April 1987

Report No. STAN-CS-87-1154

A Programming-and Problem-Solving Seminar

by

Tomas G. Rokicki and Donald E. Knuth

Department of Computer Science

Stanford University
Stanford, CA 94305

Computer Science Department

A PROGRAMMING AND PROBLEM-SOLVING SEMINAR

by

Tomas G. Rokicki and Donald E. Knuth

This report contains edited transcripts of the discussions held in Stanford's course CS304, Problem Seminar, during winter quarter 1987. Since the topics span a **large** range of ideas in computer science, and since most of the important research paradigms and programming paradigms were touched on during the discussions, these notes may be of interest to graduate students of computer science at other universities, as well as to their professors and to professional people in the "real world."

The present report is the seventh in a series of such transcripts, continuing the tradition established in STAN-CS-77-606 (Michael J. Clancy, 1977), STAN-CS-79-707 (Chris Van Wyk, 1979), STAN-CS-81-863 (Allan A. Miller, 1981), STAN-CS-83-989 (Joseph S. Weening, 1983), STAN-CS-83-990 (John D. Hobby, 1983), and STAN-CS-85-1055 (Ramsey W. Haddad, 1985).

-This research and/or preparation was supported in part by the National Science Foundation under grant CCR-86-10181.

Table of Conteuts

Index to Problems in the Seven Seminar Transcripts

STAN-CS-77-606	Map Drawing
	Natural Language Numbers
	Efficient One-Sided List Access
	Network Design
	Code Generation
STAN-CS-79-707	The Camel of Khowârizm
	Image Cleanup
	Cubic Spline Plotting
	Dataflow Computation with Gosper Numbers
	Kriegspiel Endgame
STAN-CS-81-863	Alphabetic Integers
	King and Queen versus King and Rook
	Unrounding a Sequence
	Grid Layout
	File Transmission Protocols
STAN-CS-83-989	Penny Flipping
	2 x 2 x 2 Rubik's Cube
	Archival Maps
	Fractal Maps
	Natural Language Question Answering
STAN-CS-83-990	Bulgarian Solitaire
	Restricted Planar Layout
	Dynamic Huffman Coding
	Garbage Collection Experiments
	PAC-Land and PAC-War
STAN-CS-884055	Monotonic Squares
	Code Breaking
	Hasdware Fault Detection
	Distributed Stabilit}
	High-Tech Art
This report	Toetjes
	Multigrades
	Type Checking
	Discovering the Wheels
	Flashy Signs

Cast of Characters

Professor

DEK	Donald E. Knuth

Teaching Assistant

TR	Tomas G. Rokicki

Students

NB	Nancy Blachman
DB	Derrick R. Burns
CC	Craig D. Chambers
RC	Rohit Chandra
MD	Marcia A. Derr
TF	Tomás Feder
MH	Max Hailperin
BH	Barry Hayes
TH	Thomas A. Henzinger
JH	Jakobsson Håkan
DM	David K. Mellinger
IM	Inderpal S. Mumick
SN	Steven Nowick
RM	Martin C. Rinard
RK	Kelly B. Roach
ER	Ed Rothberg
DS	David H. Salesin
SS	Scott Seligman
JS	Jack S. Snoeyink
CS	Caslos S. Subi
AW	Alex Wang
EW	Elizabeth S. Wolf
MW	Michael Wolverton
RZ	Ramin D. Zabih

Data Sheet CS304/1987

Class: CS304, "Problem Seminar." Meets 11:00–12:15 Tuesdays and Thursdays in room 301, Margaret Jacks Hall.

Discussion leader: Don Knuth. Office is 328 Jacks, phone 723-4367. [Please make an appointment with Phyllis Winkler, 326 Jacks, if you want to talk to him outside of normal class hours .]

Teaching assistant: Tomas Rokicki. Office is 322 Jacks, phone 723-1646; computer address ROKICKI @ SUSHI. Office hours on Mondays and Wednesdays, 11–12.

Problems: There ase five problems and we will take them in order, spending about two weeks on each. Also, there will be a special two-day Trivia Hunt. Students should work in teams of two or three on each problem, and also when participating in the Trivia Hunt. No two students should be members of the same team more than twice; this way everybody will get to know almost everybody else. We stress cooperation and camaraderie, not concealment and competition! (Exception: The Trivia Hunt will be a contest to see which team can score the most points.)

Computer use: You may use any computer you can steal time on. Problem 5 will be done in the Macintosh Lab in Sweet Hall.

Grading: You should hand in a well-documented listing of your computer programs for each problem, along with a writeup that describes the approaches you took. This writeup should include a discussion of what you did that worked or didn't work, and (when appropriate) it should also mention what you think would be promising approaches to take if there were extra time to pursue things further. Your written work will be graded on the basis of style, clarity, and originality, as well as on program organization, appropriateness of algorithms, efficiency, and the correctness of the results. These grades will be given on an A-E scale for your own information, but your overall grade for the course will be either 'A' or 'nothing'.

Class notes: Classroom discussions will mostly involve the homework problems, but we will try to emphasize general principles of problem solving that are illustrated by our work on the specific problems that come up. Everyone is encouraged to participate in these discussions, except that nobody but Knuth will be allowed to talk more than three (3) times per class period. After class, the TA will prepare notes about what happened; therefore you will be able to participate freely in the discussions instead of worrying about your own note-taking. These class notes will eventually be published as a Stanford report; similar reports from previous years can be found in the library [CS606 (Michael J. Clancy, 1977); CS707 (Chris Van Wyk, 1979); CS863 [Allan A. Miller, 1981); CS989 (Joseph S. Weening, 1982); CS990 (John D. Hobby, 1983); CS1055 (Ramsey W. Haddad, 1985).

Special dates: The Trivia Hunt will begin at 12:15pm on Tuesday, January 20, and it will end at 11:00am on Thursday, January 22. Demonstration Day for Problem 5 will be March 17 (in Sweet Hall).

Caveat: This course involves more work than most. other 3-unit courses at Stanford.

Problem 1, due January **20: Toetjes.**

This problem was suggested by Sape Mullender, who described it as follows:

> In Amsterdam, where I grew up, dessert is usually referred to as "toetje" (Dutch for "afters"). The problem of allocating a left-over toetje to one of the children in my family became the Toetjes Problem. The algorithm was the following: First my mother would choose a secret number between one and a hundred. Then the children, in turn, youngest to oldest, could try to guess the number. After the last guess my mother would tell whose guess was closest to her secret number and the winner would get the toetje.
>
> It quickly became clear to us kids that a clever strategy for choosing the number would help to increase one's chances of winning. In a family of two children, for instance, the first child would have to choose the middle number, 50, and the other child could then choose either 51 or 49. Years later, when our reasoning skills were more developed we could even do the optimal choice for three kids. (Naturally, we assumed that each child would attempt to maximize his or her chances without resorting to conspiracy with one of the others.) The first child had to choose 25 (or 75), the second 75 (or 25), and the third chould choose any number between 26 and 74, influencing the other two kids' chances, but not his or her own.
>
> Now that I have a degree in mathematics, the problem still puzzles me: Given that the secret number is chosen randomly from the interval $[0,1]$, what is the optimal strategy for choosing a number for the ith child in a family of n children? The ith child knows what the first $i-1$ children chose, and knows that all the children choose optimally (i.e., choose to maximize their own chance without consideration for the chances of any other child in particular).
>
> I grew up in a family of five children, and I never worked out the optimal strategy for $n = 5$. Fortunately, I was the oldest, so choosing optimally was easy for me. But I don't think it mattered very much what I chose; I think my mother cheated. I think she chose the number after we had all announced our guesses, because I can't remember anyone ever winning two times in succession.

Like most research problems, this one is incompletely specified, and we'll have to make suitable assumptions. For example, it isn't clear what policy is assumed in case of ties. Furthermore, the stated analysis of the cases $n \leq 2$ isn't quite correct.

However, it's an excellent problem! Our goal will be to analyze the cases $n <= 5$ exactly in the discrete case of numbers chosen between 1 and 100, under various tie-breaking assumptions, and to analyze the continuous case of numbers in $[0,1]$ for as many values of n as we can.

Problem 2, due February **3:** **Multigrades.**

Let the notation $a_1, a_2, \ldots,$ a, $\overset{k}{=} b_1, b_2, \ldots, b_s$ stand for the set of k equations

$$a_1^h + a_2^h + \cdots + a_s^h = b_1^h + b_2^h + \cdots + b_s^h, \qquad \text{for } 1 \le h \le k.$$

A *multigrade* of order k is a solution to these equations in integers, when $s = k + 1$ and when the a's and b's have no elements in common.

Multigrades are known to exist for $1 \le k \le 9$. For example,

$$1,2 \overset{1}{=} 0,3$$
$$1,2,6 \overset{2}{=} 0,4,5$$
$$1,2,9,10 \overset{3}{=} 0,4,7,11$$
$$1,2,10,14,18 \overset{4}{=} 0,4,8,16,17$$
$$1,2,12,14,24,25 \overset{5}{=} 0,4,9,17,22,26$$
$$1,13,38,44,75,84,102 \overset{6}{=} 0,18,27,58,64,89,101$$
$$1,2,11,20,30,39,48,49 \overset{7}{=} 0,4,9,23,27,41,46,50$$
$$1,17,41,65,112,115,168,174,198 \overset{8}{=} 0,24,30,83,86,133,157,181,197$$

The purpose of this problem is to try to find a multigrade of order 10 or more.

Problem 3, due February 17: **Type checking.**

This problem is based on the CHEX language, which is designed to allow the formalization and verification of mathematical proofs. C **HEX** has a fairly simple syntax, based on the following syntax for expressions:

(exp) - ⟨params⟩ ⟨func⟩ ⟨args⟩
⟨params⟩ - (empty) | (⟨param list))
⟨param list) ⟶ ⟨param group) | ⟨param list) ; ⟨param group)
⟨param group) - ⟨variable list) : ⟨type⟩
(variable list) - ⟨ident⟩ | (variable list) , ⟨ident⟩
⟨type⟩ ⟶ ⟨exp⟩
⟨func⟩ ⟶ ⟨ident⟩
(args) - (empty) | ((arg list))
⟨arg list) - (exp) | (arg list) , (esp)

An ⟨ident⟩ is a string of letters and/or digits and/or underline (' _ ') characters. If V is a ⟨variable list⟩, I is an ⟨ident⟩, and T is a ⟨type⟩, the construction '$V, I : T$' is equivalent to '$V : T; I : T$'; hence we can assume for purposes of exposition that all variable lists have length 1. Under this assumption, an expression has the general form

$$(v_1 : t_1; \ldots ; v_k : t_k)f(a_1, \ldots, a_m)$$

where $k \geq 0$, $m \geq 0$, the v's are identifiers, the t's are type expressions, f is a functional identifier, and the a's are expressions.

A "program" in **CHEX** is called a script, and it is simply a sequence of definitions that have the following syntax:

⟨def⟩ $\longrightarrow$ ⟨ident⟩ ⟨params⟩ : = ⟨equiv⟩ : ⟨type⟩ .
⟨equiv⟩ $\longrightarrow$ ⟨exp⟩ | #
⟨script⟩ $\longrightarrow$ ⟨empty⟩ | ⟨script⟩ ⟨def⟩

(See the example of a **CHEX** script at the end of this problem statement.)

The ⟨ident⟩ at the beginning of a ⟨def⟩ is defined by that ⟨def⟩; it is called an internal identifier if the ⟨equiv⟩ is an expression, otherwise it's called an external identifier. The ⟨ident⟩s in a ⟨variable list⟩ are called bound variables. All other occurrences of an ⟨ident⟩ are in the context of a ⟨func⟩, and these must match either bound variables or defined identifiers. The corresponding identifier is obtained by proceeding backward from the ⟨func⟩ of an expression to all preceding bound variables in the same expression, then outward to the preceding bound variables in enclosing expressions or the enclosing definition, then backward through all of the preceding defined identifiers, until the first match is found. Thus, for example, the bound variable v_i in

$$(v_1 : t_1; \ldots ; v_k : t_k)f(a_1, \ldots, a_m) \qquad (*)$$

may be referred to in the subexpressions $t_{i+1}, \ldots, t_k, f, a_1, \ldots, a_m$, unless the identifier v_i is bound again in one of those expressions; but a variable that's bound in a_1 can't be referred to in a_2. Identifiers in the ⟨type⟩ of a ⟨def⟩ can refer to bound variables of the ⟨equiv⟩ or ⟨params⟩ in that same ⟨def⟩ .

Some expressions and definitions that obey the syntax above are invalid because they break CHEX's rules. For example, an ⟨exp⟩ that uses a previously undefined ⟨ident⟩ is invalid. We shall say an expression or definition is valid if it doesn't break any of the rules given above or below.

Every valid expression has a domain $(V_1 : K_1 ; \ldots ; V_T : K_T)$, an expanded body $F(A_1, \ldots, A_M)$, and a range $\varphi(\alpha_1, \ldots, \alpha_\mu)$. These three concepts will be defined recursively by starting with small expressions and working up to larger ones. The expanded form of an expression is its domain followed by its expanded body. Our definitions will be such that the expanded form contains no appearances of internal identifiers; all such identifiers will be "substituted out ."

If e and t are espressions, the statement 'e has type t' means that the domain of e is the domain of t and the range of e is the expanded body of t, except for possible renaming of bound variables. The definition of an internal identifier is invalid unless the stated (exp) has the stated (type).

Given an expression $(*)$ whose domain, expanded body, and range are to be defined, we first define the domain and range of its functional identifier f. If f is a bound variable v;, its domain and range are the domain and expanded body of t_i. Otherwise f was defined in some (def) whose $\langle$params$\rangle$ are $(w_1 : u_1; \ldots ; w_l : u_l)$ and whose (type) has domain $(W_1 : S_1; \ldots ; W_L : S_L)$ and expanded body $\psi(\beta_1, \ldots, \beta_\nu)$. In this case the domain of f is $(w_1 : u'_1; \ldots ; w_l : u'_l; W_1 :$ u,;... ; $W_L : U_L)$, where u'_i is the expanded form of u;; the range of f is $\psi(\beta_1, \ldots, \beta_\nu)$.

Let the domain of f be $(x_1 : X_1; \ldots ; x_n :$ X,). The expression $(*)$ is invalid unless $m \le n$. If $m = n -$ d, we replace $(*)$ by

$$(v_1 : t_1; \ldots ; v_{k+d} : t_{k+d}) f(a_1, \ldots, a_n) \qquad (**)$$

by setting $(t_{k+1}, \ldots, t_{k+d})$ to the last d types in f's domain and by letting $a_{m+i} = v_{k+i}$ for $1 \le$ i $\le$ d. Thus we can assume that $m = n$.

The expression $(**)$ is invalid unless *a;* has type X_i for $1 \le i \le n$. The domain of $(**)$ is $(v_1 : t'_1; \ldots ; v_K : t'_K)$, where K = k + d and t'_i is the expanded form of t_i. The range of $(**)$ is the range of f.

Thus it remains only to define the expanded body of $(**)$. If f is a bound variable or an external identifier, the expanded body is simply $f\ (a'_1, \ldots, a'_n)$, where a'_i is the expanded form of a;. Otherwise f is an internal identifier whose definition has the expanded form

$$f(x_1 : X_1; \ldots ; x_l : X_l) := (x_{l+1} : X_{l+1}; \ldots ; x_n : X_n)\Phi(B_1, \ldots, B_N)$$
$$: (W_1 : U_1; \ldots ; W_L : U_L)\psi(\beta_1, \ldots, \beta_\nu).$$

The expanded body of $(**)$ is obtained by substituting *a;* for x_i in $\Phi(B_1, \ldots, B_N)$.

If a_i has an empty domain, x_i will not be followed by a. list of arguments in the body $\Phi(B_1, \ldots, B_N)$, so substitution is simply replacement. Otherwise, a_i is a function, and substitution for x_i involves the substitution of x_i's arguments into a_i's body. This recursive substitution process eventually terminates, because a **CHEX** script cannot define a. function of a function of a function · . . ad infinitum.

The special definition

atom := # : atom.

is assumed to be implicitly present just before every **CHEX** script. An expression used as a (type) is valid only if its range is ' atom '.

Well, those are the rules of **CHEX.** Problem 3 is to write a program that checks the validity of a **CHEX** script. Furthermore, the program should calculate the length of the expanded form of each (def $\rangle$. This length should be computed as the total number of 'tokens', where a token is either an $\langle$ident$\rangle$ or one of the punctuation marks ' (', ' , ', ' ; ', ' : ', ' = ') ' # ', ') ', ' . ', used in the language. The following is an example **CHEX** script.

bool := #: **ntom**. { there's a. special kind of atom called a *bool* }
proof (b : **bool)** := #: *atom*. { and another called a. **proof** of a. *bool* ; logicians say $\vdash b$ }

eq$(a$: *atom*; x, y : *a*) := **#**: ***bool***.

{ *eq* is a proposition that two variables of type *a* are equal}

is$(a$: *atom*; x, y : *a*) := *proof* (*eq*(a,x,y)): *atom*. { this asserts that x = y can be proved}

identical (a : *atom*; x : *a*) := **#**: *is* (a,x,x). { an axiom: $\vdash$ x $= x$ }

eq_axiom (a : *atom*; x, y, z : *a*; p : *is* (a, x, z); q : *is* (a, y, z)) := **#**: *is* (a, x, y).

{ another axiom: if $\vdash$ x = z and $\vdash$ y = z then $\vdash$ x = y }

eq_symmetry$(a$: *atom*; x, y : *a*; p : ***is***(a, x, y)) := *eq-axiom(a) y, x, y, identical (a, y), p)*:

. *is*(a, y, x). { previous axioms are used here to deduce $\vdash$ y = x from $\vdash$ x = y }

eq_transitivity (a : *atom* ; x, y, z : *a*; p: ***is***(a,x,y); q : ***is***(a,y,z)) :=

eq_axiom$(a, x, z, y, p,$ ***eq_symmetry***(a, y, z, q)): ***is***(a,x,z).

{ $\vdash x = y$ and $\vdash y = z$ implies $\vdash x = z$ }

eq_functionality$(a, b$: *atom*; x, y : *a*; p: ***is***(a, x, y); ***f*** : (x : *a*) *b*) := **#**: *is* $(b, f(x), f(y))$.

{ $\vdash$ x = y implies $\vdash$ $f(x)$ = $f(y)$ for any function f }

nat := **#**: *atom*. { another primitive atom, representing the nonnegative integers }

0 := **#**: ***nat***. { 0 is a nonnegative integer by definition}

succ$(x$: *nat*) := **#**: *nat*. { the successor function}

succ_unique $(x, y$: *nat*; p: *is* (*nat*, ***succ***(x), ***succ***(y))) := **#**: *is* (*nat* , x, y).

{ another axiom: different *nuts* can't have the same ***succ*** }

nat-pred (x : *nat*) := ***bool*** : *atom*. { a predicate defined on ***nats*** }

nat_imp (p : *natpred* ; x : *nat* ; q : *proof* $(p(x))$) := *proof* (p(***succ*** (x))): *atom*.

{ an inductive implication }

induction(p : *nat-pred* ; q : *proof* $(p(O))$; r : ***nat_imp***(p)) := **#**: (x : *nat*)***proof*** $(p(x))$.

{ Peano's axiom }

sum$(x, y$: *nut*) := **#**: *nat*. { the sum of two *nats* is a ***nat*** }

sum_ax1 (x : ***nat***) := **#**: ***is***(***nat***, ***sum***$(x, 0), x$). { $\vdash x + 0 = x$ }

sum_ax2 (x, y : ***nat***) := **#**: ***is***(***nat***, ***sum***$(x,$ ***succ***$(y))$, ***succ***(***sum***(x, y))).

{ $\vdash$ x + y' = (x + y)'}

thml (x, y : *nat*) := ***eq***(***nat***, ***sum***(***succ***$(x), y)$, ***succ***(***sum***(x, y))): *bool*. {x' + y = (x + y)'

step1 (x : *nat*) := ***sum_ax1*** (***succ***(x)): ***is***(***sum***(***succ***$(x), 0)$, ***succ***(x)). { $\vdash x'$ + 0 = x}

step2 (x : *nat*) := ***eq_functionality*** (*nat* , *nat* , ***sum***$(x, 0), x$, ***sum_ax1*** (x), ***succ***):

is(***succ***(***sum***(x,O)), ***succ***(x)). { $\vdash$ (x + 0)' = x' }

step3 (x : *nat*) :=

eq_axiom(***nat***, ***sum***(***succ***$(x), 0)$, ***succ***(***sum***$(x, 0)$), ***succ***(x), *step1* (x), *step2* (x)):

proof (*thml* $(x, 0)$). { hence I-x' + 0 = (x + 0)' }

step4 (x, y : *nat*; q : *proof* (*thml* (x, y))) :=

eq_axiom(***nat***, ***sum***(***succ***$(x), y)$, ***sum***$(x,$ ***succ***$(y))$, ***sum***(x, y), q, *sum-ax2* (a, y)):

is (***sum***(***succ***$(x), y)$, ***sum***$(x,$ ***succ***$(y))$).

