursive productions and productions with common prefixes from your grammar in (b).

$$
\begin{array}{lcl}
R & \rightarrow & TA \\
A & \rightarrow & +TA \\
A & \rightarrow & \epsilon \\
T & \rightarrow & FB \\
B & \rightarrow & FB \\
B & \rightarrow & \epsilon \\
F & \rightarrow & CD \\
C & \rightarrow & (E) \\
C & \rightarrow & a \\
C & \rightarrow & b \\
D & \rightarrow & *D \\
D & \rightarrow & \epsilon
\end{array}
$$

3. (4 points) What is meant by *structural equivalence* and *name equivalence* of types? Is one more restrictive than the other? Is one easier for a compiler to check than the other?

Two type expressions are structurally equivalent if they are the same basic type (*e.g.*, integer or character), or if they are formed by applying the *same* constructor to structurally equivalent types (*e.g.*, *pointer*(T_1), *pointer*(T_2), where T_1 and T_2 are structurally equivalent types). When type expressions can be named, two type expressions are name equivalent precisely when they are identical. Name equivalence is more restrictive, and implies structural equivalence. Two types may be structurally equivalent but not name equivalent.

Checking expressions for name equivalence is particularly simple. Structural equivalence is defined recursively, so a compiler may check the types recursively. Alternatively, a compound type may be encoded explicitly as a type attribute to facilitate equivalence checking. In general, checking structural equivalence requires more work.

4. (11 points)

Suppose you are asked to extend the C programming language to allow overloading of user-defined functions, based on the number of formal parameters and their types.

Answer the following questions as briefly, *yet clearly*, as you can. We are more interested in abstract operations (*e.g.*, "store x in a table, along with its size") than in implementation details (*e.g.*, "for performance, implement the table with a doubly-linked list of AVL trees").

(a) How can a compiler keep track of the parameters and their types? Describe what should be recorded when a function declaration is processed.

We'll assume that types are represented in the compiler as pointers to structures, called `decls`, where each `decl` represents a type. We'll also assume that names are associated

with types, variables, *etc.*, using a scoped symbol table. A symbol table entry for a user-defined function will bind a function name to a *list* of its function `decls`.

When the compiler begins processing a function declaration, it creates a `decl` to represent the function type that contains the function name, return type, the names and types of its formal parameters, and other relevant information. The compiler creates a new scope and declares the formals in it, then processes the function body in yet another new enclosed scope. After processing the function body, the 'formals' scope can be saved in the `decl`, say as a list of (name, type) pairs. The function `decl` is then added to the `decl` list in the symbol table entry for the function's name.

(b) How can a compiler determine which function to invoke in a procedure call? Describe what and how information can be retrieved when resolving a procedure call.

The compiler must determine the number and types of the actuals, and compare them with the formals lists stored in the symbol table in the list of function `decls`. A simple-minded way to do this is to construct a list of actuals types and then look for a match with one of the formals lists stored in the function's type `decls` in the symbol table.

If a match is found, the compiler should check that the matched function declaration has a valid return type for the call. Otherwise the compiler should issue a type error message.

(c) Describe type errors that can occur as a result of this language extension, and how a compiler can detect them.

A couple of type errors that can arise are unresolvable function declarations (*e.g.*, C uses structural equivalence for all types except structures, unions, and enumerations, so structurally equivalent formals are not differentiable), and a function call that does not match any function declaration, either because the number of arguments does not match, or the types do not match.

Before adding a function type `decl` to the `decl` list in a symbol table entry for the function name, the compiler should verify that the declaration is resolvable by the formals list. This can be accomplished by comparing the saved formals list in the function `decl` with the formals lists of all previously declared functions with the same name. If two formals lists cannot be resolved, the compiler should issue a type error message.

Computer Science Department
Stanford University
Comprehensive Examination in Computer Architecture
Autumn 1994

October 19, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. This exam is CLOSED BOOK. You may not use notes, articles, or books.

3. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

SOLUTIONS:
Comp. Architecture

Oct 19, 1994

Question 1.0 Short Answer (10 points)

closed book
partial credit

1.a (3 points)
If you were comparing the performance of two computers, how would you summarize the performance each computer on a set of floating point benchmarks if the performance is reported in MFLOPS? Justify your answer.

You should use harmonic mean because MFLOPS is a rate. A harmonic mean of rates will show the same relative performance of the two computers as total execution time.

1.b (3 points)
What advantage does a branch target buffer (BTB) have over a branch history table (BHT)? What disadvantage does it have?

Advantage: The BTB has a zero branch delay when the prediction is correct. The BHT has at least a one cycle branch delay.
Disadvantage: The BTB requires far more silicon area for the same number of branch instruction slots because it must store the target address along with the prediction bits. Furthermore, the area required by a BHT could be drastically decreased if the tag area were eliminated. This is not possible for a BTB since two branches might map to the same slot, yet they may not share the target instruction address.

1.c (4 points)
Some designers have advocated using a small fully associative on-chip "victim cache" to hold the lines that are replaced from a direct-mapped on-chip primary cache. The victim cache is checked at the same time as the primary cache. If there is a miss in the primary cache and the line is found in the victim cache, the primary cache line and victim cache line are swapped. If the line is not found in the victim cache, it must be fetched form the next level of the memory hierarchy. Using what you know about cache access times and the 3 C's (compulsory misses, capacity misses, conflict misses) model, explain why the victim cache is a good idea.

For a given cache size a direct mapped cache has the lowest access time but the highest number of conflict misses. In certain pathological situations were consecutive accesses are to the same set, conflict misses will result in very high miss rates. The victim cache makes it possible to satisfy these conflict misses without going off chip. A victim cache has the effect of adding associativity to a direct mapped cache with out increasing the access time.

Question 2.0 Pipelining (30 points)

A dual issue superscalar processor implementation of the DLX ISA can execute up to two instructions per cycle as long as there are no dependencies between the instructions. Here is the pipeline for a dual issue processor.

Pipeline

Stage	Function
IF	instruction fetch
ID	decode, register fetch, PC-target, dependency check
ADDR	memory address generate, branch condition
EX/MEM	ALU operation, memory access
WB	writeback

You will use the following information to evaluate this processor. The frequencies in these tables are presented as percentages of all instructions executed.

Branch statistics

Branch type	Frequency of occurrence	Frequency of correct prediction
Taken	10%	9%
Not taken	7%	5%

Some DLX instruction pair frequencies

Type	Instruction sequence	Frequency
1	`ALU op Rx, -, -` `ALU op -, -, Rx or -, Rx, -`	10%
2	`ALU op Rx, -, -` `Store Rx, -(Rb)`	5%
3	`ALU op Rx, -, -` `Load/Store -, -(Rx)`	5%
4	`ALU op Rx, -, -` `JumpRegister Rx`	1%
5	`ALU op Rx, -, -` `Branch Rx`	2%
6	`Load Rx, -(-)` `ALU op -, -, Rx or -, Rx, -`	15%
7	`Load Rx, -(Rb)` `Load/Store -, -(Rx)`	3%
8	`Load Rx, -(Rb)` `Branch Rx, -`	2%
9	`Load Rx, -, -` `JumpRegister Rx`	1%

2.a (5 points)
How many adders, register file ports, and data memory ports does this processor need to minimize structural hazards?

6 Adders

IF	1	sequential instruction address
ID	1	branch target address
ADDR	2	memory addresses
EX	2	ALUs

6 Register File Ports

4 read ports
2 write ports

2 Data memory Ports

2 loads or two stores in the same instruction pair

2.b (8 points)
What is the minimum number of register specifier comparators required to implement dependency checks and forwarding in this processor? Show your reasoning.

All instructions that compute results have at most one destination register. Instructions can have up to two source registers.

Dependency Check

Need to make sure that second instruction does not depend on the first. We must compare two source registers of the second instruction against the destination of the first instruction. This requires **2** comparators.

Forwarding

Source -> destination

EX/MEM -> EX/MEM

Need 2 comparators for each register destination of the source instruction. We have two pipes so we need 4 comparators per source instruction. We have two sources so we need a total of **8** comparators.

EX/MEM -> ADDR

Each destination instruction uses one source register so we need **4** comparators

EX/MEM -> ID

Jump register only uses one source register so we need **4** comparators

Total # of comparators = 2 + 8 + 4 + 4 = 18

2.c (1 point).
What is the ideal CPI of this processor with an ideal memory system?