{ assuming ***thm1*** *as* an induction hypothesis, we can prove that x' + y = x + y' }

step5 (x, y : *nat*; q : *proof* (*thml* (x, y))) :=

eq_functionality(***nat***, ***nat***, ***sum***(***succ***$(x), y)$, ***sum***$(x,$ ***succ***$(y))$, *step4* (x, y, q), ***succ***):

is (***succ***(***sum***(***succ***$(x), y)$), ***succ***(***sum***$(x,$ ***succ***$(y))$)).

{ from which it follows that (x' + y)' = (x + y')' } *step6* $(x, y$: *nat*$)$:=

   sum_ax2 $(succ(x)$, *y*): *is*$(sum(succ(x), succ(y)), succ(sum(succ(x), y)))$.

   { and x' + y' = (x' + y)' by definition }

step7$(x, y$: *nnt*; *q* : *proof (thml (x, y)))* := *eq-transitivity*$(nat, sum(succ(x),$

   $succ(y)), succ(sum(succ(x), y)), succ(sum(x, succ(y))),$ *step6* (x, y), *step5* (x, y, q)):

   proof (*thm1* $(x$, *succ* $(y)))$.

   { hence the induction hypothesis yields x' + y' = (x + y')' as desired }

qed_thm1 $(x$: *nat*$)$:= *induction*(*thm1* (x), *step3* (x), *step7*(x)): $(y$: *nat*)*proof* (*thm1* (x, y)).

thm2$(x, y$: *nat*$)$:= *eq*$(nat, sum(x, y), sum(y, x))$: *bool*. $\{x + y = y + x\}$

Problem 4, due March 3: **Discovering the wheels.**

A certain 3-wheel slot machine in a Nevada casino lets you see four consecutive positions of the wheels after they have stopped spinning.

Consider the following sequence of observations (recorded on April 24, 1976):

LOLC JOJB LB07	LOLC C7LJ BOBL	LCLJ OJBJ 07LB	LJLO JBJC JBOB
OLCL C7LJ LBOB	07BL JCJO BJBO	LOLC BJC7 BJBO	JLOL JCJO BLBO
JLOL BJC7 LBOB	7BL0 BJCJ LB07	JLOL JCJO LBOB	LCLJ OJBJ LBJB
LCLJ OJBJ BLBO	CLJL JBJC LBOB	OLCL BJCJ OBLB	LOLC JOJB LBOB
LJLO JOJB BJBO	LCLJ JOJB LBOB	OLCL BJCJ BOBL	LJLO CJOJ BLBO
LOLC CJOJ BOBL	JL07 OJBJ BLBJ	LOLC OJBJ BOBL	CLJL BJCJ BJBO
LO7B JOJB JBOB	LOLC JBJC LBOB	07BL JBJC LB07	LOLC CJOJ OBLB
LO7B OJBJ BOBL	LO7B OJBJ LB07	BLOL JOJB BOBJ	OLCL LJCJ BO7L
LOLC CJOJ BO7L	OLCL OJBJ BOBL	LJLO OJBJ BLBO	JL07 C7LJ BJBO
OLCL JOJB LBJB	OLCL JOJB OBLB	JLOL JBJC BLBJ	LO7B CJOJ LBOB
LO7B CJOJ BOBL	07BL JCJO BOBL	LOLC 7LJC 07LB	LO7B JCJO BOBL
JLOL CJOJ BLBO	LCLJ JOJB LBOB	JL07 OJBJ BLBO	LOLC JCJO JBOB
LOLC JOJB BOBL	JLOL CJOJ BLBO	CLJL C7LJ OBLB	OLCL JBJC BLBO
LO7B JBJC BLBJ	JLOL CJOJ LBOB	07BL JOJB JBOB	LJLO BJCJ BLBO
CLJL LJCJ BLBJ	LOLC CJOJ LBOB	CLJL JCJO OBLB	LCLJ OJBJ BJBO
LCLJ BJCJ 7LB0	LJLO OJBJ 07LB	CLJL C7LJ 07LB	OLCL JOJB LBOB
LJLO BJCJ LBJB	LOLC JC7L BJBO	LJLO LJCJ LBJB	7BL0 CJOJ LBJB
OLCL 7LJC 7LB0	JL07 BJC7 OBLB	LJLO JOJB OBLB	JL07 JOJB OBLB
LJLO JOJB OBLB	JL07 OJBJ BLBJ	OLCL CJOJ BOBL	LCLJ JCJO BLBO
LJLO JBJC LBOB	LCLJ JBJC LBOB	LOLC JOJB BLBO	OLCL OJBJ LBOB
LCLJ 7LJC BLBO	CLJL OJBJ BO7L	JL07 C7LJ BOBL	LCLJ CJOJ BOBL
CLJL BJC7 OBLB	LOLC JC7L BOBL	LJLO 7LJC BOBL	CLJL BJC7 BO7L
JL07 JOJB LBOB	LJLO JOJB BJBO	LCLJ CJOJ LBOB	CLJL OJBJ BOBL

Here B, C, J, L, and O stand respectively for bell, cherry, jackpot, lemon, and orange. The payoff table is

777	200	
JJJ	100	
BBB	18	
BBJ	18	
LLL	14	
LLJ	14	
OOO	10	
OOJ	10	
CCX	5	(X is any symbol except C)
CXY	2	(Y is any symbol except C)

The machine is fairly small, hence it is clear by physical observation that no wheel can contain more than 25 symbols.

The problem is to reconstruct the patterns of symbols on the wheels, as well as possible, based on this data. For each $n \leq 25$ and each of the three wheels, determine all of the "reasonably likely" candidate patterns and give a quantitative estimate of the proba.bility that each candidate is consistent with the data above.

Solve this problem also for subsets of the data: How certain would we be about the wheel configurations if we had only 3/4 of the above data? Only half? Only 1/4?

Consider also what can be deduced if we are able to see only three consecutive positions on each wheel (e.g., if the first reading is simply 'LOL JOJ LBO'). What if we are able to see only two consecutive positions at once (e.g., 'LO JO LB'? And what if we can see only a single symbol (e.g., 'L J L')?

What is the expected payoff?

Problem 5, due March 17: **Flashy signs.**

The Digiflash Sign Corporation is a new startup company that manufactures signboards containing an 8 x 256 array of lights, each of which is either 'off' or 'on'. The sign is attached to a standard ASCII keyboard; there is also some random-access memory, and a small processor chip.

Thus, a Digiflash sign can be thought of as a microcomputer whose only input device is the keyboard, and whose only output device is the 8 x 256 array of lights.

Unfortunately, the Digiflash people know nothing about software or user interfaces. They wish to hire the members of CS304 as consultants, so that users of Digiflash signs can easily prepare display sequences.

Digiflash users know nothing about computers and care less. But they want to set the machine up so that it will display repeating patterns of lights; these patterns are supposed to catch people's eyes and communicate important messages. The users want lots of features, but they don't want to spend time learning how to use the features.

Design a suitable user interface, and write the user manual. *The manual should be at most* one page long. Try to include as many features as possible, subject to the condition that they are easy to learn and easy to explain.

A typical display sequence shouldn't take too long to set up, but minimum setup time is not a primary goal. The Digiflash unit is not intended to be used online (like a scoreboard), it's just supposed to be good for making display sequences that are set up once and repeated many times.

Your user interface should be programmed for Macintosh, using special Digiflash-simulation routines that will be supplied by the teaching assistant. On Demonstration Day, March 17, your interface and manual will be tested by novice users specially brought to Sweet Hall for the occasion.

6 January 1987

Introduction

Twenty-three students, Don Knuth in a classic grey three-piece suit, and yours truly crowded into room 301 of Margaret Jacks Hall for the first meeting of CS 304. A course data sheet and class enrollment sheet was passed to each student, and class began.

The first thirty minutes of class were devoted to a description of the course and a perusal of the data sheet. The course will be ten weeks long, and five problems will be attempted. True to the faith invested in them by the admissions committee, the class soon calculated that each problem should take approximately two weeks. The first problem was drafted with the realization that the comprehensive examination occurred during the first two weeks of the quarter, and its due date was set to the Tuesday following the exam.

One major goal of the course is to prepare the students for research. It will often be the case that the solution to the problems is not known, and it may occur that a problem is not solvable. The problems may not be completely stated or may be open-ended.

Another goal of the class is to establish contact between students. No pair of students may work together in a team on more than one problem. Teams will consist of two or three students, with problems three and five probably requiring the larger teams.

A third goal is to acquaint the students with the resources available at Stanford. To this end, a two-day Trivia Hunt will be held for the first time, two weeks into the quarter.

DEK congratulated everyone on their high grade for the course, and then introduced the teaching assistant. These notes will be compiled after class each day, reviewed by DEK, and ready soon after.

Everyone was asked to introduce themselves next, give their interests, where they were from, and whether they played a musical instrument or rode a bicycle. This made for an entertaining half hour and helped everyone get to know each other.

Problem 1: Toetjes

Finally, in the last half hour of the class, we started on the first problem. After everyone read the problem statement, discussion was opened.

DEK noted that we could assume the mother chose her number and wrote it down, before the children started guessing, to eliminate the maternal instinct factor on random number generation. He mentioned that the problem could be extended to the real numbers, in which case epsilon (E) becomes useful; the first player might choose $1/2$, and the second $1/2 - \epsilon$, for instance.

DEK wrote some formulas on the board for the range of numbers that won for each child. Let us order the numbers chosen by the children and call the numbers x_1 through x_n. Then, the range of numbers that win for each player (ignoring ties for the moment) is:

$$0, \frac{x_1 + x_2}{2} \qquad \text{for child } 1$$
$$\frac{x_{i-1} + x_i}{2}, \frac{x_i + x_{i+1}}{2} \qquad \text{for child } i,\ 1 < i < n$$
$$\frac{x_{n-1} + x_n}{2}, 1 \qquad \text{for child } n$$

Various ways to eliminate ties were then suggested. MH suggested splitting the toetjes among the tied children; random choice among the tied children was also suggested. The youngest child among those tied was a third suggestion; it was noted that this is the same as the first guess of those tied, since the children guessed from youngest to oldest. BH suggested that nobody (or perhaps the mother) win.

Another suggestion was that the game be played over. This was separated into two cases: restricting the next game to the two who were tied, and just starting the game over with all of the children.

DS (or was it DM?) had an interesting suggestion. Make moves that allow a possible tie illegal. Thus, it would be illegal to choose an odd integer if the least number chosen greater than yours was odd, or if the greatest number chosen less than yours was odd.

KR -suggested that the mother choose some number that is not half of an integer; for- instance, she might choose a number in the interval $(42.5, 43)$. It was shown that all numbers within that range are equivalent. It seemed plausible that the range $(42.5, 43)$ would also be equivalent to $(43, 43.5)$, but this was shown not to be the case with the chosen numbers 42 and 44. Under this scheme, the mother has 198 essentially different choices, represented by the intervals $(\frac{n}{2}, \frac{n+1}{2})$ for $1 < n < 200$.

The tie breaking ideas were then examined, and those with equivalent payoff and optimal playing strategies grouped. The divided payoff is obviously equivalent to the random choice among tied children. DEK asked whether they might be both equivalent to starting the game over. BH objected, saying that if there is a way to force a tie, even between two other players, this might be the best move. DEK mentioned that the optimal strategy, and thus the play of the children, should remain the same from game to game.

At this point, DEK recommended that everyone work out for Thursday solutions to the two-person and three-person cases for both the discrete and the continuous case and as many of the tie breaking strategies as possible.

TF mentioned that optimal play might still allow some freedom of choice which would affect the payoff for the other children. For instance, when playing between two other chosen numbers, where you play does not affect your payoff (at least, not in the continuous version), but it affects the payoff of the other children. So a 'courteous' sibling will play in the middle of an interval.

8 January 1987

Preamble

On the second day of CS 304, more discussion of bicycles and music was prompted by the inexplicable appearance of four new students in the class. The first edition of notes sold out within minutes; more haadouts are available on request. The four members of the class who are not taking the comprehensive examination were pointed out as idea.1 teammates. Attention was then quickly focused on the 'toetjes' problem.

Amble

After some initial discussion over the pronunciation of 'toetjes', DEK listed and numbered the tie breaking strategies from Tuesday.

1) Fractional payoff
2) Random choice
3) Start over
4) Youngest
5) Oldest
6) Nobody
7) Lowest guess
8) Illegal to choose move which ties
9) Mother chooses fraction $\neq \frac{n}{2}$
10) Highest guess

BH said that if a player had several equally good moves at some point, his selection could affect subsequent moves. RZ said that a tie breaking game between the two contenders (not listed in the above strategies) would be equivalent to random selection, unless there was a number exactly in the middle for the discrete case.

MR suggested that the mother must choose a different number each time in response to a suggestion that the optimal strategy would remain invariant. EW said that a child might choose an optimal move for him, and yet be in conspiracy with another child by his choice among equally good moves. This complication was removed when DEK said we could assume only self interest among the children. Yet, somebody pointed out that strategy 3 might be different from random choice, because starting over gives a tied player less than a 50% chance; hence strategy 3 makes tying less desirable.

DEK then listed different methods by which selection among equally desirable moves might be made.

a) Choose randomly
b) Choose lowest
c) Choose highest
cl) Choose middle

MH then stated that play would be influenced by the strategy chosen. TH requested a clarification of what was meant by 'optimal play'; can we take into account the selection strategy? MR asked if the other players could know each other's selection strategy.

This question was answered implicitly by the next presentation by DEK, when he used some mathematical notation to try to clarify the problem. Let us call the numbers selected $x_1 \ldots x_n$, ordered by their selection sequence. Let us introduce a function f, such that $f(x_1, \ldots, x_{i-1}) = x_i$; given the prior moves $x_1, \ldots, x_{i-1}$, this is the best move for the ith child. Let us also introduce a probability function $p_k(x_1, \ldots, x_n)$, which calculates the probability that the kth child will win, given all selected numbers. This latter function is straightforward to calculate, and the former can be calculated recursively. For instance, assuming a five player game,

$$f(x_1, \ldots, x_4) = x_5 \Rightarrow p_5(x_1, \ldots, x_5) = \max_{x \notin \{x_1, \ldots, x_4\}} p_5(x_1 \ldots x_4, x)$$

This is an algorithmic function for $f(x_1 \ldots x_4)$; it depends on the selection strategy (a-d) if several x_5's achieve the maximum. Using this function, we can calculate $f(x_1, x_2, x_3)$ as follows:

$$f(x_1, x_2, x_3) = x_4 \Rightarrow$$
$$p_4(x_1, \ldots, x_4, f(x_1, \ldots, x_4)) = \max_{x \notin \{x_1, x_2, x_3\}} p_5(x_1, x_2, x_3, x, f(x_1, x_2, x_3, x))$$

This can be extended step by step:

$$f(x_1, x_2) = x_3 \iff p_4(x_1, x_2, x_3, f(x_1, x_2, x_3), f(x_1, x_2, x_3, f(x_1, x_2, x_3))) =$$
$$\max_{x \notin \{x_1, x_2\}} p_5(x_1, x_2, x, f(x_1, x_2, x), f(x_1, x_2, x, f(x_1, x_2, x)))$$

MR mentioned that the selection strategy affects f, and therefore p. MH observed that knowing the selection strategy of the other players can cause the previously equivalent selections to be not equivalent.

DEK suggested that one example be worked out completely. Before we could begin, JS said that only the last person has a large choice of equivalent moves. He supported this with an example. If player $n - 1$ (the second to last player) moves closer to one of the other players, it will affect the play of player n, thus affecting player $n - 1$. TF looked at another angle; playing in the middle of a large interval might be a bad choice if the next player moves adjacent to you, reducing your chances; it might be smarter to move into a smaller interval where there will be less competition.

DEK then focused discussion temporarily on strategy 8 (see the above table), where a move is illegal if it can lead to a tie. MH said that under this strategy, only odd intervals can remain at the end of a game (where the parity of an interval is the parity of the difference of its bounds.) TB said that for the two player game, one must choose an even number and the other an odd. DEK observed that there were no legal moves within a gap. MR introduced a new twist; such a move might be legal if further moves made ties impossible. RZ mentioned that this might be possible if an even interval were reduced to two odd intervals.

DEK listed the possibilities; an even interval can be reduced to either two odd or two even intervals, but an odd interval must always be reduced to one even and one odd interval. Thus, a player is allowed to make an even interval only if the number of players following him is equal to or greater than the number of even intervals after his move!

At this point BH expressed some frustration with the discussion so far. He noted that we had worked on this problem for two days, without finishing an analysis of even one case. DEK observed that with what had been discovered so far, we could complete an analysis for any of the tie breaking strategies in the discrete case with a simple computer program.

DEK then directed the discussion to the analysis of the two person case for as many of the tie breaking strategies as possible. RZ corrected the solution for the two person case given in the problem statement, saying that the two numbers chosen should be 50 and 51 rather than 49 and 50. DEK wondered if this solution would change under any of the tie breaking strategies, and TF contributed the fact that 52 would be a valid choice for the second player if the first player chose 50 and the choose highest strategy was adopted.

Somebody proposed that tie breaking strategy 9 was equivalent to 2, assuming that the mother chooses a truly random number, so strategy 9 could be discarded. It was also shown that the first two strategies were equivalent.

An analysis of the three person case was then started. We considered the payoffs in terms of the interval sizes. The numbers (n_1, n_2, n_3) represent the numbers chosen sorted by numerical sequence; we used the case of $(10, 60, 65)$ as an example. The numbers (a_1, a_2, a_3, a_4) represent the intervals, calculated by successive differences, assuming the sequence of n's was prefixed with a 0 and suffixed with a 101; this gave us the intervals $(10, 50, 5, 36)$. The payoff was calculated by hand; n_1 wins in 35 cases, n_2 wins in 27 cases, and n_3 wins in 38 cases. A formula for the number of winning cases based on interval sizes was proposed:

$$p_1 = a_1 + \left\lceil \frac{a_2}{2} \right\rceil$$
$$p_2 = \left\lfloor \frac{a_2}{2} \right\rfloor + \left\lceil \frac{a_3}{2} \right\rceil$$
$$p_3 = \left\lceil \frac{a_3}{2} \right\rceil + a_4$$

This was found to fail for the case of p_2, however, and TH was first with a solution; rather than use the difference between the two bounds as the interval size, use the number of values between the two bounds (which is one less.)

The continuous case was examined next. First, we looked at the third play, assuming that the first two players had played at n_1 and n_2. There are three gaps, a_1 and a_3 on the two ends, and a_2 in the middle. If the third player moves in interval a_2, his expected payoff is $\frac{a_2}{2}$, no matter where he plays. If he moves in interval a_1, choosing some $b < n_1$, his payoff is $b + \frac{a_1 - b}{2}$ or $\frac{a_1+b}{2}$. To maximize this, b is maximized, yielding a payoff of $a_1 - \epsilon$. Therefore, his best move is trivially found.

A handy model based on an infinite number line was presented next. Imagine that when you chose x, all points $2n + x$ and $2n - x$ are marked on the real line, for all integers n. This 'reflective' model may be useful for visualization.

The second move was examined next. We can assume the first player played at some point, say, 0.2. If the second player plays in the lower interval, the third will play at $0.2 + \epsilon$. If the second player plays close to the first, at b, let us say, the third will play at $b + \epsilon$. If the second player plays far from the first player, the third player will play between the two. For a first play of 0.2, we found that the boundary between the latter two cases to be when the second player plays at 11/15.

At this point we were almost out of time, so we talked about how many operations it would take for a computer program to analyze the discrete case for three players. We can calculate $p_k(x_1, x_2, x_3)$ in constant time. Calculating $x_3 = f(x_1, x_2)$ will take about 100 iterations. The next level, $x_2 = f(x_1)$, will take approximately 100 iterations of a calculation for x_3, and x_1 will take approximately 100 iterations over x_2. This yields about 100^3 operations. MH suggested that this can be reduced by a factor of 100, since we have a simple algorithm to calculate x_3. He also suggested another way to reduce the complexity; scale the problem down, say from 100 numbers to 50, solve it there, and refine the solution somehow for the larger case.

Postamble

For class on Tuesday, DEK wants a list of the members of each team, and solutions for as many combinations of the three person game as possible.

13 January 1987

More on Toetjes

ER started the third class day with a report on his solutions to the integral case. With selection in the range of 1 to 100, the three person game yielded the optimal solution of choices 75, 25, and then 26. The lowest optimal choice was taken at each point. The solution required two minutes on a Macintosh. He also ran a four person game, with selection restricted to 1 to 25. After an hour on the Mac, the solution 21, 13, 5, and 4 was found.

JS said that an optimal strategy might be for the first $n-1$ players to spread themselves out-evenly, in positions $(k - 1/2)/(n - 1)$ for $1 \leq k < n$, except that some players may try to shrink an interval by a small amount to reduce the chances of the last player moving next to them. This seemed to agree with ER's results.