If we can execute two instructions every cycle, the ideal CPI of this processor is 0.5

2.d (6 points)
Assume the processor uses squashing branches with static branch prediction which is encoded in the opcode of the branch. What is the CPI due to control hazards? Make all reasonable assumptions to minimize CPI. Assume that the branch is the second instruction in the pair.

Here are the cases to consider:

Branch Prediction	Branch outcome	penalty (cycles)	frequency
taken	taken	1	9%
taken	not taken	2	1%
not taken	taken	2	2%
not taken	not taken	0	5%

CPI due to control hazards = 0.09*1 + 0.01*2 + 0.02*2 = 0.15

2.e (4 points)
Using the instruction sequence table, what is the CPI due to instructions that cannot be issued in pairs because of register dependencies? Assume the worst case.

In the worst case none of the instruction pairs in the table can execute together. Thus all pairs will have a CPI of 1. This is 0.5 more than the ideal CPI, thus
Extra CPI = 0.5(0.10 + 0.05 + 0.05 + 0.01 + 0.02 + 0.15 + 0.03 + 0.02 + 0.01) = 0.22

2.f (5 points)
Using the instruction sequence table, what is the CPI due to instruction sequences that cause stalls on this processor?

Stalls are caused when the ADDR or ID stages use a result generated in the EX/MEM stage. Here are the cases
Types 3, 5, 7, and 8 cause a 1 stall cycle
Types 4 and 9 cause a 2 stall cycles

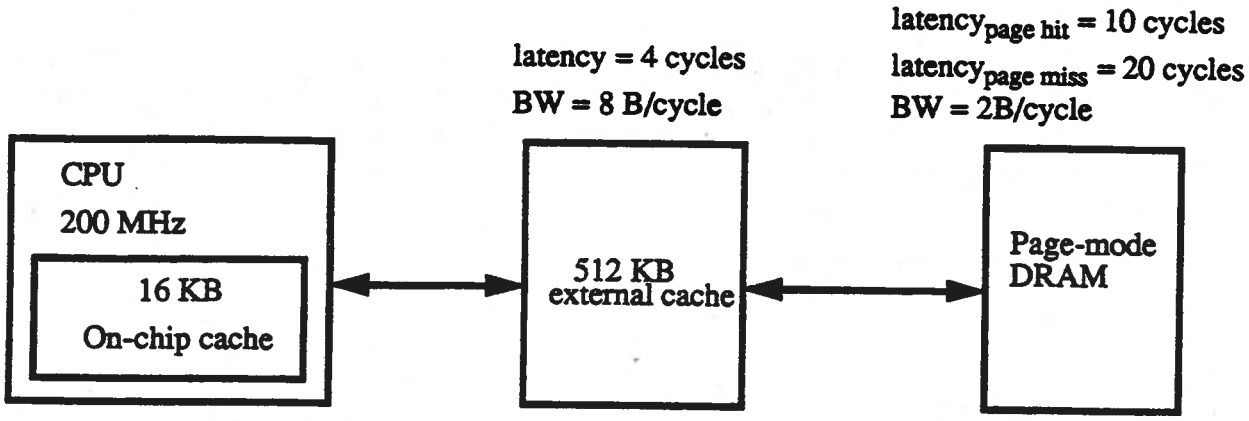
Stall CPI = 0.05*1 + 0.02*1 + 0.03*1 + 0.02*1 + 0.01*2 + 0.01*2 = 0.16

2.g (1 point)
What is the real CPI of this processor with an ideal memory system?

Add up CPI from parts c, d, e, and f.
real CPI = 0.5 + 0.15 + 0.22 + 0.16 = 1.03

Question 3.0 Memory System Design (20 points)

You are considering external cache option for a new high-performance workstation that will use a 200 MHz CPU with an on-chip cache (1 cycle access, direct-mapped). The system is shown below:



After running a set of real programs, you have come up with the following **global** miss ratios:

line size (bytes)	16 KB direct-mapped	512 KB direct-mapped	512 KB 2-way set associative
8	0.17	0.025	0.021
16	0.13	0.018	0.015
32	0.11	0.015	0.012
64	0.08	0.01	0.008

Other necessary information:

1. The probability of a page hit in the page-mode DRAM is 0.3.
2. Making the 512 KB external cache 2-way set associative adds one cycle to its latency but does not affect its bandwidth.

3.a (14 points)
Find the line size and set-associativity that minimizes AMAT measured in CPU cycles for the 512 KB cache. To guarantee multilevel inclusion, the line sizes of the 16 KB cache and 512KB cache must be equal.

Here is the AMAT expressio for the direct-mapped case

$$\text{AMAT} = 1 + \text{MR}_{16K} \times \left(4 + \frac{L}{8}\right) + \text{MR}_{512K} \times \left(10\,(0.3) + 20\,(0.7) + \frac{L}{2}\right)$$

line size	AMAT 256 KB 1-way	AMAT 256 KB 2-way
8	2.38	2.46
16	2.23	2.29
32	2.37	-
64	-	-

A direct mapped cache with a line size of 16 bytes has the lowest AMAT increasing the line size or making external cache set-associative will not improve performance. The - means no need to calculate.

3.b (7 points)
Assume that excluding the memory system the CPI of the processor is 1.4 and that there are 0.4 data references per instruction. How long will it take to execute one million instructions on this processor?

First calculate CPI

$$\text{CPI} = \text{CPI}_{\text{CPU}} + \text{CPI}_{\text{memory}}$$

$$= 1.4 + \left(\frac{\text{Instruction refs} + \text{Data refs}}{\text{\# of instructions}}\right) \times (\text{AMAT} - 1)$$

$$\text{CPI} = 1.4 + 1.4\,(2.23 - 1) = 3.12$$

Now calculate CPU time

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times 1/\text{clock freq.}$$

$$= 1\text{M} \times 3.12 \times 1/200\text{MHz}$$

$$= 0.0156 \text{ seconds}$$

SOLUTIONS:
Databases

1994 Comprehensive Exam in Databases
Sample Solution

1. (a) $\Pi_{\text{name,address}}((\sigma_{\text{dept="CS"}}(\text{student} \bowtie \text{enrolled})))$

   (b)
   ```
   SELECT name, address
   FROM   student, enrolled
   WHERE  student.ID = enrolled.ID
          AND dept = "CS"
   ```

   (c) Parts (a) and (b) will not return exactly the same result. SQL returns duplicate values in results while relational algebra does not. If a student is taking more than one CS course, then the student's name and address will appear multiple times in the answer to (b) but only once in the answer to (a). Note that if the query for (b) uses SELECT DISTINCT then the results will be the same.

   (d)
   ```
   SELECT sum(units)
   FROM   enrolled, course
   WHERE  enrolled.dept = course.dept
          AND enrolled.code = course.code
   ```

   (e) No. Relational algebra does not include aggregate operators such as sum.

2. B is the only key. Since B does not appear on the right of any functional dependency, it must be in any key. However, we can use $B \rightarrow E \rightarrow A$ to show $B \rightarrow A$, and then use $AB \rightarrow C$ to show $B \rightarrow C$. Thus B by itself is a key and hence is the only key.

3. (a) Lock Compatibility Matrix:

		Lock Held By Another Transaction:		
		read	write	update
Lock Requested:	read	Y	N	Y
	write	N	N	N
	update	Y	N	N

   (b) Example:

   ```
   Transaction T1 requests update lock for item A -> request granted
   Transaction T2 requests update lock for item A -> T2 must wait
   Transaction T1 requests write lock for A -> request granted
   Transaction T1 completes and releases all of its locks
   Transaction T2 gets update lock on A and continues processing
   Transaction T2 requests write lock for A -> request granted
   Transaction T2 completes and releases all of its locks
   ```

LOGIC COMP SOLUTIONS

Solution 1: (A) If P is a necessary condition for Q, then $\neg P \rightarrow \neg Q$. This is equivalent to $Q \rightarrow P$, which means that Q is a sufficient condition for P.

Solution 2: (A) By the definition of *"only if"*

Solution 3: (C) By a simple induction argument

Solution 4: (D) Straight-forward from the definitions.

Solution 5: (B)

Solution 6: (D) We can't conclude anything about the validity of $\mathcal{F}$ from the given assumption that $\mathcal{F}$ is satisfiable.

Solution 7: (B)

Solution 8: (D)

Solution 9: (D)

Solution 10: (D)

Solution 11: (B)

Solution 12: (A)

Solution 13: (A) follows from the compactness theorem.

Solution 14: (B) In general, validity need not be recursive, but since we have a sound and complete axiom system, we can enumerate all proofs. The proof of any valid sentence will be eventually appear, and so we have an effective enumeration for the valid sentences.