This brought up the subject of epsilons. DEK stated that ϵ is positive and smaller than any positive real. DM said that, because of this, ϵ would not factor into the payoff. DEK replied with the fact that the epsilons would factor into the payoff when the continuous case mapped into the discrete case. BH noted that the mapping might fail, because the discrete case introduced edge cases and funny possibilities.

Discussion then proceeded to the form the epsilons would take. DEK introduced polynomials in epsilon, noting that $k\epsilon^2 < \epsilon$ for all real k. After some initial discussion over the order in which the exponents would be chosen, JS suggested that the exponents were arbitrary, and could be assigned simply by each player choosing his exponent by comparing it to the previous players; a simple ordering should result.

MR pointed out that if ϵ was smaller than any real number and didn't really affect the payoff, ϵ was not big enough to make any intervals smaller and thus affect subsequent plays.

DEK suggested that we analyze the four player game in the continuous case. With a conjectured solution of 1/6, 1/2, and 5/6 for the first three players, the last player can move anywhere from 1/6 to 5/6 with equal payoff.

BH suggested that we modify the game somewhat; we play it on a circle, and see what happens with this simplified game. The first player chooses some point, and the game is composed entirely of bounded intervals. JS agreed that solving this case might shed some light on the ϵ problem. Noting the fact that the last player will play in the largest remaining interval, JS mentioned that the people who play on the ends could choose points ϵ closer to the ends and effectively reduce the game to the circle game. For instance, for the five player game, the first player might choose $1/8 - \epsilon$. This protects a region of 1/8 to his left, since nobody else will want to move there. The second player might choose $7/8 + \epsilon$, for the same reason. Now we have effectively a circle game with the remaining players. The next two players would presumably choose $3/8 + O(\epsilon)$ and $5/8 + O(\epsilon)$ in some fashion.

In reply to BH's conjecture that this might be a problem in which we never use a computer, DEK agreed that computers are often applied too quickly, but he also started discussion on how we might use a. computer in this situation. An accurate solution of the continuous case might allow an optima.1 solution to be found in a discrete case more quickly, perhaps through something similar to alpha-beta pruning. Alpha-bet a pruning does not apply directly in this case, since it is not a. two-player game. DS agreed that something like alpha-beta pruning ought to help, because other players who maximize their scores are implicitly minimizing yours. One way to do some pruning will probably be to analyze moves in order of their probable optimality. It is unnecessary to explore the alternative of moving into a small interval whose payoff is guaranteed to be less than the payoff known to be achievable by an already-analyzed strategy. Also, if you find that some move gives a previous player a poorer score than he knows he can obtain, you may be able to avoid exploring that move because the previous player would not have allowed you to make it.

DEK brought out a potential problem with the discrete case where ties are broken by replaying the game; in the case of a. tie, the chance of winning is based on the chance of winning, recursively. Thus, calculating the optimal move for a player may depend on his optimal move. It's not just a. simple recursive function where you can iterate, either; the choice of moves may profoundly affect subsequent play.

TF then presented a full analysis of the second to the last player's choices in the continuous case, assuming that optimal moves are chosen either randomly or at the middle of intervals. He started by defining the 'length' of an interval as the distance between the two bounding points divided by 2, if it is bounded on both sides, or simply the actual length of the interval for the end intervals. **He** then broke the situation into four cases, depending on the relative lengths of the largest and second-largest intervals, and showed that in each case the second to the last player will move into the longest interval.

Following this, JS presented his analysis, based on 'terminal' and 'non-terminal' intervals. Let's pretend the game is over. We define the following terms: A terminal player is a player whose neighbors on either side played before he did. At any point in the game, a terminal interval is an interval that will not be moved in by the end of the game.

DEK pointed out that the sequence of intervals and moves can be thought of as a tree: The first player's move goes at the root, and the other moves go into the left or right subtree if they are less than or greater than the first move, respectively. Then terminal intervals correspond to the leaves of the tree.

JS continued: Let us assume that there are k terminal players. Let's ignore these players, and look at the moves of the $n - k$ non-terminal players, marking each interval as a terminal or a non-terminal interval. One of the non-terminal intervals is the largest; this is the one the last player will play in. The length of this interval shall be called s.

Let us now look at some other non-terminal interval. If the length of the interval is less than s, then at least one of the two players bounding the interval is not playing optimally. For if they had moved to make the interval slightly larger but still smaller than s, they would win more.

On the other hand, the interval cannot be larger than $2s$, as otherwise after some player plays in the middle, one of the two new intervals would be larger than s, and the last player would move there. Thus, each non-terminal interval will be between s and 2s in length. Each player will play to optimize the lengths of the intervals on either side of him, so it appears that all of the non-terminal intervals will close to length $2s$, and all of the terminal intervals will be of length s.

DEK ended the meeting by commenting on how promising some of the ideas looked, and everyone went home to study for the comps.

15 January 1987

Just Desserts

Class started with a festive *Happy Birthday* sung by the class for BH. DEK reminded everyone that the Trivia Hunt would start Tuesday. He recommended that the teams be kept to a uniform size.

KR started discussion on Toetjes by asking if we really needed epsilons. DEK mentioned that the epsilon system presented last time had flaws when combined with the operation of maximization. For example, the following function suggested by BH has no maximum:

$$f(x) = \begin{cases} 2x, & 0 \le x < 1/2 \\ x - 1/2, & 1/2 \le x \le 1 \end{cases}$$

We might say that the maximum value $1 - 2\epsilon$ occurs when x = $1/2 - \epsilon$, but some sort of sequential scheme for choosing epsilons seems indicated.

KR continued by presenting his solution to the continuous case for three players in the modified game that has virtual players at 0 and 1. If a plays at $1/3 - \epsilon$, and b plays at $2/3 - \epsilon/2$, then c should play anywhere between a and 1. Assuming a random choice, the payoffs for a, b, and c are $7/24$, $1/4$, and $1/6$ respectively. But if a plays at $1/3$ exactly, b plays at $2/3$ and c moves anywhere randomly, the expected payoffs are $5/18$, $5/18$, and $1/60$. So a increases his payoff if he uses epsilons in his move.

MR wondered what would happen if b moved to the right a little, playing at the point $2/3 - \epsilon/2 + \gamma$. This would force c to move between a and b. The new payoffs calculated for this situation were $1/4$, $1/4$, and $1/6$.

We decided to look more closely at the game with the zero and one points already taken, so that we could understand the calculations underlying KR's solution. For the two person case, the first person might play at $1/2 - \epsilon$, and the second player somewhere to the right. With this case, the payoffs would be $S_0 = 1/4 - \epsilon/2$, $S_a = 3/8 - \epsilon/4$, $S_b = 1/4 + \epsilon/2$, and $S_1 = 1/8 + \epsilon/40$. With the first player at $1/2$, the payoffs are $1/4$, $3/8$, $1/4$, and $1/8$. Thus, it is to the advantage of the first player to move exactly in the middle.

Infinitesimals were brought up next. BH mentioned sequential choice of infinitesimals seems like John Conway's notion of 'o-games' in the book Winning *Ways*. When someone asked what infinitesimals were, DEK described the progression from the rationals to the reals to infinitesimals, and referred the interested student to Surreal *Numbers*.

Reference was made back to KR's presentation, where it was noticed that the first player decreased his winning by $(7/24 - 5/18) = 1/72$ by moving exactly on $1/3$. BH noted that in the discrete case, such a small difference might drop out.

DEK asked how feasible it was to do these case analyses on the computer, since "computers are good at cases." Rational arithmetic is an obvious necessity; floating point will not work. MH said that infinitesimals were also required, and BH conjectured that only one would be required. DEK suggested an interesting possibility; a symbolic representation might be used for the numbers. Then, every time a calculation required the comparison of two numbers, all possibilities be tried. Thus, the search procedure searches a tree of constraints on the values.

JS said that the epsilons should be reported in the score. DEK agreed; taking the limit as epsilon approaches zero is not sufficient. This is reflected in our discussion of γ above; if b's payfoff is just known to be $1/4$ in the limit, we miss the fact that he is better of taking $\gamma = 0$.

Our analysis of the 2-person case explains the moves of b in KR's analysis of the 3-person case, since the 3-person game reduces to a 2-person game scaled by a factor of $2/3 + \epsilon$ after player 2 chooses $1/3 - \epsilon$. That is why b moves exactly in the middle of the remaining interval.

This suggests that a recursive approach might work, where the n-player game can be solved by looking at scaled-down versions of a (previously solved) $(n - 1)$-player game. If we know the expected payoffs of S_0, S_a, S_b, S_c, S_1 in the 3-player game, including the dependencies on ϵ, we can analyze what happens in the 4-player game when the first player guesses $1/4 - \epsilon$. MH pointed out that this would lead to quadratics in ϵ and γ; somebody else mentioned that ϵ and γ are independent. so that we can not tell whether 3ϵ is greater than or less than 2γ, for example.

JS then suggested that only one epsilon move need actually be made in a game. For instance, after the first player in this 4-person game moves to $1/4-\epsilon$, the remaining interval is already stretched, hence the second player can move to $2/4 - \epsilon$ instead of $2/4 - \epsilon - \gamma$. DEK suggested thta we might be able to let the computer prove this conjecture by showing that $\epsilon = 0$ turns out to be optimum.

RC mentioned that we had not proved that, assuming the first two players played near the endpoints, all subsequent plays would be between them. We had not even shown that it was optimal for the first two players to play near the endpoints. DM said that we could prove that it is dumb to play at $1/4 + \epsilon$ for the three person continuous game, since the last person will move at $1/4$. BH mentioned that if the first player plays in the vicinity of his lowest optimal point, our analysis holds, but he might have another equally good move somewhere in the middle. DM countered by saying that you wanted to prevent the last player from moving next to you, so you wanted to move next to a previous play to control the size of one of your intervals.

DEK then proposed that we look at a possibly recursive analysis. If there are n players, and the first player plays at a, then the subsequent plays can be looked at as a i-player game on the interval (0, a) and a j-player game on the interval (a, 1), where $i + j = n - 1$. For instance, the first player might move at 0.4. Let us say that there are three remaining players. If the second player decides to treat the right-hand interval as a 2-person game, he will move to its midpoint, 0.7; then the third player will play somewhere in the left-hand interval. However, if the second player plays at thinks of the right-hand interval as a 3-person game he will play at $0.6 - \epsilon$, and the third player will play somewhere between the second player and 1.

BH thought that, if there were nine people left to play, one might be able to figure out how many would go on each side by the respective lengths of the two sides and by the expected payoff for the smaller games.

However, TR noticed that a simple recursive analysis might not hold. If, for instance, a plays at 0.46, b's best move is at 0.23, which matches the recursive strategy. At this point, however, c can play at $0.69 - \epsilon$, after which d will play somewhere in the interval $0.77 + \epsilon, 0.92 - 2\epsilon$, thus not creating only intervals smaller than 0.23, which will force e to move in one of the first two intervals. With this analysis, c's payoff (assuming d plays in the middle of his interval or chooses randomly) is 0.1925.

If c plays according to the three-person strategy, he would play at 0.64, d would play at 0.72, and e would play as before, yielding c a payoff of 0.18. Thus, the size of the other intervals affects the play within a single interval.

At this point, the bell rang. The first set of writeups are due Tuesday; one writeup per team.

20 January 1987

From Toetjes To Trivia

Sape Mullender appeared in Tuesday's class and bestowed upon us the correct pronunciation of the word 'toetjes'. Finally we could talk about our problem with some semblence of intelligence. Reports were due, and the day's class was spent discussing the results.

MR., representing his team of CC, DB, and himself, quickly presented some results. They used a. brute-force approach, generating all possible combinations of moves before the last; the last player's move was decided by moving into the largest remaining interval. The lowest optimal move was chosen in the case of equally good moves, as this was the easiest to implement. DEK asked how hard it would be to modify the program to calculate the solutions for all of the tie-breaking strategies and move selection strategies, perhaps all in parallel. MR replied that it would be easy to modify the code to handle each case individually, but that it was not written to handle cases in parallel. His team's results for the five person game with 19 and 20 numbers are, respectively,

$$(3,4,3) \quad (4,5,2.5) \quad (8,3,4) \quad (12,0,4.5) \quad (17,0,5)$$
$$(3,4,3) \quad (4,5,2.5) \quad (8,3,4.5) \quad (13,1,5) \quad (18,2,5)$$

Each triple (i, j, k) represents player j moving at location i with payoff k. This solution was checked against a proposed continuous solution of the first $n-1$ players dividing up the numbers into equal intervals, with the end intervals being half the size of the others, and it fit nicely. But we noticed a discontinuity between the cases 19 and 20.

MH then represented his team (ER, AW, and himself) with their solution. He started with the assertion that the original pruning argument presented in the previous class was flawed. Let us say that player i, in an early branch of the search space, received payoff a. If player j considers a move which would limit i to a payoff of less than a, that move can be pruned, according to the original argument. By pruning here, however, we cannot prove that player i can receive payoff a. In other words, the pruning destroys the search for the optimal solution.

MH then presented his solution to this problem; prune only at the level you are searching. In other words, if you are considering a move for player i, consider all moves which have a potential payoff greater than the best seen so far, but prune all those with a priori less payoff. With this approach, the team solved the five-player case with 25 numbers in approximately ten minutes of CPU time. The solution they presented was:

$$(3,5,3) \quad (4,4,3.5) \quad (10,2,6) \quad (16,3,6) \quad (22,1,6.5)$$

Once again, the tie-breaking strategy was to choose the least equivalent move. The pruning at the various levels was, from highest, to lowest, 0, 42, 2,637, 18,743, and 1,305,072. Each of the 42 prunes at level 1 accounted for a savings of approximately $42(25)^3$, so there was significant' pruning indeed.

MH continued by mentioning how they wanted to add in a simple heuristic to generate a more reasonable search order, increasing the pruning, but they spent so much time debugging the ideas presented in the previous class that they did not get around to it.

TH then presented the results of his team (including KR and HJ) in his quiet-spoken way but amazed his classmates with the news that they had calculated the solution for five players with 100 numbers in ten minutes. This was accomplished by selecting the moves for the last two players algorithmically. The lowest among equally good moves was chosen, except for the last player, who played in the middle of the lowest equally-sized largest intervals. Two optimal games were:

$$(13,1,25) \quad (38,2,25) \quad (63,3,19) \quad (76,5,13) \quad (89,4,18)$$
$$(12,4,18) \quad (25,5,13) \quad (38,3,19) \quad (63,1,25) \quad (8872725)$$

Once again, these results reflected the general solution for the continuous case of dividing up the line among the first $n-1$ players.

TF then presented solutions for his team (with RC and RZ). They concentrated on the continuous case, and found an algorithmic solution. Because of time constraints, he only presented the solution with virtual players already at 0 and 1.

First, he argued that the order of the remaining intervals does not matter for subsequent play. Let us say that there are k intervals, and their lengths are a_i for $1 \leq i \leq k$. The solution rests on choosing the maximum B such that

$$\sum_i \left\lfloor \frac{a_i}{B} \right\rfloor \geq k$$

Interval i shall have c_i subsequent moves played into it, so

$$\sum c_i = k$$

Now, B is the size of the interval for the last player. so. dividing the remaining k intervals into intervals smaller than B means that

$$\sum_i \left(\left\lceil \frac{a_i}{B} \right\rceil -1 \right) < k$$

Combining these two, the team found that for some interval i, a_i/B must be an integer. If this happens for more than one value of i, we can imagine altering some intervals by ϵ so that we can take $c_i = \lfloor a_i/B \rfloor$.

Continuing, he noted that the first $c_i - 1$ players in interval i can play at jB, where j is an integer denoting their order of play. The last player in each interval will then divide the remaining interval in half.

At this point, we were almost out of time, so TF's excellent presentation was cut short. TH mentioned that his team had also solved the continuous case, and had found that the first $n-1$ players indeed divide the line into equal intervals. However, they had to confine attention to a particular tie-breaking strategy.

An interesting result was noted because no one wants the last player to play next to them. Through analysis of the possibilities, TH's team calculated that the only player that had to make an epsilon move was approximately the ($\lg n$)th player from the end, because the previous players were subdividing the line by choosing the largest interval in which to move, and moving as close to the center as possible while still moving at $(i - 1/2)/(n - 1)$ for some integer i.

BH mentioned that epsilon could be another dimension of sorts; rather than including it in the numerical calculations, certain intervals could be marked as being shorter than others despite the fact that their lengths might be numerically equivalent. TF noted that his solution did not require epsilons, and at this point the bell sounded.

The Trivia Hunt questions were passed out, and few were unimpressed with the questions. Students without teams were rapidly assembled into teams, and the class broke up and started hunting. (See Appendix A.)

28 January 1987

Back to the Notes

The two class periods following Martin Luther King Day were used by DEK to explain and illustrate the handouts on multigrades and the L^3 short vector algorithm. (See Appendices B and C.) This handout marks the return of the class notes. They say sequels are never as good as the original; I plan to disprove that.

The Attack of the Killer Toetjes

TF dropped by my office a few days ago with an interesting example I thought I would share with the class. I wish I could share credit.

It is intuitively and aesthetically appealing to say that, for the continuous n player game with endpoints taken, the first $n - 1$ players will play at i/n for $1 \leq i < n$, with perhaps some epsilons to force the last player somewhere. But this turns out not to be the case, at least for the seven player game.

The first player can expect a payoff of at most $1/n$ (plus some small epsilon, perhaps) with the above strategy; this is $1/7$ for the seven player game. We will show that if the first player plays exactly in the middle, his payoff will be at least $i/48$ (minus some small epsilon, perhaps), which is greater.

After the first player plays at exactly $1/2$, the remaining six players will distribute themselves among the two remaining sides. We will look at the case for three players playing on each side; the other cases give greater payoffs, but will not be mentioned here.

On each side, the three remaining players will play an optimal three player game. An optimal strategy for this game seems to be (for the unit interval) for the first player to play at $1/3 - \epsilon$, the second at $2/3 - \epsilon/2$, and the third anywhere in the right-most two intervals. The other optimal strategy is the mirror image. With this game, the payoffs are (without epsilons) $S_0 = 1/6$, $S_a = 7/24$, $S_b = 1/4$, $S_c = 1/6$, and $S_1 = 1/8$.

The payoff for the first player of the seven player game, therefore, with the assumption that equivalent moves are chosen at random, is equal to $(S_0 + S_1)/2 = 7/48$. This is greater than the $1/7$ calculated above.

Multigrades On The Bounty

DEK opened Thursday's class by pointing out an error in writeup on the L^3 algorithm, and then yielded the floor to class discussion.

TF was the first to speak up. His group (with TH, MR, and CS) had been working on multigrades in modular arithmetic. For any old prime of the form $2k + 3$ (i.e., every odd prime), a unique ideal multigrade modulo p of order k can be found. For integer k where $2k + 3$ is not a prime, no such multigrade can be found. In addition, ideal multigrades modulo any number of the form p^α for integer α can be found for this p.

MR explained the the choice of the number $2k + 3$. An ideal multigrade of order k has $2k + 2$ numbers. Since a constant can be added to each term of a multigrade and still yield a multigrade, the number 0 could be eliminated with no loss of generality if the modulus were $2k + 3$. This allowed them to choose half of the numbers from 1 to $2k + 3$ for the left side, put the others on the right side, and simply check for a multigrade.

TF then explained how he proved that the construction works for all p. He noted that you can multiply a multigrade by a constant and still have a multigrade, even in modular arithmetic. The first $p - 1$ powers of a primitive root r of a prime p, modulo that prime, will generate all of the integers from 1 to $p - 1$. Thus, if you have a unique multigrade consisting of $a_1, \ldots, a_s \overset{k}{=} b_1, \ldots, b_s$, $s = k + 1$, where all of the a's and b's are different, multiplying it by r modulo p should leave the integers from 1 to $p-1$ divided into the same two sets. The only way this is possible is for one set to be the even powers of r modulo p, and the other set to be the odd powers of r modulo p. Multiplying the multigrade by r simply reverses the two sets.

If r is a primitive root, $r^{(p-1)/2} \equiv p - 1$. This can be shown easily since its square (modulo p) is unity. The only two numbers which are square roots of unity modulo p are 1 and $p - 1$, and 1 is already generated by r^0.

We can further infer that $r^{x(p-1)/2}$ mod p is 1 when x is even, and $p-1$ when x is odd. This can be rewritten as $i^{(p-1)/2}$ when $i = r^x$. Thus, for any integer i such that $1 \leq i < p$, i is an odd power of r modulo p when $i^{(p-1)/2} \equiv p - 1$, and an even power of r modulo p when $i^{(p-1)/2} \equiv 1$. Thus, one set is the set of roots to the equation $x^{(p-1)/2} - 1$, and the other is the set of roots to the equation $x^{(p-1)/2} + 1$. Since these polynomials of degree $k + 1$ differ only in their constant term, their roots define ideal multigrades of order k.