Solution 15: (C) The validity problem for propositional logic is decidable. That is, for any sentence we can determine if it is valid or not. For predicate logic, the validity problem is only semi-decidable.

Solution 16: (B) II is true because Gödel's Incompleteness Theorem holds for the theory of natural numbers with addition and multiplication. III is true because Gödel's Incompleteness Theorem does NOT hold for theory of natural numbers with only addition.

Solution 17: (C)

Computer Science Department
Stanford University
Comprehensive Examination in Numerical Analysis
Autumn 1994

October 21, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. This exam is CLOSED BOOK. You may not use notes, articles, or books.

3. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive Examination: Numerical Analysis (30 points). Fall 1994

(Problem I)

(15 points). **Linear Systems and Matrices**

(a) Define the condition number $\mathcal{K}$ of a matrix A. If

$$Ax = b, \quad Ay = b + r$$

state the upper and lower bounds relating the relative error $\frac{\|x-y\|}{\|x\|}$ to the relative peturbation $\frac{\|r\|}{\|b\|}$. *Briefly* discuss the implications of these inequalities for the numerical solution of linear systems.

(b) Let A be an $m \times m$ symmetric matrix with eigenvalues λ_i and corresponding eigenvectors ϕ. What are the important properties of the λ_i and ϕ_i which follow from A being symmetric.

(c) Let $\|\cdot\|$ denote the Euclidean vector norm and also use the same notation for the induced matrix norm. By noting that any vector can be expressed as a linear combination of the ϕ_i, prove that

$$\|A\| = \max_{1 \le i \le m} |\lambda_i|.$$

(d) Calculate $\mathcal{K}(A)$ in the Euclidean norm for a symmetric positive definite matrix.

(Problem II)

(10 points). **Quadrature**

(a) Define the composite trapezoidal rule for the approximate integration of

$$I := \int_a^b f(x)dx$$

over n intervals of equal length $h = (b-a)/n$. State the magnitude of the error in terms of h, under an assumption on the smoothness of f which you should state.

(b) Show the weights $\{w_i\}_{i=1}^n$ and nodes $\{x_i\}_{i=1}^n$ in the Gaussian quadrature formula

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=1}^{n} w_j f(x_j)$$

satisfy

$$\sum_{j=1}^{n} w_j (x_j)^i = 0, \quad i = 1, 3, \ldots, 2n-1$$

and

$$\sum_{j=1}^{n} w_j (x_j)^i = \frac{2}{i+1}, \quad i = 0, 2, \ldots, 2n-2.$$

SOLUTIONS:
Numerical Analysis

Solutions to Comprehensive: Numerical Analysis (30 points). Fall 1994

(Problem I)
(15 points). **Linear Systems and Matrices**

(a) Define the condition number $\mathcal{K}$ of a matrix A. If

$$Ax = b, \quad Ay = b + r$$

state the upper and lower bounds relating the relative error $\frac{\|x-y\|}{\|x\|}$ to the relative peturbation $\frac{\|r\|}{\|b\|}$. *Briefly* discuss the implications of these inequalities for the numerical solution of linear systems.

SOLUTION: Definition is

$$\mathcal{K}(A) = \|A\|\|A^{-1}\|.$$

The basic relationship is

$$\frac{1}{\mathcal{K}(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|x-y\|}{\|x\|} \leq \mathcal{K}(A)\frac{\|r\|}{\|b\|}.$$

This is important in numerical analysis, because backward error analysis shows that, instead of solving $Ax = b$ the computed solution exactly solves $Ay = b + r$ for some small vector r. The basic relationship above then shows the confidence with which we may interpret the numerical solution: if $\mathcal{K}(A)$ is large, the numerical errors may be large.

(b) Let A be an $m \times m$ symmetric matrix with eigenvalues λ_i and corresponding eigenvectors ϕ. What are the important properties of the λ_i and ϕ_i which follow from A being symmetric.

SOLUTION: The eigenvalues are real. The eigenvvectors are orthogonal so that

$$\phi_i^T \phi_j = \delta_{ij}$$

where $\delta_{ij} = 0$ for $i \neq j$ and $\delta_{ij} = 1$ for $i = j$.

(c) Let $\|\cdot\|$ denote the Euclidean vector norm and also use the same notation for the induced matrix norm. By noting that any vector can be expressed as a linear combination of the ϕ_i, prove that

$$\|A\| = \max_{1 \leq i \leq m} |\lambda_i|.$$

SOLUTION: Let

$$v = \sum_{j=1}^{m} a_j \phi_j.$$

Then

$$\|v\|^2 = \sum_{j=1}^{m} |a_j|^2.$$

Also

$$Av = \sum_{j=1}^{m} \lambda_j a_j \phi_j$$

so that

$$\|Av\|^2 = \sum_{j=1}^{m} \lambda_j^2 |a_j|^2.$$

Thus

$$\|Av\|^2 \leq R^2 \|v\|^2.$$

Thus

$$\|A\|^2 := \sup_{\|v\|=1} \|Av\| \leq R^2.$$

Hence $\|A\| \leq R$. To prove that $\|A\| = R$ note that $A\phi_l = \lambda_l \phi_l$ and choosing that value of l for which $|\lambda_l|$ is maximized we see that $\|A\| = R$.

(d) Calculate $\mathcal{K}(A)$ in the Euclidean norm for a symmetric positive definite matrix.

SOLUTION: We have that A^{-1} has eigenvalues λ_i^{-1}. Thus

$$\|A^{-1}\| = 1/r, \quad r := \min_i \|\lambda_i\|.$$

Hence

$$\mathcal{K}(A) = R/r.$$

(Problem II)

(10 points). **Quadrature**

(a) Define the composite trapezoidal rule for the approximate integration of

$$I := \int_a^b f(x)dx$$

over n intervals of equal length $h = (b-a)/n$. State the magnitude of the error in terms of h, under an assumption on the smoothness of f which you should state.

SOLUTION: The rule is

$$I \approx \frac{1}{2}[f_0 + f_n] + [f_2 + f_3 + \ldots + f_n].$$

Here $f_j = f(x_j)$ and $x_j = a + jh$. The error is $\mathcal{O}(h^2)$ provided $f \in C^2([a,b], R)$.

(b) Show the weights $\{w_i\}_{i=1}^n$ and nodes $\{x_i\}_{i=1}^n$ in the Gaussian quadrature formula

$$\int_{-1}^1 f(x)dx \approx \sum_{j=1}^n w_j f(x_j)$$

satisfy

$$\sum_{j=1}^n w_j (x_j)^i = 0, \quad i = 1, 3, \ldots, 2n-1$$

and

$$\sum_{j=1}^n w_j (x_j)^i = \frac{2}{i+1}, \quad i = 0, 2, \ldots, 2n-2.$$

SOLUTION: Gaussian integration is chosen to accuirate on all polynomials of degree $j : 0 \leq j \leq 2n-1$. This is equivalent to asking that the rule exactly integrate x^j for $j = 0, \ldots, 2n-1$. Thus

$$\int_{-1}^1 x^j dx = \sum_{j=1}^n w_i x_i^j.$$

Evaluating the integral gives the result.

Computer Science Department
Stanford University
Comprehensive Examination in Software Systems
Autumn 1994

October 20, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.
2) The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.
3) The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.
4) This exam is CLOSED BOOK. You may NOT use notes, articles, books, computer, etc.
5) Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; your can also get credit for realizing that certain approaches are incorrect.
6) If you are convinced you need to make an assumption to answer a question, state your assumption(s) as well as the answer.
7) Be sure to provide justification for your answers.

Comprehensive Exam: Software Systems (60 points)

1) (16 points) This question asks you to implement a barrier synchronization function using only semaphores and a small number of shared variables. A barrier synchronization function waits until the specified number of processes arrive at the "barrier" before allowing any of the processes to continue. For example assume N processes execute the following code fragment:

```
Func1();
barrier(N);
Func2();
barrier(N);
Func3();
```

The barrier should ensure that no process starts executing `Func2()` before all N of the processes have executed `Func1()`. Similarly, by the time that the first process calls `Func3()` all processes should have returned from `Func2()`. You function should take a single argument, N, the number of processes participating in the barrier. It should also work correctly on the above code fragment and contain no busy waiting.

2) (8 points) Some computer systems have been designed recently with a larger physical memory address space than they have virtual address space. In other words they have more fewer bits of virtual address space than physical memory address space. Explain why this is not a totally unreasonable design for a computer system. Be sure to indicate what limits the design imposes.