Using these two equivalent methods, we quickly generated some more modular multigrades:

$$1,4,5,9,3 \overset{4}{=} 2,8,10,7,6 \pmod{11}$$

$$1,4,3,12,9,10 \overset{5}{=} 2,8,6,11,5,7 \pmod{13}$$

TF then mentioned how his team generated solutions modulo a power of p. They took a. multigrade for a lower power of p, and added multiples of that lower power of p to each of the terms.

DEK mentioned how these modular multigrades resemble p-adic numbers. A p-adic number is analagous to a floating point number in some respects, except that expansions are to the left of the decimal point instead of to the right, and all arithmetic is performed modulo p^n, where p is the base and n is the number of digits. Thus, instead of knowing a real number to within 2^{-n}, you know a 2-adic number modulo 2". These p-aclic multigrades TF and his team found may or may not correspond to integer multigrades, as the p-adic numbers might be similar to repeating decimals.

At this point discussion was shifted to the L^3 algorithm by BH, who said that Lenstra, Lovász, and Lenstra's complexity analysis showed it to run rather slowly. The core of the algorithm is basically an exchange sort; could it be sped up by using something more analogous to, say, a. Shell sort? DEK responded that he does not think any such method is known. However, although the complexity analysis shows it to perform badly in the worst case, actual execution is fairly fast. For some problems, the algorithm was running in essentially linear time. He mentioned how a lot of problems which were thought very difficult or even impossible to solve because of worst-case analyses actually yielded a solution in practical cases. BH supported this assertion, mentioning that Karmarkar's algorithm, which is faster than the simplex method for linear programming in theory, actually does not perform as well. DEK said that we don't know that for sure yet.

DB then presented some results from his experimentation with the L^3 algorithm, starting with the polynomial $(x-1)^6$:

$$0,3,5,11,13,16 \overset{5}{=} 1,1,8,8,15,15$$

Subtracting the constant 8 from all terms yields the symmetric result

$$-8,-5,-3,3,5,8 \overset{5}{=} -7,-7,0,0,7,7$$

This is nicer than any previously known multigrades of order 5, since it has a. smaller spread between largest and smallest terms. Thus the L^3 algorithm gains a victory already!

DEK inquired as to the stopping criterion, and DB mentioned that it was the same as in the original handout. Somebody pointed out that the length of this vector is $\sqrt{18}$, not $\sqrt{12}$, although we do have an ideal multigrade. Thus the algorithm did not really find the shortest vector, but it found one we like. DEK mentioned the possibility of changing the stopping criterion: We could terminate when the sum of absolute values equals $2k+2$. He reviewed some simple facts about norms. The so-called l_p norm of a vector can be defined as

$$\|x\| = (\sum |x_i^p|)^{\frac{1}{p}}$$

where p is chosen to be between 1 and ∞. Choosing $p = \infty$ gives you the maximum norm, choosing $p = 2$ gives you the Euclidean norm, and choosing $p = 1$ gives you the sum of the absolute values. All norms satisfy the triangle inequality.

MW then mentioned that permuting the original vectors had a. substantial impact on the running time and output of the program. DEK mentioned that this was an opportunity to exploit parallelism; multiple processors or computers could work on the problem with various initial permutations, and when one finds a solution, it can notify the others. This reminded him of the classic solution to simple alphabetic ciphers; simply get 26! people, give each a permutation of the alphabet, and show them a.11 an encrypted message. One of them will understand it and raise his hand.

The bell rang, and class was dismissed.

3 February 1987

More Musings on Multigrades

DEK uncovered some more problems with the original L^3 handout over the weekend as he implemented and tested the algorithm. The μs form a lower-triangular matrix of the following form:

$$\begin{pmatrix} 1 & 0 & 0 & \ldots \\ \mu_{21} & 1 & 0 & \ldots \\ \mu_{31} & \mu_{32} & 1 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

If the μs are adjusted from left to right by subtracting multiples of previous rows, μs earlier in t-he row get disturbed. The proper way to adjust them is right to left.

Reports were due today, so each group discussed their results. TH started with what his group had done since their presentation the previous class day. Every ideal multigrade of order k must be of the following form:

$$p(x) + (x^p - 1)q(x)$$

where $p(x)$ is the unique ideal multigrade of order k modulo p as a polynomial in x and $q(x)$ is some other polynomial. It can also be written as

$$(x - 1)^k r(x)$$

where r(x) is another polynomial. They tried using Euclid's algorithm to determine r(x) and q(x), but had no results at that time.

DB then talked about his group's results, which included a new ideal multigrade of order 5, with maximum element 22 and no repeated values:

$$22, 17, 16, 6, 5, 0 \overset{5}{=} 21, 20, 12, 10, 2, 1$$

This was reduced to a symmetric ideal multigrade by subtracting 11 from each term:

$$11, 6, 5, -5, -6, -11 \overset{5}{=} 10, 9, 1, -1, -9, -10$$

DEK then talked about his experimentation with the problem. For an order 6 multigrade, working with polynomials of degree up to 102, and using initial values for d of $\{3,5,7,11,13,17,19\}$, his program found the same solution as the one in the book after trying several permutations of input vectors. Limiting the polynomial degree to 100 and using initial values for d of $\{2,3,5,7,11,13,17\}$, his program found another solution very quickly. The solution found was

$$0,18,19,50,56,79,81 \overset{6}{=} 1,11,30,39,68,70,84$$

as this seems better than previously known solutions of order 6.

KR mentioned that he also worked on a solution for the order 6 multigrade, but did not quite get one; he multiplied one of his close guesses by $(x-1)$ and this led to an ideal multigrade of order 7. BH said that the search space was decreased more significantly for the larger primes one chose for d; terms like $(x-1)$ do not help speed up the program as much.

MW mentioned that his team had doubled the size of the polynomials with their program, and it yielded the same multigrade as before but it look a lot longer; it seemed exponential to him. DEK noted how quadratic running time can seem exponential, and that more statistics might want to be taken. Such data had been taken by BH, who said that the variance in running time from the different input permutations was so incredible that the mean was almost meaningless. DEK said that a reasonable estimate of running time (to within an order of magnitude) was often possible despite this large variance.

DEK then pointed out another mistake in his first thoughts on the L^3 algorithm; if one set the termination condition to $\|b^*_{i+1}\|^2 \geq \|b^*_i\|^2$, the program is guaranteed to go immediately into an infinite loop. Oops.

DB then presented a new multigrade of order 5:

$$0,7,7,21,21,28 \overset{5}{=} 1,3,12,16,25,27$$

MW mentioned that they had found lots of multigrades of order 5 in addition to this one. They considered using data structures designed for sparse vectors rather than the straight linear arrays, but their investigations of the A matrix seemed to indicate that it was not that sparse. DEK agreed, saying that only the final vector is usually sparse; almost all of the intermediate vectors are fairly dense. BH mentioned that at four bytes per number, a 500 by 500 matrix still fit in less than a megabyte, and this is not a lot of memory on current machines.

KR said that his team had thought of a new alternative to try when the L^3 algorithm did not find a short vector. Rather than shuffling the original input vectors around and running the algorithm again, simply shuffle the current list of vectors and feed that as input. Another possibility is to use the known multigrades as input. Using these techniques, his team rediscovered the order 7 multigrade and found several new order 5 multigrades.

MW then presented his team's results for multigrades of order 8 or larger. The smallest multigrade they could find of orders 8, 9, and 10 had 14, 20, and 26 terms, respectively. They started their L^3 algorithm with simple binomial coefficients.

BH than casually stated his results; multigrades of order 5 were simply 'dropping out of the sky,' and multigrades of order 6 and 7 were found with some hints to their program.

DEK then mentioned the only previous computer work he knew of on this problem was done in 1965, simply used the cancellation technique presented in the multigrade handout, and did not yield particularly good results. This led MR to ask how long a researcher might work on a problem like this. It was a good question, according to DEK, who then talked about how he had spent the entire previous day trying to construct a sentence of the form 'This sentence contains forty-two a's, thirty-three b's, . . ., and one period.' He tried an exhaustive search technique which had not proved to be efficient enough to yield a solution in reasonable time. According to DEK, researchers usually keep a queue of unsolved or partially solved research problems, and continue working on them subconsciously.

He mentioned how when working on a thesis, you spend a lot of time getting familiar with a subject area, so thinking can proceed in larger steps. Then you can find yourself in a. position where you know more about a certain narrow area than anyone else, leading to a feeling that you must solve certain problems in that field, since you can do it more easily than anyone else.

BH said that his group implemented the L^3 algorithm on a LISP machine, and started it running on $(x-1)^{11}$ with up to 500 vectors; the last time he checked, it was killing itself in garbage collection. He noted how easy it was to prototype the program on the machine, though; 'bugs seemed to jump out at you.'

MR asked why anyone cared about the L^3 algorithm; what practical use could it have? DEK said that it had been used in theory to prove some results, and that it has been applied successfully in cryptography.

We then started looking at generating multigrades by the technique of putting constraints on undetermined coefficients. MR mentioned that they got very complicated very quickly. DEK started with

$$a, b, c, d, e \overset{0}{=} f, g, h, i, j$$

when MR complained, mentioning that they had started at the other end and worked backwards, and assume that a certain number of terms cancel at each stage. We decided to try it the way we started, and continued to the next step:

$$a, b, c, d, e, f+1, g+1, h+1, i+1, j+1 \overset{0}{=} a+1, b+1, c+1, d+1, e+1, f, g, h, i, j$$

Now we can pick terms to cancel, yielding the constraints

$$\begin{aligned} a &= b+1 \\ b &= c+1 \\ h &= f+1 \\ f &= g+1 \end{aligned}$$

KR said that we might use an ordering relation, such as

$$a \le b \le c \le d \le e$$

to further constrain the values. BH mentioned that if you simply choose variables to cancel, are you doing any better than just picking random numbers?

At this point we were nearly out of time, so DEK congratulated the class on its good results, even though no new ideal multigrades of a higher order were found. He suggested that everyone read and start working on problem 3, even though it might turn out to be unsolvable.

5 February 1987

CHEX: It's Not Just For Breakfast Anymore

CHEX (pronounced 'cheks', not to be confused with CHEX) is a small language that checks proofs strictly by checking syntax and types. Our goal is to write a program that determines if a given **CHEX** script is valid, and calculates the length in tokens of each expression when totally expanded.

CHEX deserves the distinction 'small' because of its 2 1/3 page user's manual given in the first handout. DEK claimed this problem may turn out to be easy to solve, but it should teach us something about logic and formula manipulation in the meantime. His so-called 'ultimate thesis advisor'-a mathematician by the name of deBruijn—had designed a similar language called Auto-Math to be used for proof checking. DEK rewrote this language, using a Pascal-like syntax, and thus was born **CHEX.** A **CHEX** script can be interpreted as a valid proof when it satisfies the typing rules of **CHEX.**

An implementation of a **CHEX** compiler might entail a recursive descent parser and some symbol table routines. The necessary data structures for checking types would be needed as well. For the problem we may want additional data structures to help determine the lengths of expansions.

We turned to studying the example script in some detail. First, the expression

$$bool := \#\colon atom.$$

was discussed. Here, *bool* is defined to be an external (which is equivalent to making it an axiom) of type *atom*. This *bool* can represent either true or false. At this point, MR complained that we had not explicitly stated that yet, and DEK agreed, saying we had to start out with something. We then moved on to the next statement:

$$proof\ (b : bool) := \#\colon atom.$$

Here *we* define *proof* to be an external function that takes a *bool* into an *atom*. We -continued with

$$eq(a : atom;\ x,\ y : a) := \#\colon bool.$$

Equality is defined as a function taking an atomic type and two values of that type, returning a *bool*. Finally, we encounter a definition:

$$is(a : atom;\ x,\ y : a) := proof\ (eq(a, x, y))\colon atom.$$

This is interpreted like a macro. After all of the parameter types are checked, the actual substitutions can be performed as in the handouts. The function *proof* returns an *atom,* so the return values match.

It was argued that this is unnecessary redundancy; that the return value of is, for instance, could be determined from the return value of *proof*. DEK noted that all proofs are tautologies and therefore redundant. In **CHEX**, the redundancy allows mistakes in the proof to be located and helps the user understand the proof.

MR then asked if a definition of *proof* as follows

$$is(a : atom;\ x, y : a) := proof\ (eq(a, x, y)) : bool.$$

would be legal, as *bool* is an *atom*. DEK said it would not; the types must match exactly. He considered generalizing the typing to allow subtypes to match a type, but realized how complicated the type checking could become. A **CHEX** compiler would then require a search for every type match.

In response to another question by MR, DEK showed that a definition like the following would not be valid:

$$f\ (b : bool;\ x : eq(bool, b, b)) := proof(x) : atom.$$

The reason is that ' *eq(bool , b, b)* is not an acceptable type, because its type is not *atom*. The mapping that takes an expression into its type must also take the type into *atom* when applied twice:

$$exp \rightarrow type \rightarrow atom$$

DEK then asked if there might be combinatorial explosion in the number of tokens in the final expansion. MAD mentioned that fully expanding each expression and counting tokens was probably not a good way to calculate the lengths, and DEK agreed. The numbers might well get unimaginably large.

For instance, Djikstra once said that it is very difficult to imagine even a number such as 1000, unless you are lying in a field and 1000 horses approach to trample on you. 10^{10} is an unfathomable number by these standards. Let us imagine we keep adding exponents onto exponents: $10 \uparrow 10 \uparrow 10 \uparrow 10$ might be represented as $10 \uparrow\uparrow 4$, for instance, and continuing this, we might represent $10 \uparrow\uparrow 10 \uparrow\uparrow 10 \uparrow\uparrow 10$ as $10 \uparrow\uparrow\uparrow 4$. Of course, this notation does not make it any easier to represent large arbitrary integers, but it can be used to give us an approximate order of magnitude, as it were.

It was suggested that we try and manufacture as complicated a first legal line as we could. After a few attempts, we came up with the following:

$$x(a : atom;\ b : (y : a)atom;\ c : a) := b(c) : atom.$$

At this point MR complained that he would rather work with lisp S-expressions rather than the syntax of **CHEX**, but of course this is a trivial transformation. DEK mentioned hovv, once the data structures are defined, the work on the project could be split up among the members of each team, and he asked that each team try to have a first pass ready by Tuesday.

The discussion then returned to calculating the length of expanded expressions. DEK asked if it was even necessary to remember the right hand side of a definition. It was realized that it might be necessary to keep the right hand side for type checking; expansion might be necessary to compare some formal parameters with the expected types.

DEK then clarified a shorthand that is used in **CHEX**. Let us say we have a function *p* which takes three arguments, but only one is given. This means we have a function of arity two. For instance, *eq(nat)* is equivalent to

$$(x, y : nat)eq(nat, x, y)$$

and $(b : bool)eq(atom, proof(b))$ is equivalent to

$$(b : bool; y : atom)eq(atom, proof(b), y)$$

which is a function of two arguments.

10 February 1987

Party CHEX

Trivia Hunt awards were presented at the beginning of class. Each award was typeset using the original POX typesetting system and was a special hand-crafted limited edition a-ward by DEK.

Class was opened by a comment by DEK on how a language designer must participate in the implementation of that language. Only then will the designer realize the ambiguities and problematic details in the language specification. For instance, with the original version of TEX, he specified a language which was implemented by a few of his graduate students over a period of a month or two. Then, after working on it himself for a few minutes, he understood the horrendous problems faced by those graduate students.

CHEX is one such new language, so DEK decided to implement a type checker himself. He wrote a program (using the WEB system of structured programming) which would expand out all of the expressions, and found some typographical errors in the original handout. First of all, in the expression

$$f(a : t) := (b : tt) : p(a)$$

one cannot refer to b in the $p(a)$ expression in contrast to what the original handout says.

There was also a minor problem with the description of filling in extra arguments in functions; the effects of previous arguments on the range was not mentioned. Let function f have arity n. Consider the following expression

$$(v_1 : t_1; \ldots . v_k : t_k)f(a_1, \ldots, a_m)$$

with $m < n$, and f with domain $(x_1 : X_1, \ldots, x_n : X_n)$. The above expression is a function of arity $k + n - m$. We can rewrite this as

$$(v_1 : t_1; \ldots; v_k : t_k; V_{k+1} : X'_{m+1})f(a_1, \ldots, a_m, V_{k+1})$$

where X'_{m+1} is X_{m+1} with any mention of $(x_1, \ldots, x_n)$ replaced by $(a_1, \ldots, a_n)$. Thus, for instance, if f has domain $(x_1 : bool; x_2 : proof(x_1))$, the expression

$$(v_1 : atom; v_2 : bool)f(v_2)$$

is equivalent to

$$(v_1 : atom; v_2 : bool; v_3 : proof(v_2))\, f(v_1, v_2)$$

BH mentioned how similar these **CHEX** scripts were to w-grammars, in their capacity for proofs; with w-grammars, the full power of logic is available from syntax alone.

DEK mentioned how infinite expanded forms introduce serious problems; if $f(x)$ expands to $f(f(x))$, type checking can become very difficult, and the expansion will not terminate. This is not possible in **CHEX**, however; no function can be used until its type is fully defined, including the types of all parameters. Thus, there can be no 'apply' function applicable to itself.

He then introduced the concept of rank. A function of rank 0 is a constant function with no arguments. A function of rank 1 can have multiple arguments, but the types of all arguments are of rank 0. Continuing, a function of rank 2 has arguments all of whose types are of rank 1 or rank 0. Because **CHEX** has strict typing, we know all of the types of the arguments, so we can calculate the rank of any function. In addition, expanding an expression of rank k one level is guaranteed to result in an expression of rank $k-1$. Therefore, infinite expansions should be impossible in **CHEX**.

Another interesting point was made by DEK when he pointed out the equivalence of the two definitions

$$f(v_1 : t_1) := (v_2 : t_2) : (w : t)$$

$$f := (v_1 : t_1; v_2 : t_2) : (v : t_1; w : t)$$

Thus, some simplification is possible by simply moving the arguments to the left of the := to the right of the := , and handling them there. BH and TH noticed how this might change the length of the expansion due to introduced or deleted parenthesis, for instance. DEK agreed that it would be good to define the problem more precisely.

EW asked what a token was defined as for this problem. For example, is := a token? DEK answered that it should be considered as two tokens; each character counts. He described how tokens could be represented on some systems, such as **METAFONT;** each character- has a certain class, and you simply scan forward collecting characters of the sanie class into a single token. But for this case, each character shall be considered a single token, except in identifiers.

A few more errors in the **CHEX** script at the back of the original notes were pointed out, a missing underscore and a few missing *nuts* were found. The missing *nuts* were in calls to the function is with only two parameters, both of type *nut;* DEK mentioned how this could be fixed with a definition of the form

$$is(x, y : nat) := proof\ (eq(nat, x, y)) : atom$$

thus redefining is to be of arity two, using the old definition. JS suggested improving this to

$$is(x, y : nat) := is(nat, x, y) : atom$$

and DEK agreed. He pointed out that the symbol table routines in a **CHEX** parser must be done correctly so that such a redefinition is possible.

DEK began to speculate on another example.

$$f(f : bool; f, f : proof\ (f))f$$

which was equivalent to

$$f(f_1 : bool; f_2, f_3 : proof(f_1))f_3$$

Someone pointed out that a strict reading of the rules would give rather

$$f(f_1 : bool; f_2 : proof\ (f_1); f_3 : proof\ (f_2))f_3$$

because the original formula is supposedly equivalent to

$$f(f : bool; f : proof\ (f); f : proof\ (f))f$$

DEK agreed, but said that he would rather change the wording of the rules.

MR then inquired about the concept of domains. He asked for the domains of *bool* and *eq*. The domain of *bool* is empty, because it is a constant, or of arity zero. The function *eq*, on the other hand, has domain $(a : atom;\ x, y : a)$ because those are the types of arguments it expects; the first must be an atomic type, and the last two must be of that type. But if *eq* is given three arguments, the domain of the resulting expression is empty.

Discussion then turned to type checking. TF pointed out that expansion might need to be performed before decided that two types were not the same. For example, suppose we have a parameter of type $f((r, s : nut)h(r, s))$ and a corresponding argument of type $f((r,s : nat)h(s,r))$. This is not necessarily a mismatch, if the function f has been defined to be, say,

$$f(x : (a,\ b : nat)nat) := x(0,0) : nut.$$

DEK said he had hoped it might be possible to put something in the data structure that would tell how far a function was from being fully expanded. Then if, for example, you want to match $f(X)$ with g(y), where f was three steps from fully expanded and g was five steps away, you would know to expand g twice before proceeding. But he was afraid such methods would only work in simple cases.

Sombebody suggested that this matching process looked a little bit like unification. DEK said maybe, but it did not seem as simple. He remarked incidentally that the comprehensive exam committee had recently found disagreement between textbook authors on the meaning of unification. For instance, when trying to match (or unify) the expressions $f(x,g(x))$ and $f(h(x),\ y)$, is the x in each expression the same x? If they are, then there is no unification ($h(x)$ can never be the same as x); otherwise, the unification is $f(h(X),g(h(X)))$.