3) (10 points) As the price of DRAM memory has improved relative to that of magnetic disk space, the ratio of the amount of physical memory to the amount of backing store has been has been getting larger. Some system have as much or more physical memory than backing store (swap space). In response to this, some virtual memory systems have been modified to allocate backing store in a different way. Rather than allocating the backing store when a virtual page is first created, the backing store is allocated only when the page is first paged out.

   a) Describe the benefits of this change.

   b) Describe the problems introduced by it.

4) (8 points) In some computer systems the maximum size of a transfer to or from an I/O device is limited to a relatively small size (e.g. 32 kilobytes) due to historical artifacts. On these systems, would there be any advantages of using a file system with a block size larger than the maximum I/O transfer size?

5) (8 points) What is the difference between starvation and deadlock?

6) (10 points) Frequently when sending data over a network it is beneficial to both encrypt the data for security and compress the data to decrease the transfer time. Which order would you suggest these operations be performed? Be sure to justify your answer.

SOLUTIONS:
Software Systems

Comprehensive Exam: Software Systems Solutions (60 points)

Oct 20, 1994

1) (16 points) This question asks you to implement a barrier synchronization function using only semaphores and a small number of shared variables. A barrier synchronization function waits until the specified number of processes arrive at the "barrier" before allowing any of the processes to continue. For example assume N processes execute the following code fragment:

```
Func1();
barrier(N);
Func2();
barrier(N);
Func3();
```

The barrier should ensure that no process starts executing `Func2()` before all N of the processes have executed `Func1()`. Similarly, by the time that the first process calls `Func3()` all processes should have returned from `Func2()`. You function should take a single argument, N, the number of processes participating in the barrier. It should also work correctly on the above code fragment and contain no busy waiting.

```
int numwaiters = 0;
Semaphore mutex = 1;
Semaphore waiters = 0;
Semaphore release = 0;

barrier(int N) {
   P(mutex);
   numwatiers += 1;
   if (numwaiters == N) {
      for (int i = 1; i < N; i++) //Last one wakes all
         V(wait);
      for (i = 1; i < N; i++) // Wait for all to exit
         P(release);
      numwaiters = 0;
      V(mutex);
   } else {
      V(mutex);
      P(wait);
      V(release);
   }
}
```

2) (8 points) Some computer systems have been designed recently with a larger physical memory address space than they have virtual address space. In other words they have more fewer bits of virtual address space than physical memory address space. Explain why this is not a totally unreasonable design for a computer system. Be sure to indicate what limits the design imposes.

Having more memory than virtual address space means that a single process can not easily take advantage of the all the physical memory in the system. Multiple processes such as in a multiprogramming workload can take advantage of the extra memory. So can OS data structures such as the file cache.

3) (10 points) As the price of DRAM memory has improved relative to that of magnetic disk space, the ratio of the amount of physical memory to the amount of backing store has been has been getting larger. Some system have as much or more physical memory than backing store (swap space). In response to this, some virtual memory systems have been modified to allocate backing store in a different way. Rather than allocating the backing store when a virtual page is first created, the backing store is allocated only when the page is first paged out.

   a) Describe the benefits of this change.

Besides avoiding the extra work required to do the allocation and deallocation of the swap space, it also allows the amount of virtual memory in-use to be as big as the sum of the

swap space and the physical memory. On some machines this can be much larger than the amount of swap space.

b) Describe the problems introduced by it.

It possible to enter a deadlock condition because there is no space to page out of a page.

4) (8 points) In some computer systems the maximum size of a transfer to or from an I/O device is limited to a relatively small size (e.g. 32 kilobytes) due to historical artifacts. On these systems, would there be any advantages of using a file system with a block size larger than the maximum I/O transfer size?

Large file system block sizes also reduce the size of the metadata for the file. This reduces the amount of disk space needed by the file system metadata and can result in faster access to the file's contents.

5) (8 points) What is the difference between starvation and deadlock?

In both starvation and deadlock the system is not making forward progress. The difference is that in deadlock there is no way it can go forward while in starvation there is a way but it's not happening.

6) (10 points) Frequently when sending data over a network it is beneficial to both encrypt the data for security and compress the data to decrease the transfer time. Which order would you suggest these operations be performed? Be sure to justify your answer.

By compressing the data first we remove redundant information that can make the job of the person trying to break the encryption easier.