RZ thought it might be possible to maintain a cache of common expressions and their expansions to speed things up.

We then started to consider calculating the lengths of expansions using something that might be called 'sharp' functions. Using the formula

$$is(a : atom;\ x, y : a) := proof\ (eq(a, x, y)).$$

we can say that every use of is expands to eight tokens (two identifiers, two commas, and two pair of parentheses) plus the lengths of *a*, x, and y. Symbolically,

$$is^{\#}(a :\ atom;\ x,y : a)\ :=\ 8 + a^{\#} + x^{\#} + y^{\#}.$$

The length of the expansion of any function of rank 1 can be written in a similar way.

BH said that allowing functions as arguments makes this more complex, but KR mentioned that he had already thought of that. A 'sharp' function can be created which is analogous to the original function, but calculates lengths, in terms of the lengths of its arguments, which might include recursive calls to the sharp function corresponding to a function argument. KR said that at least the first several lines of the first **CHEX** proof resulted in linear functions.

At the end of class, there was some scrambling to get the Trivia Hunt awards signed by DEK, and the students began to eagerly work on their **CHEX** programs.

12 Febrary 1987

CHEX Again

The CHEX script in the original notes was corrected and passed out in a new handout at the beginning of class. A second CHEX script was also prepared and handed out.

DEK opened discussion on computer proof checkers. Using a computer proof checker just changes the job to proving that the proof checker is indeed correct. Of course, a proof checker might be a useful tool in developing a proof, as it might catch some errors more easily than a human. In order to be able to place confidence in a proof checker, however, one has to be fairly confident that no incorrect proofs will be accepted.

Writing proofs is a notoriously difficult job. Let us take one classic problem called 'orthogonal latin squares'. Each element of an $n \times n$ matrix is a pair of integers, each between 0 and $n - 1$. The idea is to arrange all possible pairs of integers in that matrix such that when one takes the first number of each pair, every row and column is a permutation of the integers $0 \ldots n - 1$, and similarly for the second number of each pair. It was 'proved' that this was possible for all integer values of n except those that were congruent to 2 modulo 4, and those were impossible. Unfortunately for the proof, one of DEK's college math teachers found a correct matrix for $n = 10$, violating the proof.

Checking a CHEX compiler is not just a simple matter of running sample correct proofs and sample incorrect proofs, however. Large software programs such as compilers are often tested by finding a large application (like the Unix operating system for a C compiler) and making sure it works; typically, this only exercises half the code of the system, however. Our sample CHEX scripts do not exercise many of the weird things that are possible with the language. To really test a compiler, you need to get in a nasty frame of mind and try and trip up the compiler in any way you think you can.

MAD mentioned how Bell Laboratories likes to hire psychologists to help debug and test their applications; these psychologists are often very successful and finding problems completely overlooked by the programmers.

DEK described the testing of TEX, with a test called the trip test. All boundary conditions are tested, each line does things no sane user would ever dream of doing. DEK mentioned how he would be embarrassed to explain any of the lines in the trip test; they do things in such a convoluted and extraordinary fashion. After initial versions of the trip test were finished, they were run through a profiler that identified source lines never executed, and trip was extended to exercise those source lines. This continued until over 99% of the lines of TEX were executed at least once by trip, making it a very comprehensive test suite. Indeed, almost every single port of TEX to a new machine has uncovered at least one bug in the Pascal or C compiler used.

This method of testing was first used by DEK on a compiler in 1964; perhaps the most tedious part of such a test is determining, by hand, what the program should do on the test. Only two bugs were ever found in that 1964 compiler, one of which was actually present but overlooked in the output of the test routine.

Even with such thorough testing, however, bugs still pop up. After fourteen months with no bug reports for TEX (whose main compiler is approximately a. megabyte of source), a bug finally surfaced when DEK experimented with mixing right-to-left and left-to-right text.

No implementation of TEX can call itself TEX unless it meets two criteria: (1) It must pass the trip test, by writing almost exactly the same output when run over a certain file as DEK's master copy. (2) The implementer must be happy with how it works on his system.

To check a CHEX compiler, we ought to simply make minor changes to some correct proofs, and insure that it fails them. For a two week project, a full test suite is probably too much work.

DEK then went on and described his work for Burroughs as a hardware simulator. His task was to take hardware schematics that had already passed a hardware simulator, and find bugs in them. He said this fit his personality well; he loved to break code and designs. While he was with the company, he found between 100 and 200 bugs in their hardware that would have probably ended up in the final designs had he not caught them.

At the beginning of the class period, some students were poring over the following equations written on the board:

$$f(a : atom;\ x, y : a) := \# : atom$$
$$g(b : nut;\ p: (x : nat)atom) := p(b) : atom$$
$$h(x, y : nut) := g(x, f(nat, y)) : atom$$

MR's question was: what does the last line expand to? Expanding the right hand side gives

$$g(x, (t : nat) f(nat, y, t))$$

and this, in turn, expands to

$$f(nat, y, x).$$

An intermediate form in the last expansion, after argument substitution, is

$$(t : nat) f(nat, y, t)(x)$$

but CHEX has no syntax for a function before it is applied.

MR asked if variables must always be substituted from left to right, and DEK said that yes, they had to be, because of the type checking. This restriction made it difficult for him to introduce ordered pairs into the language. He could write a. function for the first element of an ordered pair, but did not succeed in determining one for the second element.

RZ mentioned that his team had implemented a CHEX compiler that seemed to work and was surprisingly fast on the examples he ran. MW had been working on the 'sharp' -functions, and said that these functions can complicate domains a lot. The bell rang, so class was dismissed.

The second CHEX script is as follows. (See Appenix D for a. commented form.)

```
bool:=#:atom.
proof(b:bool) :=#:atom.

for_all(t:atom;p:(x:t)bool):=#:bool.
generalize(t:atom;p:(x:t)bool;q:(x:t)proof(p(x))):=#:proof(for_all(t,p)).
```

```
specialize(t:atom;p:(x:t)bool;x:t;q:proof(for_all(t,p))):=#:proof(p(x)).

exists(t:atom;p:(x:t)bool):=#:bool.
existence(t:atom;p:(x:t)bool;x:t;q:proof(p(x))):=#:proof(exists(t,p)).
choose(t:atom;p:(x:t)bool;q:proof(exists(t,p))):=#:t.
thus(t:atom;p:(x:t)bool;q:proof(exists(t,p))):=#:proof(p(choose(t,p,q))).

trick(a,b:bool;p:proof(a)):=b:bool.

implies(a,b:bool):=for_all(proof(a),trick(a,b)):bool.
implication(a,b:bool;p:(q:proof(a))proof(b)):=
 generalize(proof(a),trick(a,b),p):proof(implies(a,b)).
modus_ponens(a,b:bool;q:proof(implies(a,b));p:proof(a)):=
 specialize(proof(a),trick(a,b),p,q):proof(b).

and(a,b:bool):=exists(proof(a),trick(a,b)):bool.
conjunction(a,b:bool;p:proof(a);q:proof(b)):=
 existence(proof(a),trick(a,b),p,q):proof(and(a,b)).
first_conjunct(a,b:bool;p:proof(and(a,b))):=
 choose(proof(a),trick(a,b),p):proof(a).
second_conjunct(a,b:bool;p:proof(and(a,b))):=
 thus(proof(a),trick(a,b),p):proof(b).

imp_refl(a:bool):=implication(a,a,(q:proof(a))q):proof(implies(a,a)).
imp_trans(a,b,c:bool;p:proof(implies(a,b));q:proof(implies(b,c))):=
 implication(a,c,(r:proof(a))modus_ponens(b,c,q,modus_ponens(a,b,p,r))):
 proof(implies(a,c)).
and_symm(a,b:bool;p:proof(and(a,b))):=
 conjunction(b,a,second_conjunct(a,b,p),first_conjunct(a,b,p)):
 proof(and(b,a)).

lemma1(a,b:bool):=implication(and(a,b),and(b,a),and_symm(a,b)):
 proof(implies(and(a,b),and(b,a))).
```

17 February 1987

Proof Checkers: Would You Like To Play A Game?

Class opened with BH initiating discussion when he made a comment that the hardest thing about this course is working in groups. Scheduling meetings among four busy graduate students is difficult, and splitting a problem into subproblems and delegating responsibility is a difficult problem. Especially with this CHEX problem, it is probably harder to specify an interface (e.g., between the parser and type checker) than it is to write the components themselves. DEK acknowledged these difficulties, saying the only person he could really work well with "online" is Ed Bender who is a professor of Math at San Diego. They seldom meet anymore because thky have an implicit commitment to do research when they do meet.

BH quoted Jeff Ullman that the optimal size of a committee is 0.6. DEK said that in practice the most reasonable organization is with a chief programmer delegating responsibility. He said that problem 4 does not split well into subproblems, so a small group size might be appropriate. On problem 5, a larger group might work well. DEK stressed that working together can be learned, and that even Edsgar Dijkstra could work in a team; the original Algol-60 compiler was written by Dijkstra and Zonneveld, taking turns calling out octal machine-language instructions. DEK said he even played a. piano duet with Dijkstra at one point; there were four hands, so Dijkstra led, but c'est la vie.

Nest DEK commented about the two new **CHEX** scripts he handed out. (See Appendices E, F.) **CHEX3** was a problem he originally solved in an extension to Auto-Math that used subtypes. (See Appendices E and F.) When he designed **CHEX** he wanted to use subtypes, which would have complicated the language but made solving problems like the one in **CHEX3** easier.

CHEX4 formalizes proofs by contradiction. It builds on **CHEX2,** and shows that $b \rightarrow \neg\neg b$ can be proved constructively. By modus tollens (the law of contraposition, if $a \rightarrow b$ then $\neg b \rightarrow \neg a$) we have a constructive proof that $\neg\neg\neg b \rightarrow \neg b$ as well. However, there is no way to prove constructively that $\neg\neg b \rightarrow b$; this is the famous law of the excluded middle, which states that you must be able to prove anything whose denial leads to a contradiction. Introducing one additional primitive called *non_constructive*, which states

$$\neg\forall x.\neg p(x) \rightarrow \exists x.p(x)$$

allows us to prove the law of the excluded middle. He said he was interested in performing a thorough check on these scripts, because they seemed more interesting than most elementary mathematics.

Then the groups presented their results. The first group (MW, KR, AW, and DS) performed length calculations with the associated sharp functions presented earlier, and showed that the expanded forms can grow with order $\Omega(2^{2^n})$ with n definitions. An example of this is

$$f_1(a,b) := \# : atom$$
$$f_2(a,b) := f_1(f_1(a,b), f_1(b,a)) : atom$$

$$f_n(a,b) := f_{n-1}(f_{n-1}(a,b), f_{n-1}(b,a)) : atom$$

DEK mentioned that this kind of function is a. classic example of functions that break optimizing compilers. (They lose the value of $f(a, b)$ while computing the second argument $f(b, a)$, if f gets its two arguments in registers.)

MW showed that the length of the nth definition is

$$2^{2^{n-1}-1}(a+b) + 4 \prod_{0 \le i \le n-2} (2^{2^i}+1)^2.$$

DEK pointed out that numbers of the form $2^{2^i} + 1$ ase called Fermat numbers. Fermat thought that all numbers of this form were prime, but actually only the first few are.

TF then pointed out that this product can be simplified and proved that it is equal to $(2^{2^{n-1}} - 1)$". The trick is to multiply by $(2-1)^2$ and then use successively $(x^n-1)(x^n+1) = (x^{2n}-1)$.

For $n = 5$, the expression sums to approximately 327,000. It was conjectured that functional arguments probably don't grow faster; a later argument by TF proved otherwise.

They continued their results, presenting some of the complexities they had to overcome. They had to implement lazy function evaluation (normal order evaluation in lambda calculus), which avoids evaluating functions that are never applied. They also found that the types of domains of the arguments must be kept around, which complicates the implementation.

They said that counting the lengths of these expressions takes exponential time, but not exponential space. BH supported this, saying that a DAG might be constructed with lengths of common expressions cached; he conjectured that the size of this DAG should not grow exponentially. MW said that might not help in the worst case. It was thought that the example function above might not contain common subexpressions, but DEK pointed out that the example does indeed contain many common subexpressions.

He then went on to elaborate on caching. In an environment where variable binding is constantly changing, it might be hard or expensive to detect common sub-expressions. Some analogies to the game of GO were made at this point, as GO-playing programs maintain a cache of pattern evaluations. DM mentioned that almost all chess programs use caching to remember previous chess positions; RZ brought up similarities to dependency-directed backtracking.

Some assumptions MW's team made were that the shorthand $x, y : a$ not be used, and that no functions be redefined. These are both trivial restrictions that can be removed with a simple preprocessing step.

The team of MD, TH, SS, MR, and DB then presented their results. Proof checking was implemented by TH, who had a high degree of confidence in its correctness; length checking was implemented by MD, who had a lower confidence level. It was found that they needed a full expansion routine for checking, so they also used this function as a brute-force length calculation routine. Because the lengths exceed 400,000 by the end of script 3 (according to RZ and TF), this routine was not sufficient.

The parser and scanner were written in **YACC** and *C++* and produced S-expression represent at ion of scripts. These were written by SS, who also developed a theory for counting functions, with sharp functions associated with each real function. SS illustrated these symbolic calculations on some special cases. He pointed out that the syntax of **CHEX** makes it possible to determine the length of a formula by knowing just the number of identifiers and the number of left parentheses.

TF returned to the question of how long the expressions can get with the following example:

$$g(f : (a, b : atom)atom) := (a, b : atom)f(f(a, b), f(b, a)) : (a, b : atom)atom.$$

$$f_1(a, b : atom) := \# : atom.$$

$$f_2 := g(g(f_1)) : (a, b : atom)atom.$$

$$\cdot \qquad\qquad \cdot$$

$$f_n := g(g(f_{n-1})) : (a, b : atom)atom.$$

With such an expression, the length of a final expansion is $\Omega(2^{2^{2^n}})$, where n is the number of definitions. Future extrapolation along these lines led to a function of approximate order $2 \uparrow\uparrow n$.

The team of HJ, LW, DM, and BH presented their report next. They also used YACC to convert **CHEX** scripts into S-expressions. BH did the length calculation with a brute-force full-expansion routine. His idea of using DAGs and cached expansions was unfortunately never implemented due to time constraints.

There was some controversy about what the final expanded form should look like; must $(x, y : t)$ be expanded to $(x : t; y : t)$ or not? DEK said that yes, this expansion should be performed. TH mentioned that if just the number of identifiers were counted, a number of the same order would be found.

The team of RZ, TF, CS, and RC were next. They implemented a proof checker and found some bugs in it with some pathological self-referential test cases they made up themselves, but it did run correctly over DEK's scripts. The bugs their checker found in the five **CHEX** scripts match the bugs reported by TH's team, boosting confidence in both proof checkers.

They keep all variable bindings in an environment instead of substituting in expressions; TH's checker replaces bound variables in expressions whenever an expression with a nonempty domain is applied to arguments. It was felt that TH's approach was easier to implement but slower in execution. Checking script 2 and script 3 took 2 minutes with RZ's checker on a Lisp machine; the same scripts required 45 minutes on a DEC 2060 with TH's checker. These numbers cannot be compared directly because of the different machine environments. BH also mentioned that just one inefficiently coded Lisp function might account for such a speed difference.

DEK finished class by noting that TR and a few friends were in St. Louis trying to win a programming contest. He wished us luck, the reports were turned in, and class was dismissed.

19 February 1987

Playing the Slots

Once again DB won the dress-up contest, although DEK made a valiant effort.

Class started with DEK telling how he recorded the slot machine data in Carson City (the particular casino remains a secret). His first attempt was just to write down his observations on paper, but he was physically (but nicely) dissuaded from this. Not one to be deterred, he strapped the smallest tape recorder he could find to his chest, and attempted to record the numbers using Morse code. After some initial observations, he went into the men's room to verify his data, and found that he could not understand his Morse code above the noise of the slot machine. Being musically inclined, he decided to 'hum' the data into the tape recorder, choosing a pitch for each symbol. By this point, he was most certainly under very careful scrutiny, and his observers must have wondered at this oddly random tune, but he got his data and actually won money to boot. This was on a nickel machine. (Maybe other machines have different payoff precentages?)

Someone asked which column of the data was the actual 'pay line' determining the nickels that came out. DEK said that this particular machine had multiple paylines; you could choose any one of a number of lines on which to bet. He did not recall whether his payline was the second column or the third.

DB pointed out similarities to some biomolecular problems. DEK agreed that there were some similarities, and pointed out that he has a DNA problem saved up for 1989, the next time he will teach CS304. The idea is to reconstruct a sequence from fragments of that sequence.

DB opined that the Post Correspondence Problem (PCP) is also similar, and PCP was then explained by DEK. Given two sequences of strings over some alphabet, let us say $\{\alpha_1, \ldots, \alpha_n\}$ and $\{\beta_1, \ldots, \beta_n\}$, is there a way to construct a string γ that can be formed by some combination of α's as well as some combination of β's? This is a classic unsolvable problem, even for small n. DEK hopes that the slot machine problem will not be this difficult.

There are some big differences in these problems, however. In the DNA problem, you have the approximate length of the molecule, but in the PCP problem you do not have any idea how long the matching string might need to be. In addition, our slot machine problem is probabilistic.

JS conjectured that the first column of the data does not look random at all. He asked if the observations were consecutive, and DEK said that for the most part they were. He gave us 92 data points out of approximately 95 observations, some of which were lost in the noise. SS thought that there would be no reason to make a slot machine irregular, but DM mentioned that, the casino wants a customer to win periodically so they keep playing. DEK also brought up that people do not generate random numbers very well, and they often find patterns in random number tables that cause them to think the tables are not random.

JS said that if he fixed the wheel size and computed 5% confidence intervals for the number of symbols, then no wheel is possible. Therefore at least one symbol occurs out of its 5% interval. DEK countered this by saying that if you do twenty experiments with truly random data., one of them will appear to be nonrandom because it is outside the 5% interval He said that the χ^2 test is used to tell how well a sequence of observations fits an expected probability distribution, and mentioned that a fit can be too good as well as too poor. In addition, we have more to approach the problem with than just the number of appearances of each symbol; we also have some ordering information.

KR made the point that the wheel might spin close to 1 1/2 times each time for instance, so the transitions aren't really random. Does the data suggest that the wheel spins a. fixed amount each time? DEK pointed out that the wheels are of the same size, and they stop from left to right. SS said he had read in a magazine that today's slot machines have a. maximum of 84 symbols on one wheel, but casinos want even more symbols.

DM found some patterns in the wheel. On the first wheel, the L's and 7's occur only even numbers of positions away, and the O's, C's, J's, and B's also occur only even numbers of positions away. Therefore, unless we have not seen a portion of the wheel, the L's and 7's only occur on the even positions and the others on the odd, and the wheel size is even.

DEK then formulated a metaproblem; what is the optimal wheel of a given size for psychological reasons? SS said such a wheel would show the highest number of high symbols, yet still have the lowest payoff.

DS came up with a wheel of size **16** that fits the patterns for the first wheel by hand, and conjectured that it was the smallest wheel which fits the data. DM had written a program that attempts to fit the sequences into a wheel, and found that there were 11 unique patterns of length 4 for the first column, and 12 for the second. The first column can fit into a size 14 wheel, and the second into a size 16 wheel.

MD made the observation that there are no C's in the last wheel. That might explain why there is no payoff for CCC.

TF said it might be interesting to figure the probability of a sequence of 4 not occurring in 92 observations, and conjectured that this probability would be very small. If this is the case, then we have probably seen every symbol on the wheel at least once. Of course, this probability depends on the size of the wheel. We might be able to calculate our wheel size by calculating, for each wheel size, the probability that we would have made the observations we did, including repeated patterns.

KR made an interesting observation. If we overlap two observations of 4-sequences that match the first two symbols of one sequence to the last two of another, do the middle four symbols of the new 6-sequence occur in another observation? KR said they appeased to, so it was probable that we have seen the entire wheel.

JS directed conversation back to the χ^2 test, and DEK told another interesting story. Most textbooks which describe the χ^2 test have a rule of thumb which states if $np_i \geq 5$ for all i, then the model should fit fairly well. DEK and a student of his studied the exact distributions for certain choices of p_1, p_2, and p_3 and found that the traditional rule of thumb is not very good; convergence to the limiting values as $n \to \infty$ was slower than expected. He regrets never having published these examples.

JS said that his calculations show that with a wheel size of less than 29, it is very unlikely that there is a. 4-sequence that is missing. DEK wasn't convinced by this; each 4-sequence was unlikely to be missing, but with 29 sequences, one might perhaps be missing.

JS then corrected himself; he really computed the probability for a. symbol that was never seen.

DEK then proposed a subproblem for the class to think about. Let us assume we have a wheel of size 10 with four symbols. Hypothesis h_1 states that there is 1 A, 2 B's, 3 C's, and 4 D's. Hypothesis h_2 states that there is 1 A, 2 B's, 4 C's, and 3 D's. We take 20 readings, and see 2 A's, 3 B's, 7 C's, and 8 D's. What are the probabilities for each hypothesis being correct, if we assume that one of them must be? He then explained multinomials. If there are b_1 A's, b_2 B's, b_3 C's, and b_4 D's out of s total symbols on the wheel, the probability that we will see a_1 A's, a_2 B's, a_3 C's, and a_4 D's in n observations is

$$\binom{n}{a_1, a_2, a_3, a_4}\left(\frac{b_1}{s}\right)^{a_1}\left(\frac{b_2}{s}\right)^{a_2}\left(\frac{b_3}{s}\right)^{a_3}\left(\frac{b_4}{s}\right)^{a_4}$$

The 'choose' notation extends directly from binomials:

$$\binom{n}{a_1, a_2, a_3, a_4} = \frac{n!}{a_1!a_2!a_3!a_4!}$$

DEK then presented another subproblem. Let's try to put the sequences together using graph theory. Imagine that we have a four-sequence ABCD; we might form a node labeled ABC and a node labeled BCD, and label a transition between the nodes with D. If we build such a graph for all of the patterns (nodes with the same name are the same node), a Eulerian cycle will give us a possible wheel, assuming we have seen the entire wheel. (A Eulerian cycle is a path through a graph such that each transition is used exactly once.) The question arises, then, as to how many symbols to put in each node. The above example uses three; would two work better? Multigraphs might need to be used as well; these allow multiple transitions between a single pair of nodes. BH mentioned that we should number the transitions with the number of times each transition appears in the input data to help build the multigraph. DEK agreed, saying that the number of 4-sequences, triples, pairs, and single symbols in the graph should be in accordance with the input data.

DEK compared Eulerian paths with Hamiltonian paths. Finding a Hamiltonian path is a classic *NP*-complete problem; finding a Eulerian path is easy. Any connected graph that satisfies Kirchhoff's law (the number of incoming edges is equal to the number of outgoing edges) has an Eulerian path.

DEK brought the Netherlands into this problem as well when he brought up deBruijn cycles. These are cycles of length 2" that contain all possible patterns of length n; for instance, a deBruijn cycle of length 3 is BBBAAABA. There is a straightforward construction; just start with n B's, then choose A whenever you can do so without repeating a pattern that has already occured. This idea can be generalized for larger alphabets.

24 February 1987

Spinning Our Wheels

DEK started class with a comment about statistical inference; what is the degree of confidence*? He said that different groups of statisticians disagree violently about how to answer this question, or whether it even has a meaningful answer. But in our problem 4, we had a discrete case for which the issues might be clear enough that we can understand them.

We started discussion by taking the 49 shown on the second hand of a watch modulo the 16 class members, started with 0 at KR and ended up at MR. MR asked what kind of assumptions we were going to make about the wheel. Should we assume the wheel is fair? DEK said he would find it very hard to believe that it was not fair. BH related a story about a friend who had two very old slot machines and was trying to fix them up. The friend asked BH to come over and look at the mechanics. Each of the spinning wheels had a hole for each symbol, and a timer drove a bar through these holes, one at a time,, to stop them. Interestingly, some of the holes were blocked. DEK retracted his comment about fairness, but said that the multiple paylines made it more likely that the machines were pretty fair.

CS mentioned that the casinos usually advertise the payoff of their slot machines, and asked if DEK might remember this figure. DEK could not remember the exact number. SS mentioned that you have to make some assumption about the relative probability of various payoffs; 95% is believable, but 300% is not.

SS then mentioned that Bayes' theorem might have some application to this problem. DEK said that the applications of Bayes' theorem were highly controversial for a while; it must be applied carefully. He then went on to describe Bayes' theorem.

Let $P(A)$ represent the probability of A, and $P(A \mid B)$ represent the probability of A given that B is true. By definition we have

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

Let us say that we have n disjoint possibilities, represented by $A_1, \ldots, A_n$, one of which must occur. Let E be some other event, depending on the A_i's in some fashion. The above equation implies that

$$P(A_i \mid E) = \frac{P(E \mid A_i)P(A_i)}{P(E)}$$

The denominator $P(E)$ can be expressed as follows, since the A_i are disjoint and exhaustive:

$$P(E) = P(E \mid A_1)P(A_1) + \cdots + P(E \mid A_n)P(A_n)$$

Using these equations together with the $P(E \mid A_i)$'s, we can calculate $P(A_i \mid E)$, if we make some assumptions about the values $P(A_j)$.

For example, in the problem posed last time, let E be the event that 2 A's, 3 B's, 7 C's, and 8 D's show up. They Bayes formula

$$P(H_1 \mid E) = \frac{P(E \mid H_1)P(H_1)}{P(E \mid H_1)P(H_1) + P(E \mid H_2)P(H_2)}$$

gives the probability that H_1 is the correct hypothesis. We have

$$P(E \mid H_1) = \binom{20}{2,3,7,8}\left(\frac{1}{10}\right)^2\left(\frac{2}{10}\right)^3\left(\frac{3}{10}\right)^7\left(\frac{4}{10}\right)^8$$
$$P(E \mid H_2) = \binom{20}{2,3,7,8}\left(\frac{1}{10}\right)^2\left(\frac{2}{10}\right)^3\left(\frac{4}{10}\right)^7\left(\frac{3}{10}\right)^8$$

hence the 'a posteriori probability of H_1 is

$$P(H_1 \mid E) = \frac{3^7 4^8 P(H_1)}{3^7 4^8 P(H_1) + 4^7 3^8 P(H_2)} = \frac{4P(H_1)}{4P(H_1) + 3P(H_2)}$$

However, we cannot evaulate this unless we know the 'a priori' probabilities $P(H_1)$ and $P(H_2) = 1 - P(H_1)$ of hypotheses H_1 and H_2. Bayesians often assume that, in the absence of other information, $P(H_1) = P(H_2) = \frac{1}{2}$. This is controversial. But if it is true we would say that-hypothesis 4/7 of being correct, given observation E.

DEK said that he had been thinking of another "minimax" way to decide between two hypotheses, but he was not sure if it was valid. The class pressed him to explain more, and he did, but the formulas do not seem to be relevant to problem 4 so they will be omitted here.

DEK then asked what people thought the wheel might look like. BH said that it did not look very random to him. DEK said that the current issue of ***Discover*** magazine has an excellent article about randomness. Among other things it points out that people have very poor perception of randomness. Professor Tversky of the Psychology department is quoted for his study of basketball shooting percentages. He determined that there was no such thing as 'hot hands'; the players' shots fit probability theory well.

AW did a χ^2 analysis on the relative numbers of each individual symbol for each size wheel. He mentioned that the ranges observed for J's and B's indicated that there were more J's than B's oh both the first and second wheel, making the JJJ payoff of 100 much more likely than the relatively small BBJ payoff. DEK immediately shot down the conjecture that something was amiss by reminding us that casinos use psychological strategies to get and keep the customers playing.

DEK then pointed out an interesting fact in the χ^2 analysis in volume 2 of his ***The Art*** of ***Computer Programming*** series. The first edition had an incorrect approximation formula for the χ^2 distribution function, and it was amazing how many people write letters reporting that error. The table for this distribution on page 41 was taken from another book. He did not calculate his own values, he claimed with obvious guilt in his voice, because that was numerical analysis and he does not know much about numerical integration.

DB then put forth the claim that the wheels had to be at least of size 16. In addition, if you assume that you are missing at most one 4-sequence, the first wheel must be of size 16 + $\{4,6\}$ and the second of size 18 + $\{4,6\}$, where {a., ***b}*** is a shorthand for ia + ***jb*** where ***i*** and j are integers greater than or equal to zero. He supported this conjecture by drawing the transition graphs for the first two wheels. For the first wheel,

$$\begin{tikzcd}
\mathtt{LCL} \arrow[d] & \mathtt{OLC} \arrow[l] & \mathtt{LOL} \arrow[l] & \mathtt{BLO} \arrow[l] & \mathtt{7BL} \arrow[l] \\
\mathtt{CLJ} \arrow[r] & \mathtt{LJL} \arrow[r] & \mathtt{JLO} \arrow[r] \arrow[u] & \mathtt{L07} \arrow[r] & \mathtt{07B} \arrow[u]
\end{tikzcd}$$

In order for this graph to obey Kirchhoff's law, the five left-most transitions must be doubled. With this stipulation, the wheel would be of size 16. In addition, any number of transitions around the left-most cycle could be added, so the length could be 16 + (6). If there is a 4-sequence that we have not seen, then there is a possible transition from BLO to L07; this additional transition would require that the three right-most transitions be doubled, so the size would then be 20 + $\{4,6\}$.

For the second wheel, we have the more complicated graph

$$\begin{tikzcd}
\mathtt{07L} \arrow[d] & \mathtt{B07} \arrow[l] & & \\
\mathtt{7LB} \arrow[r] & \mathtt{LBO} \arrow[u] \arrow[d] & \mathtt{BLB} \arrow[l] \arrow[r] & \mathtt{LBJ} \arrow[dd] \\
\mathtt{OBJ} & \mathtt{BOB} \arrow[l] \arrow[r] & \mathtt{OBL} \arrow[u] & \\
& \mathtt{JBO} \arrow[u] & & \mathtt{BJB} \arrow[ll]
\end{tikzcd}$$

This graph presents a small problem; as given, there must be a transition from OBJ to BJB. If this transition exists, then we must double the edges BJB→ JBO, JBO→BOB, BOB→OBL, and OBL→BLB. This then gives us possible wheel sizes of 18 + ***(4,*** 6}. BH noticed that only one data point justified the existance of the node OBJ, and such a point would require another J on the wheel. He mentioned that it was good that point was in there, as he was trying to construct an algorithm that would reject data points if they do not seem to fit. He expressed a hope to make it general enough. MR pointed out how easy it would be; simply have a check on the input stream: **if** *data*=BOBJ **then** ***reject.*** MW made a comment which will certainly reserve him a place in history: 'That's not a kludge, that's a heuristic.'

After some discussion, it was decided to drop the data point BOBJ, and replace it with BLBJ; it was felt that this data point represented an error in transcription. (DEK had two unreported data points for the last wheel, one of which was BLBJ. Given the unreliable nature of his recording method, it is higly likely that BOBJ was a typographical error caused by humming the wrong note.) With this point taken out, only the edges BOB→OBL and OBL→BLB need be doubled, yielding possible sizes of 14 + $\{4,6\}$. There is also a possible edge from JBO to B07, if we missed a single 4-sequence; this edge would require in entirety the doubling of B07→07L, 07L→7LB, 7LB→LBO, LBO→BOB, BLB→LBJ, LBJ→BJB, BJB→JBO and the tripling of the edges BOB→OBL and OBL→BLB. This configuration would require wheel sizes of 24 + $\{4,6\}$,; since the wheel size is probably smaller than 25. only 24 would be possible in this case.

Combining these observations, we notice that the wheel cannot be of size 16 or 18. The only remaining possible sizes are 20, 22, and **24.** DB said that it might be possible to generate all possible wheels of these sizes, and see which are most consistent with the data.

26 February 1987

More Slots

DB started class when he said that the approach he and his team was looking at was to analyze all possible wheels, discarding those with a payoff of greater than 100%, and then use Bayes' rule to compare their relative likelihood. Then, for each possible wheel, also calculate how closely they fit using χ^2 analysis.

TH brought up an interesting problem that managed to consume most of class-how many different wheels of size n are possible, given a set of c symbols, not including rotations and reflections? DEK mentioned that this is called the necklace problem. This problem had been assigned as a programming project to students when, back in the early days of the PhD program, a one-day programming project with everyone working on the same problem sufficed. We looked at a simple case first; let us say we have 4 positions and 2 symbols. There are only six possibilities:

A A A A   A A A B   A A B B   A B A B   A B B B   B B B B

On the other hand, there are sixteen ways to fill four slots with two symbols. It is obvious that one cannot simply divide 16 by the number of permutations of 4 objects, since 6 does not divide 16.

We attempted to solve this problem for some time. First, we counted the number of times each of the above six cases occurred in the sixteen combinations, and realized that the number of rotations might be derived from the factors of n. After significant effort and some educated guesses, we arrived at the solution

$$\frac{1}{n}\sum_{d|n}\phi\left(\frac{n}{d}\right)c^d.$$

In this equation, ϕ is Euler's function, and can be computed by:

$$\phi\left(\prod_i p_i^{a_i}\right)=\prod_i\left(p^{a_i}-p^{a_i-1}\right)\qquad p_i\text{'s prime}$$

DEK apologized for spending so much time on this. In fact, he believes that such enumerations, though interesting mathematically, are hardly ever relevant in real-world applications. For example, a random process will hardly ever choose A A A A out of the sixteen possible necklaces with probability exactly 1/6. When an algorithm is analyzed under the assumption that all nonisomorphic input data sets are equally likely, we usually get results that do not jibe with practical experience.

DEK now changed the subject and mentioned how statisticians often come up with different interpretations of the same data, and said that it would be interesting to compare the results of the different groups. He asked if any groups were still going to be doing χ^2 analysis, as he was interested in comparing methodology, and one group gave a definite yes.

A wheel-counting problem was brought up to compare it with the necklace counting problem. How many different ways are there to choose n beads from c colors, given that the order is irrelevant? If we add the additional constraints that there must be at least one bead of each color, a solution technique becomes clear. Place 12 markers down, and simply choose $c - 1$ divisions between the markers, and assign the colors in order to the resulting intervals. Thus, for this case, the solution is simply

$$\binom{n-1}{c-1}$$

because there are $n - 1$ gaps between n markers. To remove the constraint that at least one bead of each color be chosen, we simply invent new variables, mapping the constraints onto the constraints of the original problem:

$$\sum_{1\leq i\leq c} n_i = n, \qquad n_i \geq 0$$

$$\sum_{1\leq i\leq c} n'_i - 1 = n, \qquad n'_i \geq 1$$

$$\sum_{1\leq i\leq c} n'_i = n + c, \qquad n'_i \geq 1$$

which leads to the solution

$$\binom{n+c-1}{c-1}.$$

We then focused our attention on determining the possible wheels. If we use the graph technique presented last time, then we will have as many unknowns as there are arcs in the graph. These arcs must satisfy Kirchhoff's law, however, so that supplies us with a number of linear equations equal to the number of nodes in the graph, reducing the search space considerably. If we put only one symbol at each node on the graph, then we have $n(n-1)/2$ possible arcs and n nodes; if we put three symbols at each node and assume that we have seen every 3-sequence, then using the graph for the first wheel, we have 12 arcs and 10 nodes; if we assume that we have seen every 4-sequence, we reduce the number of arcs by one (because we never saw a 4-sequence corresponding to the arc BLO$\rightarrow$L07).

JS generalized the graph approach by putting each 4-sequence observation onto a node, and labeling the arcs with from one to four letters to another node. Thus, for instance, there might be an edge labeled BLO from the node JL07 to the node 7BL0. Now we look at every Eulerian circuit of length n, where the length is the sum of the number of letters on each arc. The only assumption using this technique is that you have seen every symbol on the wheel. If, however, you restrict yourself to those arcs of length 3 or less, then you are assuming that you have seen every symbol pair on the wheel. With this approach, the number of possible wheels might be estimated and Bayes' rule applied.

DEK then mentioned that those groups doing a χ^2 analysis might take a possible wheel and generate a set of data points randomly from this wheel, and run χ^2 over this generated data set to get a feel for how reliable χ^2 is in this problem. He pointed out that χ^2 only calculates the probability of a wheel in isolation, as opposed to in comparison with other wheels, and that care should be taken when comparing χ^2 results for different wheels.

He brought up an analogy from random number generator testing. There might be twenty different tests for the randomness of a generator, and they might all be passed at different levels, but it is very difficult to come up with one number that gives the 'goodness' of a random number generator. In addition, it is likely that one of the tests will be failed at the 95% confidence level and give strange looking results.

3 March 1987

Jackpot

KR, as his own team, wheeled out his results first. He examined the transition graph approach as presented in the previous two class periods, but felt that it was too restrictive; such an approach made too many assumptions. He settled on an approach with much weaker assumptions based on regular expressions. He demonstrated this approach with an example.

Let us pretend that we have a small wheel, and that we have only three observations: `LL`, `LO`? and `OL`. We desire to construct a regular expression satisfying these observations that represents all possible wheels. We limit our alphabet to observed symbols. We fix one observation to be the beginning of the wheel, and repeat it at the end of the regular expression. If we choose `LL` as this symbol, then we have the regular expression Σ^*LL$\cap$LLΣ^*. We intersect this expression with the expressions Σ^*LOΣ^* and Σ^*OLΣ^* for the observations `LO` and `OL` respectively, to form a regular expression for all possible wheels. To restrict attention to a wheel of a particular length, we intersect the regular expression with Σ^{n+k}, where n. is the length of the wheel and k is the length of our first (fixed) observation. The intersection of regular expressions is known to be a regular expression, and an algorithm for computing the intersection is well known.

KR wrote a program to generate a deterministic finite state machine from these regular expressions, and to simplify it so that he could use it to generate possible wheels. DEK asked how many wheels were found, and KR said that he had no exact numbers yet, but there were a lot,. This phrase was to be repeated often this class period.

DB then presented the results for his team, which included the likes of TF, BH, and MR. They took two approaches, one of which was almost identical to KR's regular expression approach. Like KR, they had no final numbers, or rather, too many numbers. They also tried a. more generalized graph approach. Rather than restrict, themselves to edges of length one, they labeled their edges with from one to four symbols for all possible transitions, with each node being one four-symbol observation. Then they considered possible wheels by traversing this graph with an Eulerian circuit, starting with only the edges labeled by one symbol, and subsequently adding in the other edges. Each set of added edges removed an assumption; for instance, with only the edges labeled by one symbol, an assumption was made that every 4-sequence had been seen. Unfortunately, they had no numbers.

DEK then asked what the probability was that there was a 3-sequence that had not been seen, assuming that the wheel was of size 25. An upper bound for a missing s-sequence on a wheel of size n with m observations each of size k is

$$n\left(\frac{n-k+s-1}{n}\right)^m$$

For a. wheel of size 25, with 92 observations of length 4, the chances that we missed a particular 3-sequence is less than 0.012. The. chances that we missed a particular 3-sequence is less than 0.0005.

BH then described his approach. He used AI techniques because he felt the problem was ill-defined. He designed two agents, one who attempted to push the solution towards being an Eulerian path, and the other who attempted to find a probable solution using χ^2 analysis. Unfortunately, the latter agent was constructed first and had too much influence; the program needs be tuned before it would find a good wheel. No numbers.

JS, MD, and AW then presented their numbers. Yes, they did have numbers! They used the same generalized graph approach (it was JS who originally presented it) and found that approximately 1000 wheels were possible for each of the first two columns, and a couple thousand for the third column. Generating all the possible wheels took their program approximately six seconds. They eliminated redundant edges; for instance, if the subgraph JLOL $\rightarrow$ LOLC $\rightarrow$ OLCL existed, each edge labeled with a single symbol, they would not have an edge from JLOL $\rightarrow$ OLCL labeled with two symbols, as it would be redundant.

They made the assumption that every 2-sequence on the wheel had been seen. (According to the formula above, the chances of there being an unseen 2-sequence is less than 1 in 5000, so this is probably a valid assumption.) Thus, they considered all edges labeled with l-sequences, 2-sequences, and 3-sequences.

DEK asked them how they processed these wheels once they were generated. They used Bayes' rule, assuming that all possible wheels were equally likely. This allows some cancellation. They mentioned some problems with floating point underflow, but they worked around these.

For the first column, the found a wheel of length 22 that has a probability of 75%, and a wheel of length 16 with a probability of 25%. All other wheels have probabilities of 0.025% or less. For the second column, a wheel of length 22 has probability 87%, one of 16 has probability 13%, and the rest have miniscule probability. The third column found three wheels of length 22 with probability 25% each which are functionally equivalent, two wheels of length 16 with probability ***12Yo*** each, and a. few more wheels of minimum probability. DEK opined that, if two wheels are functionally equivalent, they might well be considered the same wheel in computing the Bayesian probability.

At this point, we were almost out of time. MW, TH, and RZ mentioned that they found essentially the same results as the previous team, but they found for one column a wheel of length 18 that was more probable than any of length 16. Nonetheless, their most probable wheels were the same. JS checked again and agreed that one of his wheels was indeed of length 18, so there was total agreement. RC, EW, and HJ did both Bayesian and χ^2 analysis, and their results corraborated those presented so far. A wheel of size 22 was the only one to pass the χ^2 test at the 95% confidence level for all three columns. A wheel of size 16 passed the χ^2 test at the 95% confidence level for columns 1 and 2, but not 3. The final team, DM, SS, and CS had essentially the same results.

The final results, then, are as follows. The first two wheels are, respectively, `7BLOLCLJLOLCL JLOLCL JLO` and `7L JC JO JB JC JO JB JC JO JBJC`. The third wheel is one of `7LBOBLBJBOBLBOBLBOBLBO`, `7LBOBLBOBLBJBOBLBOBLBO`, and `7LBOBLBOBLBOBLBJBOBLBO`.

Then we then calculated the payoff for these wheels:

	Chance	Prize	Payoff
777	1	200	200
JJJ	30	100	3000
BBB	30	18	540
BBJ	3	18	54
LLL	50	14	700
LLJ	10	14	140
OOO	60	10	600
OOJ	12	10	120
CCX	264	5	1320
CXY	1188	2	2376
	1648		9050

So, the payoff is 9050/10648 or 85%. There is a probability of 1648/10648, or 15%, that you will win something on a given nickel.

Floppy disks containing library code for the next problem set, and a. description of the routines provided, were handed out at the end of the class period. The *signmanager* document ation follows:

You are given two playgrounds to write in, called *field1* and *field2*, each 1024 horizontally by 32 vertically. $(0,0)$ is at the upper left-hand corner. Each is an array of 2048 16-bit integers, but you don't know that. All coordinates are handled modulo the field size; in other words, your playgrounds are doughnuts. The following three functions can be used to set, clear, and test a bit in the playgrounds.

procedure *pset*(**var** *f*: **field**; *x*,*y*: **integer**);
procedure *pclr*(**var** *f*: **field**; *x*,*y*: integer);
function *ptst*(**var** *f*: *field*; *x*,*y*: *integer*): *boolean*;

For your more ambitious artistic needs, a general-purpose bit-blit routine is also provided. This routine takes a. source field and a. destination field, x and y offsets for both, an x size and a y size, and an operation code, and performs a. bit-blit. If the source and destination arc on the same field and overlap, you take your chances with what will happen.

Again, the bit-blit is performed as if both source and destination were a doughnut. The *op* is a. four-bit parameter whose sisteen legal values allow all possible sixteen operations on two bits. To calculate the value you want, take the function you want to perform, and write it as a function of s and d; for instance, $s \oplus d$. Now, rewrite it, as the sum of the following four products: sd (8), $s\bar{d}$ (4), $\bar{s}d$ (2), $\bar{s}\bar{d}$ (1). Our exclusive or would become $s\bar{d}$ + $\bar{s}d$. Now add the values in parenthesis following each term above; we get 6. This is the value to use! For instance, some common ones are 5 to invert the destination, 12 to copy bits? 0 to clear the destination, and 15 to set the destination.

> **procedure** *blit*(**var** *sf:* field; *sx,sy: integer;*
>   **var** *df: field; dx,dy: integer;*
>   *x,y: integer; op: integer);*

Before you *do* any of the above, however, you *must* call *initialize* to set things up. By default, a text window is opened for standard input and output; use the procedure ***hideall*** to shut this down for final display. Before you exit, call the function *cleanup* to restore the system to its original state.

> **procedure** *initialize;*
> **procedure** *hideall;*
> **procedure** *cleanup;*

The displayed window is a 256 horizontal by 8 vertical window on one of the two fields, with the fields still treated as doughnuts. The following routine sets the origin (upper left-hand corner) of the displayed window to some point. This routine allows quick scrolling, panning, or double-buffering of the display.

> **procedure** *setorig*(**var** *f: field; x,y: integer);*

For I/O, the following routine gets a key if one is available. If none have been typed, it returns *chr(0)*. The d iaracter is not echoed to the screen.

> **function** *checkkey:* char ;

If you want to use specified delays, the *ticks* routine returns the number of sixtieths of a second since the system has been turned on. This routine is useful for your own ***wait*** routine, for instance. In addition, a *time* and *temperature* routine are also provided.

> **function** *ticks: LongInt;*
> **procedure** *time*(**var** *y*, m, d, *h*, min, s: integer);
> **function** *temperature: integer;*

Random numbers are available too. The function *rand* returns a random number between 0 and $i - 1$.

> **function** *rand(i: integer): integer;*
> **procedure** *seed(i: LongInt);*

If you are having difficulty understanding why your code does not work, the following routine will copy the contents of both fields to the top portion of the Macintosh display. Since the fields are twice as wide as the screen, the top half of each field is placed above the bottom half.

> **procedure** *debugdisplay;*

User Interfaces

DEK asked how many groups there were so he would know how many naïve users to find. He said that it would be easier to get naïve users for our demonstration if it was held in the evening, but several students had conflicts that evening. MD mentioned that when she worked at Bell Laboratories, they would hire suburban housewives to test their user interfaces. JS asked what the organization of the demonstrations would be like, and how naïve the users would be. DEK said that he was looking for people who had no experience with computers, but might be visually oriented. He was considering looking for volunteers in the ast department.

DEK told a story of a problem given to a previous class: They were to design a user interface for a terminal to be located directly inside Margaret Jacks Hall, on which random visitors could ask arbitrary questions pertaining to the building and get appropriate responses. Each team of students were asked to volunteer a typical question; one team submitted a. question in French! The interface was intended to be a nice as possible; often features get in the way of user friendliness. There is usually a tradeoff between available features and ease of use. For the DigiFlash problem, we want to allow the user to generate attention-getting displays with ease.

DM then mentioned that he had located one of the LED signs in the Stanford Credit Union. The input was just a. keyboard and the output was the sign itself, just as in our problem. He played with it, and determined that it was programmed through escape sequences. Each message was preceded by an escape sequence which determined how the message would be displayed, followed by the actual message itself.

DM verified that the letters did indeed seem to be slanted as they scrolled. DEK commented on how real this phenomena appeared to be, and explained its source. The lights in the display are displayed one row at a time, from bottom to top. The computer shifts the entire message between the time the bottom row and the top row are displayed. Thus, the upper pixels of a letter might already be shifted one pixel when they are displayed. The eye tends to merge all of the shifting and scanning into a smooth, scrolling, slanted letter.

Unfortunately, we would not be able to replicate this effect with the Macintosh. The display on the Macintosh is updated at only sixty times per second. At that rate, the eye cannot merge scrolling 'broken' letters into a smooth slanted scroll; each frame is shown long enough to make the letters appear irregular. In addition, the resolution of the Macintosh screen is not high enough to simulate the slant effect by actually slanting the letters as they are drawn.

DEK talked then about a sign he saw in a Boston subway that was malfunctioning. It would go through a seemingly random sequence, which included a.11 forms of garbage characters. At one point? half of the screen scrolled one way, the other half another way.

MD mentioned that the Palo Alto post office has one of these displays, which just displays messages on and then blinks them. JS said that this might reflect the users. He taught a. summer computer camp, and he was asked to show the parents around the camp.

The parents would act almost afraid of the computers: they tended to stick to what they knew. The children, on the other hand, would experiment to see what the machine would do. For instance, using LOGO, a child would often intentionally run the turtle off the screen to see what would happen. (For the curious, it wraps around to the other edge of the screen.) An adult would not attempt this; JS mentioned one instance where an adult told him he had accidentally run the turtle off the screen and whether that was okay.

DEK felt that adults were more cautious because adults have been conditioned not to break the rules: this influences how they handle new situations. He explained this with an example. As an undergraduate he wrote a simple game testing reflexes, where you were to hit a button after the computer flashed a light. The faster you hit the button, the higher your score. People soon realize that they can hold down the button to get high scores, so he programmed the machine to respond to a held button with 'Don't hold the button down. Sir? I think you lack integrity.' Adult who played the game became worried that they had offended the machine somehow!

Discussion went back to user interface design when DEK suggested that the sign was large enough to have two active areas, one for user entry and one for help, for instance. RZ said that an easy way to satisfy the one-page user manual requirement was to include the sentence 'Type h for help' and put the user manual in the machine itself.

JS mentioned how much could be done with simply a keyboard and small display; the Rockwell AIM 6502 system was a very successful microcomputer, complete with BASIC and assembler in ROM, and it had only about a 32-character display. DEK mentioned a notebook Epson portable computer with only three lines that was very useful.

RC mentioned that a team might want to include subliminal messages for the judges. Unfortunately, on a video screen refreshed at sixty frames per second, the subliminals would not be very subliminal.

RZ then brought up the subject of modes. DEK said that the machine should never get in a. state the user could not easily get out of, and that a user should always be able to tell what state he is in. Sometimes this is easier said than done. For instance, in TEX, if an argument to a macro is being scanned, an \end will be swallowed, even if the user is typing the input at the keyboard. RC mentioned the difference between the editors vi and EMACS. The former is a moded editor, with separate insert and edit modes; the latter is (almost) always in insert mode, with the editing commands bound to control keys.

Some questionable user interfaces were then presented. TR mentioned that a certain relatively common sequence of characters, which might be typed to the vi edit mode when the user thought he was in the insert mode, would make changes to and write the file to the disk. SS mentioned that the command 'I' is not understood in the XEDIT/Edsgar editor, so it is passed to the operating system. Unfortunately, this reboots the machine! DEK commented on how the E editor on SAIL uses the ϵ and λ keys to move among files; he often finds himself typing them to the TENEX prompt instead. RZ brought up the Unix passwd program, which complains if you ask it to accept a. password that is too short. If you keep trying, however, it will accept it after a certain number of attempts. SS mentioned how TWENEX will accept a password that is too long, and truncate it for you, without, notifying you. Then, if you try and use the longer version of the password, it will not let you log on; you have to figure out the truncated version. RZ suggested that a user-interface hall of shame be started.

MR said that the uniform user interface presented by the Macintosh and its application programs is one of the nicest features of the machines. This interface was designed by Apple and suggested (but not required) for all applications on the machine. DEK hypothesized that the capitalist market supports software that uses the uniform interface and squelches packages that do not.

DEK then brought up an oft-omitted feature of a manual-it should tell the user how to exit the program. Usually it is wise that this be the first thing mentioned. Most manuals assume that once the user is in their program, he never wants to get out!

SS asked if we could put labels on the keyboard, and DEK said yes, that was in fact a good idea. MR took this one step further and asked if he could build his own keyboard; the feeling of the class was yes, he could, if he wanted. SS described the control panel of a spaceship he saw somewhere; it had a START button to go somewhere, and a non-functioning STOP button that was only there to reassure the user. RZ mentioned that this might be like the 'close door' key on most elevators.

On the subject of keyboards, DEK said that most keyboards for Chinese languages designed by westerners had a very large matrix of keys. A Chinese scholar at Stanford, however, opted to use a standard ASCII keyboard, with key combinations to allow the large number of characters. The scholar even suggested a way in which the computer might 'learn' how a particular typist typed each character, allowing the typing to be customized to the individual user.

DEK described a keyboard used by court reporters. It has five pairs of keys for the fingers of each hand, and four single keys for the two thumbs. Each pair of keys can be struck in three combinations; top key, bottom key, and both. Using this keyboard, an entire syllable can be typed in one stroke. Unfortunately, after he learned this keyboard for typical conversations and decided to try and take chemistry notes on it, he discovered that he could not translate the chemistry to strokes fast enough. Like most secretaries, he had learned abbreviations for business letters ('yours very truly') and typed them as units, but he did not have a set of 'units' for chemistry.

12 March 1987

User Headaches

After a long week using the Macintoshes, several issues were raised by the students. JS mentioned the restriction of 32,000 bytes of automatic variables under Lightspeed Pascal, and asked if it was legal to use the memory allocation functions to access more memory. Were we going to assume that the hardware of the DigiFlash executive sign limited us to some amount of memory, and if so, how much? He also mentioned that it would be nice to have more than the two fields declared; this way images could be built up and just swapped into place. Since each field requires 4096 bytes, and an array internal to the signmanager code required 8192 bytes, memory was already fairly scarce. DEK said that a meeting of the DigiFlash board would be held on the following day, during which a. decision would be made. (The decision was to allow the use of extra memory.)

D'S then narrated his experiences with the sign at the Stanford Credit Union. Their sign was taken down when the customers complained about it. It had a. field of approximately 256 by 16 pixels, and includes a script font, among other things. The programming was very simple; before each message, a key was pressed to indicate whether the message would scroll top down, bottom up, from the left, or simply flash. Then, the message was typed, and consisted of a.11 characters until the next control key typed. Basic editing functions were provided; a user could scroll a message character by character or scan through the messages one at a time. In addition, there were a few canned sequences; one was a script 'Welcome'; another was a script 'Thank You', and the third was a flying saucer animation sequence, and the fourth was a slot machine simulation. Interestingly, even though so few features were available, the sign really appeared exciting. Even bland test messages were very impressive. The machine was obviously designed carefully. DEK mentioned how the flashing red lights would have an appeal over the diminuitive black and white Macintosh screen.

After playing with the machine for a while, they finally found the instructions on the back of the keyboard. Apparently there was an entire printed manual? but the people at the Credit Union could not locate it.

DS was convinced by this experience that users should have comparatively few options. A successful system necessarily has very successful defaults.

BH cited this as an example of a properly engineered system, in comparison with so niany systems that were over-engineered. He said, 'You would not believe how complex a system, say, for example, a computer typesetting system, can get.' No names were mentioned. DEK countered this with the notion of tools which were designed to be used by experts as well as novices; the tool must grow with the user. Your demands on the system might change, and the system should be able to accomodate these changes. In addition, the help a system provides a novice should not slow down the expert user.

BH said that this is true, and might influence how the DigiFlash interface was programmed. Was this a sign to be used on Times Square, or just at a local record Wherehouse? MW mentioned that a product designed for demonstration also might differ from a product designed for use, especially if the people exercising the system during the demonstration were only either expert or novice users.

DEK commented on how difficult it was to evaluate the quality of software. MR suggested that systems are usually compared with other existing systems that do the same thing, on a feature by feature basis. BH mentioned that more features do not necessarily make a better product. MD said that user satisfaction can be measured. DEK agreed, saying that sometimes an interface or method is so intuitive and so natural that everyone is left wondering why they did not come up with it first. Yet, before it was seen, the method was not at all obvious. EW mentioned that people need to be convinced that a. system is interesting; a system is not interesting in of itself. MD said that poor packaging can make a good system look bad. DEK agreed, saying that the weakest link is what kills a system.

MR. said that it is a. natural tendency of the designer and implementer of a system to want to add features. Some of these features might not be wanted by end users, leading to frustration of a.11 involved. DEK confessed that he loves England, because everything there is not targetted at the lowest common denominator; the strong class system makes some things more attractive.

DEK then asked what kind of editing functions the students were thinking of providing. If a user has typed in his messages and they are almost right, but contain a few minor errors, he would not want to start over. BH mentioned that it could not be too complex, because the record Wherehouse is not going to hire someone simply to run the sign. MD disagreed, pointing at TR as a potential market for a more complex sign. TR had mentioned earlier that he would not mind having a personal DigiFlash to use for random messages. DEK mentioned that there would probably be a market for people who wanted to install DigiFlash signs in the rear window of their cars. BH asked how much such a system would cost. DEK said, 'Tens of thousands of lives'. MD mentioned that different models could be offered; the ones with larger screens could also include more features. DB pointed out that there is probably a large market for a limited-feature product, and a small market for a more complex DigiFlash. DEK mentioned that a base unit might be sold, and then peripherals and additional software could keep the customers buying, as in the current microcomputer market.

The conversation had strayed from editing features, so RZ brought it back on track. He suggested having incremental searching and regular expression matching, as in EMACS or vi. Class time was exhausted by then, so we took the tongue out of RZ's cheek and class was dismissed.

17 March 1987

User Experiences

Class broke bright and early on the students huddled in the Macintosh laboratory, putting finishing touches on their projects. DEK entered to scattered applause; he was wearing his umpire's shirt. After some scrambling to find machines that worked with the projector, the demonstrations were started. A summary of the demonstration, written by DEK, constitutes the remainder of these notes:

There was general agreement that the most successful team, for the purposes of this demo and test, was team 4: DS, EW, MD, and MR. The research they did at the credit union was excellent, as was the general organization of their user manual.

There was also general agreement that the most attractive displays were produced by team 2: DM, SS, KR, JS, and TH (once the user learned how to enter it). The fonts were excellent and, of course, the animation was a nice surprise.

Team 3: MW, CS, TF, AW, and BH rates honorable mention both for a good user manual and for excellent graphics. The 'materialize' feature was a. creative idea (although it could perhaps have been made more visually exciting with some more experimentation), and the extra functionality of flashing/bold/reverse letters and extended characters was haadled in a way that a. user could easily pick up.

Team 1 fulfilled the necessary function of demonstrating how difficult the assignment was to carry out in just two weeks.

Here are some more specific comments. Team 4: Especially nice features of this user manual were the diagram (with redundancy) and the easy-to-see headings:

OVERVIEW

STEP-BY-STEP

Step 1: Type

Step 2: Play

Step 3: Fancy Displays

Step 4: Play Again

Step 5: Experiment

The wording is notably jargon-free: 'Hold % and press P. Look! DIGIFLASH is playing your messages, one after the other. To go back to editing, use %E.' (Maybe it would be better to say 'hold % DOWN.. .'?) But there are a few glitches: Sometimes the manual refers to 'line' and other times to 'message'; better to say 'message' everywhere. The word 'scroll' is unfamiliar to non-CS types; its use was not necessary. Likewise the word. 'menu'. 'Up' could rather have said 'To previous message'.

Somehow I think both lower and upper case would have been better, even though we saw with team 2 that the way to get this is not obvious to a newcomer. Just say 'to get capital letters, hold the shift key down while typing'?

The statement 'Don't worry, Digiflash II has been carefully redesigned not to bite!' is too apologetic. But something like it (to underline the fact that the user should not be afraid of breaking the machine) is important for most people who are unfamiliar with computers.

The most successful displays will probably not jump back and forth between scrolling options, so it would have been better to use the previous option as the default for each new line.

Icons for the options were nice.

The least successful part of the manual was the inscrutable comment

```
[<- THIS IS VISIBLE HERE] BUT NOT HERE.
```

I think it would have been better to say, e.g.,

```
<- AN EXTRA-LONG MESSAGE CAN BE SEEN IF YOU MAKE IT MOVE
```

and then below, when the options are listed, say

<- move `the message to the left.`

The cursor was not a success. Something that flashes is much easier to spot.

The blinking option went by too fast, on the solution by team 2, and perhaps also in his team's case (I don't remember). The default should be to blink a few times, not just once.

Team 2: The user manual suffered from sentences like 'the backspace key works in the obvious way'; this caa be intimidating. Again, there's unnecessary jargon and inconsistency:

> "When the DIGIFLASH is turned on, a flashing vertical bar indicates that it is ready for you to enter the first line of your message. . .The left and right arrow keys may be used to position the cursor (i.e: the flashing bar) anywhere within the line. Move the cursor to the left of some character and hit the up or clown arrow key a few times to change the character style. . ."

See? No point in saying 'cursor' when you must define it the first time and then you use it only once later. Just continue calling it a flashing bar. Also, these sentences are long and wordy. Formal latin terms like 'i.e.' aren't appropriate even if punctuated properly (this should have said 'i.e.,' not 'i.e:'). Compare to the following sentences by group 4: 'Type some more messages. Use the BACKSPACE key if you make a mistake.' In this style you could have said:

> "Turn on your DIGIFLASH. You'll see a flashing vertical bar. This means it's ready for you to type. When you hit a key, a letter appears where the bar was flashing; try it! To remove a. letter you don't want, hit BACKSPACE."

And so on. Short sentences.

The options for getting in and out were interesting, but the LINK option wasn't easy to understand. I think it would have been better to use LINK for line entry (so that you can demonstrate how the line comes in, knowing how the previous line went out), instead of for line exit (when you have to 'push down' until seeing the next line).

Also, it was a bit boring to be prompted for 5 or so things on each line; some way to say 'same as last line' would be nice. On the other hand,. your way (with the handy '?' option, which could demonstrate LINK if you use it as suggested above) is preferable to that of team 3, since their user didn't know when the mode of display was to be entered.

Your font was pretty good but the periods were too narrow. There was no way to make '. . .' look right.

DiJKSTra as the name was an OK in-joke since we needed a Dutch connection.

Team 3: The biggest mistake here was the way chosen to get ***out*** of bold mode by typing bold a second time! That must be the worst imaginable thing. In a WYSIWYG system like this, the way to get to normal should be just like the way you get to bold, flashing, etc.

I didn't have a chance to check if you updated the time (to keep it current) wherever it appeared in a message. That would have been the right thing to do, of course! Congratulations for providing this feature, with a nice simple user interface.

It would have been natural to allow 8 type styles (flashing/nonflashing, bold/normal, black/white) instead of 4.

The Materialize feature looked rather binary, hence not as appealing as the animation. Some kind of 'dissolve' or selective flashing might be more exciting; this would be interesting for further exploration.

Your instruction 'Scroll from the left' is a problem first because of jargon ('scroll') but mostly because people think of this as a rightward motion! 'Scroll to the right' would have been much better.

Team 1: The user manual goes to great lengths to be explanatory, but fails totally, in part because of the great length. Let me quote from the beginning, and make some comments:

> You can create 6 different, types of MESSAGEs. These MESSAGES differ from each other in
>
> 1. The way they SCROLL (read stroll) on the screen:

(a) Either the message starts from OFF-SCREEN (i.e. starts with a. blank screen), and SCROLLs the message from right to left.Characters enter from the right end,and fall off the left end, of the screen.

OR

(b) The message starts displaying initially with the message LEFT-INDENTED,(i.e. the first character of the message is on the left end of the screen.)It then SCROLLs the MESSAGE across from right to left as in (a).

2. The NATURE of the characters on the screen.. .

and so on. Notice how unhelpful all this itemizing and wordiness is. Why are some words capitalized sometimes? Technical terms need to be defined only if they're going to be used often and if there are no ordinary English substitutes. People are not normally so logical that they like to see formal definitions. Also, the term 'left-indented' is contrary to normal usage (since 'indented' always means moved in from the margin, not flush with it). That's almost like saying 'Henceforth I'll say up when I mean down.'

I tried to repeat faithfully the weird spacing around punctuation in the above quotation. Mechanics of typography are important, since deviation from normal conventions is an unnecessary distraction.

How to rewrite the above? I would probably not start by saying how many types of messages there are; I might never even mention that explicitly. I'd probably say, 'The display of messages either starts ONSCREEN-with the beginning of the message instantly visible-or OFFSCREEN-with the screen starting out blank and the message moving in from the right. In both cases the message continues to move from right to left.'

Instead of having a user type a number between 1 and 6, it would be much better to have marked keys. For example, there might be an ONSCREEN key and an OFFSCREEN key, given your setup.

When both uppercase and lowercase letters are used, 8 dots high, it's much better to leave the bottow row for descenders on g, j, p, q, y.

The importance of quick response to keystrokes was amply demonstrated.

Overall, this obviously turned out to teach quite a few lessons! Besides all the technical things, I was especially happy to see the interaction between team members; it's hard to work together on something like this, and you did well indeed.

Will we ever again be able to watch a flashing sign without remembering these traumat ic days??

Appendix A: Trivia Hunt

1. Who taught this course in 1971? In what room did it meet? What color are the walls and doors of that room painted now? 15 points *each*

Prof. Floyd taught CS204, Problem Seminar, in autumn quarter 1971-1972. The room was Math 380-D, according to the Time Schedule [courtesy of Stanford Editions and Publications]. The doors are stained brown; the walls are covered with wallpaper: white in front, beige at the sides, brown at the back. Stanford Maps and Records has no record of changes since 1971; floorplans dated 1971 show the room as it now stands. (When Knuth posed this problem, he was thinking of room 204 in Polya Hall, where he taught the class some years later; that room has off-white painted walls and dark blue doors.) Some teams interviewed Floyd and Martin Frost (who was one of the students in the course that quarter), but neither Floyd nor Frost could remember whether the class was actually held in that room.

2. Who received the first two PhD's in Mathematics from Stanford? *10 points each*
   In what years did they finish their work? *10 points each*

William Albert Manning received the first, on May 18, 1904, according to the 13th commencement program. Frank Clark Hoyt received a. PhD jointly from Math and Physics on June 20, 1921; this may qualify although the title was definitely physics. Another such was awarded to George Russell Harrison in 1922. But according to the reference book Stanford Dissertations (call number AS36.S7), and according to a. definitive-looking list, in the math library compiled by Prof. Bacon in 1972, the second math PhD was completed by Vern James (a student of Manning) in 1927.

3. Who received the first two PhD's in Computer Science from Stanford? 5 *points each*
   In what year did they finish their work? *5 points*

William Marsha.11 McKeeman and Dabbal Rajagopal Reddy completed their theses in August, 1966 and their degrees were awarded in September of that year. They are listed under 'Computer Science' in the 1967 commencement program. Previous students who received PhD's in the Computer Science Division of the Mathematics Department (Causey, Grace, Moler, and Rudin) were listed under 'Mathematics' in the commencement programs; there was not as yet, a Computer Science department.

4. What was the title of John McCarthy's PhD thesis? 10 *points*
   How many pages long was it? **10** *points*

Projection Operators and Partial Differential Equations. Princeton University, 1951. 28 pages. [Reference: Dissertation Abstracts 15 (1955), p. 601.] However, somebody had a friend look up the thesis in Harvey Mudd library at Princeton, and it. really was only 27 pages long.

5. When did Gene Golub celebrate his 13th birthday? *8 points*

Gene was born on February 29, 1932, so his 13th birthday was February 29, 1954; several students (e.g., Schaffer, Weening) remember the party. He also celebrated age 13 somewhere near February 28 or March 1, 1945, when his bar mitzvah took place.

6. Of what journal is Leo Guibas the problem section editor? *12 points*

The Journal of Algorithms. [Reference: Inside front cover of any issue.]

7. Give the titles, coauthors, and years of publication of each book by Jeff Ullman.
4 points each

The following are in the card catalog (and they are also known to Socrates): 1. (with Hopcroft) Formal Languages and their Relation to Automata, 1969. 2. (with Aho) Theory of Parsing, Translation, and Compiling, vol. 1, 1972. 3. (with Aho) Theory of Parsing, Translation, and Compiling, vol. 2, 1973. 4. (with Aho and Hopcroft) Design and Analysis of Computer Algorithms, 1974. 5. Fundamental Concepts of Programming Systems, 1976. 6. (with Aho) Principles of Compiler Design, 1977. 7. (with Hopcroft) Introduction to Automata Theory, Languages, and Computation, 1979. 8. Principles of Database Systems, 1979. 9. (with Aho and Hopcroft) Data Structures and Algorithms, 1983. 10, Principles of Database Systems, 2nd edition, 1983. 11. Computational Aspects of VLSI, 1984. 12. (with Aho and Sethi) Compilers: Principles, Techniques, and Tools, 1986. His vita also lists books scheduled to appear: 13. (with Goldschlager and Mayr) Theory of Parallel Computation, 1987. 14. Database and Knowledge-Base Systems, 1988. 15. (with Aho) Principles of Computer Science, 1988.

8. What was Don Knuth's first scientific publication? ***10 points***

The Potrzebie System of Weights and Measures, MAD magazine 33 (June 1957), 36-37. [Reference: Knuth's vita.]

9. What faculty member in our department wrote a PhD thesis about game theory?
20 points

Nils John Nilsson, An application of the theory of games to radar detection problems (EE, 1958). Found in main card catalog and elsewhere.

10. Which faculty members in our department had thesis advisors who have won the ACM Turing award?
20 points for each correct triple (faculty name, advisor name, year of Turing award)

(Feigenbaum, Simon, 1975); (Guibas, Knuth, 1974); (Gupta, Newell, 1975); (Manna, Floyd, 1978); (Manna, Perlis, 1966); (Nelson, Tarjan, 1986); (Pratt, Knuth, 1974); (Rosenbloom, Newell, 1975); (Waldinger, Simon, 1975). [Personal communications; the Turing winners are listed in CACM, January 83.]

11. Identify the author and source of the following quotations:
10 points for each author
15 points for each source

a. Right as a serpent hit hym under floures
Til he may seen his tyme for to byte . . .

Geoffrey Chaucer, The Squieres Tale [part of his Canterbury Tales], lines 512-513.

b. The upper pointer is stepped down, and proceeds on its downward scan of the data. When it finds an item with key lower than the bound, this item is copied into the locations referred to by the lower pointer. The lower pointer is then stepped up, and the process is repeated until both pointers are referring to the same item.

C. A. R. Hoare, "Quicksort," Computer Journal 5 (1982), p. 13.

c. The most valuable acquisitions in a scientific or technical education are the general-purpose mental tools which remain serviceable for a lifetime. I rate natural language and mathematics as the most important of these tools, and computer science as a. third.

George E. Forsythe, "What. to do till the computer scientist, conies," American Math Monthly 75 (1968), p. 456; quoted by Knuth in CACM 15 (1972), p. 722.

12. What were the titles and dates of Stanford CS reports number 1 and number 1000?
Of Stanford AI memo number 1? 5 *points* each
Were any of these three subsequently published in journals?
10 points ***for each journal citation***

CS 1: Primal partition programming for block diagonal matrices, by J. B. Rosen, November 8, 1963; published in Numerische Mathematik 6 (1964), 250-260. CS 1000: Implementation of logical query languages for databases, by J. D. Ullman, May 1984; published in ACM Transactions on Databases 10 (1985), 289-321. (Also in the supplement to the ACM-SIGMOD 1985 conference proceedings, pp. l-17, where it was an invited paper.) AIM 1: Predicate calculus with 'undefined' as a truth value, John McCarthy, March 22, 1963; never published.

13. What integer number does the bit pattern 101011100010111000101110001011100011 represent in a DEC-2060 computer?
3 ***points for*** *the* ***correct answer, in decimal notation***
What floating-point number does it represent?
7 ***points for the correct*** answer, ***in decimal notation***

The integer is -21963283741, and the floating-point number is −2.1963284E+10, according to the 'ddt' program. The latter represents -21963283712 ± 128. (This and its negative are the only self-referential numbers on the machine!)

14. Where did Seidel publish his method for solving linear systems by iteration?
10 ***points for correct*** *reference,* ***plus*** 15 ***points for copy of first page***
What was his full name? What were the dates of his birth and death?
7 ***points each***

"Über ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate fiihrt, sowie lineäre Gleichungen überhaupt, durch successive Annäherun aufzulösen," in Abh. der II. Classe der k. bayerischen Ak. d. Wiss. [that's the proceedings of the math-physics section of the Royal Bavarian Academy of Sciences], vol. 11, part 3 (Munich, 1874), 81-103. He was Philipp Ludwig von Seidel, according to Meyer's Enzyklopädisches Lexikon and Scribner's Dictionary of Scientific Biography; or Philipp Ludwig Seidel according to Poggendorf's Handwörterbuch; both sources give birth date 24 Oct 1821 in Zweibrücken, death date 13 Aug 1896 in Munich.

15. What is the number of the patent on the ENIAC? 10 ***points***
How many claims did it have? 10 ***points***
When was it filed, and when was it awarded? 10 ***points***
When was it invalidated? 15 ***points***

Patent number 3,120,606. Filed 26 Jun 1947, awarded 4 Feb 1964(!). 148 claims. Invalidated 19 Oct 1973. [US Gov't Patent Office Official Gazette 799 (February 1964); Annals of the History of Computing 3 (1981), 389; US Patents Quarterly 180 (1974), 673–773.]

16. What city has been the site of both a STOC and a FOCS conference, since 1976?
15 points

Providence, Rhode Island, was STOC 85 and FOCS 77. [See the conference proceedings, or CACM calendar of events, or Directory of Published Proceedings.] Incidentally, Berkeley was STOC 86 and FOCS 75.

17. Find all journal articles published in 1982 whose title contains the word 'multigraphs'.
10 points ***for each*** correct *citation,* ***plus 5 points for each copy of' the first*** page

1. Read and Robinson, Enumeration of labelled multigraphs by degree parities, Discrete Math 42 (1982), 99-105. 2. Exoo and Harary, The smallest cubic multigraphs with prescribed bipartite, block, Hamiltonian and planar properties, Indian J. Pure and Applied Math 13 (1982), 309-312. 3. Gabow and Kariv, Algorithms for edge coloring bipartite graphs and multigraphs, SIAM J. Computing 11 (1982), 117-129. 4. Taqqu and Goldberg, Regular multigraphs and their application to the Monte Carlo evaluation of moments of non-linear functions of Gaussian random variables, Stochastic Proc. and their Applications 13 (1982), 121-138. [The first three are in the CompuMath subset of Science Citation Index, but the last is only in the full index!]

18. The fundamental reference to the theory of multigrades is a book by Albert Gloden called *Mehrgradige Gleichungen* published in the 40s. Find a review of this book in English. ***10 points for*** a *xerox* ***copy***
Find all references to this book made in scientific papers since 1976. ***15 points*** each

Reviewed by D. H. Lehmer in Math Reviews 8 (1947), 441. Referred to by E. M. Wright, "Tarry-Escott and the easier Waring problems," J. fiir die reine und angewandte Math 311 (1979), 170-173. Also referred to by G. Myerson, "How small can a sum of roots of unity be?" American Math Monthly 93 (1986), 457-459.

19. The longest known multigrades were first published in 1942, in a journal called Gazeta *Matematica*. What university libraries in the United States contain the 1942 issue of that journal? ***10 points each***

According to the Union List of Serials (1943), the only libraries at that time were at the following universities: Brown, Chicago, Columbia, Harvard. However, a check at Brown revealed that issues were not received between 1940 and 1946. Other libraries may have acquired back issues since then; this question is not resolved. Full credit was given for answers that believed the Union List of Serials.

20. Find the longest words in the English language with the following respective properties: (1) All the letters are distinct (e.g., absorpt ively). (2) The word is a palindrome (e.g., madam). (3) The letters of the word are sorted in nondecreasing order (e.g., accept). (4) The letters of the word are sorted in nonincreasing order (e.g., rookie). (5) The letters all appear on the upper row of a typewriter keyboard (e.g., typewriter). Note: If your words are not commonly known, you must state their meaning and give the name of a standard English dictionary that lists them. *12* ***points for each category, provided that no longer*** word ***has been found by other teams***

(1) dermatoglyphics, skin patterns. (2) kinnikinnik [Webster's 3rd], 'a mixture of dried leaves and bark and sometimes tobacco smoked by the Indians and pioneers esp. in the Ohio valley'. Next in line were **malayalam**, a Dravidian language related to Tamil [Guinness Book of Records], and redivider. (3) aegilops, an ulcer in the inner corner of the eye, also a genus of grasses. (4) spoonfeed. Next is spoonf ed; then wronged and sponged and some less common words. (5) prettypretty [Century Dictionary, 1914], 'a knickknack'. Next is rupturewort, a European plant of genus Hernaria. The book Language on Vacation by Dmitri Borgmann (1965) has more info on questions like this. Some noted that the longest words satisfying all five criteria. simultaneously are 'I' and 'O'.

21. What are synonyms of 'cretinous' in hackerese?
5 *points* ***for*** *each documented answe*

bagbiting, barfulous, barfucious, bletcherous, brain-damaged, chomping, demented, losing. [The Hacker's Dictionary, by Steele et al. (1983), pp. 28, 29, 36, 49.]

22. List all computers at Stanford that run the TOPS-20 operating system, and tell in what building they are located.

4 points for each correct name and 6 points for each correct location

Various 'host' commands identify them as: score, sushi, and truffle in Margaret Jacks Hall; lear, othello, hamlet, and macbeth (aka lots-a thru lots-d) in CERAS; how and why in the Business School; sumex and tiny in the Medical School; sierra in Durand; turing (aka csli) in Pine Hall. There's also 'panda' at Mark Crispin's home; this may qualify as 'at Stanford' in some loose sense.

23. Where is it possible to see a life-size painting of Leland Stanford, Jr., at age 15? Who was the artist? When was it painted? ***8 points each***

In the Stanford Museum (room 7), painted by Felix Chary of Paris in 1884.

24. In what year was Margaret Jacks born? ***5 points***
What college did she attend? ***15 points***
What was her middle name? ***20 points***
What is the connection between her name and a popular food item? ***20 points***

Her portrait at the entrance to Margaret Jacks Hall says that she lived 1875-1962. She went to Radcliffe, according to the speech by Paul Hanna when the building was dedicated. (This reference can be found in Campus Report; it's cited under her name in the Stanford Archives.) According to the biography of her father David Jacks, by Arthur Bester, her middle name was Anna.. [This can also be found in her obituary, Calif. Hist. Quarterly 41 (1962), 367.] According to the Monterey Peninsula Herald, 10 Nov 1982, page 34, Monterey Jack cheese is named after her father, David Jacks, who produced the cheese on his Carmel Valley Dairy Ranch and distributed it throughout California under the name Jacks Monterey Cheese. [Webster's and other dictionaries are not aware of the derivation of this word, nor are standard reference books on cheese.]

25. What company was co-founded by Fletcher Jones? ***10 points***
What is his connection with our department? ***10 points***

He co-founded Computer Sciences Corporation about 1959; this was the first (or nearly first) software company. He established the Jones Foundation before his death in a private plane crash in 1972. That Foundation endowed the first chair in our department in 1977. [Reference: Endowed professorships at Stanford; or New York Times 9 Nov 1972, p. 50.]

26. What is the origin of the LISP terms 'car' and 'cdr'? ***10 points***

This terminology comes from the IBM 704. It means the 'contents of address field of register' (the rightmost 15 bits) and 'contents of decrement field of a register' (the rightmost 15 of the leftmost 18); a 'register' at that time meant a memory word. The original LISP software stored pointers there. McCarthy alludes to this origin in his fundamental paper "Recursive functions of symbolic expressions," CACM 3 (1960), p. 182. An explicit explanation appears in his paper on the history of LISP, in the book History of Programming Languages edited by Wexelblatt (1981), p. 175.

Scores:

	David S	Derrick B			Marcia. D	Tom F
	Martin R	Scott s			David J	Barry H
	Ramin Z	Jack S	Rohit C	Inderpal M	Liz W	Tom H
Problem	Carlos S	Michael W	Kelly R	Hakan J	Dave M	Alex W
1	45	44	50	30	53	61
2	40	30	0	34	20	40
3	20	20	10	20	25	20
4	20	30	10	0	20	20
5	10	8	8	8	8	10
6	12	10	0	0	13	13
7	43	51	44	52	46	54
8	10	10	3	5	15	10
9	20	19	0	0	20	20
10	85	100	60	45	150	195
11	50	0	25	50	55	80
12	35	34	25	34	35	34
13	0	9	3	11	11	9
14	35	36	31	26	36	45
15	45	45	34	35	46	45
16	15	15	15	15	15	15
-17	45	65	40	45	45	45
18	38	39	24	10	39	39
19	40	0	40	40	40	40
20	0	40	0	12	38	55
21	25	20	0	20	35	42
22	120	120	130	80	124	124
23	8	27	24	24	20	24
24	30	70	0	0	50	60
25	20	20	10	20	25	20
26	10	10	5	5	10	10
Totals	821	872	591	621	994	1130

Appendix B: Multigrades

(Most of this material appears in the book by Gloden [3].) We let $s_k = a_1^k + \cdot \, . \, . + a_m^k$ and $t_k = b_1^k + \cdot \cdot \cdot + b_m^k$. The notation

$$a_1, . . . ,\boldsymbol{a,} \overset{n}{=} b_1, . . . , b_n$$

means that $s_k = t_k$ for $0 \leq k \leq n$. With an asterisk after the n it means that $s_k = t_k$ for all even values of $\boldsymbol{k} \leq n$. Without the asterisk this is called a ***multigrade.*** It is called ***trivial*** if the ***b***'s are simply a permutation of the a's. It is called ***ideal*** if ***m*** = n + 1 and it is not trivial. We assume that the a's and ***b***'s are integers.

Lemma 1. $a_1, \ldots, a_m \overset{n}{=} b_1, \ldots, b_m$ *implies* $ca_1, \ldots, ca_m \overset{n}{=} cb_1, \ldots,$ ***cb,.***

Lemma *2.* ***al,. . . , a,*** $\overset{n}{=} b_1, . . . , b_m$ ***implies*** a_1 + ***d, . . . , a,*** + ***d*** $\overset{n}{=} b_1$ + ***d,. . . ,*** b_m + ***d.***

Lemma 3 [6]. $a_1, \ldots,$ ***a,*** $\overset{n}{=} b_1, \ldots, b_m$ *implies* $a_1, \ldots, a_m, b_1$ ***+d*** $, \ldots, b_m$ ***+d*** $\overset{n+1}{=} a_1 +$ $d, \ldots, a_m + d, b_1, \ldots, b_m$.

Example 1 [4]. Successive applications of Lemma 3 starting with '$0 \overset{0}{=} 1$':

d= 2: $0, 3 \overset{1}{=} 1, 2$.
d= 3: $0, 3, 4, 5 \overset{2}{=} 1, 2, 3, 6 \Rightarrow 0, 4, 5 \overset{2}{=} 1, 2, 6$.
d= 5: $0, 4, 5, 6, 7, 11 \overset{3}{=} 1, 2, 5, 6, 9, 10 \Rightarrow 0, 4, 7, 11 \overset{3}{=} 1, 2, 9, 10$.
d =-7: $0, 4, 7, 8, 9, 11, 16, 17 \overset{4}{=} 1, 2, 7, 9, 10, 11, 14, 18 \Rightarrow 0, 4, 8, 16, 17 \overset{4}{=} 1, 2, 10, 14, 18$.
d = 8: $0, 4, 8, 9, 10, 16, 17, 18, 22, 26 \overset{5}{=} 1, 2, 8, 10, 12, 14, 16, 18, 24, 25$
$\Rightarrow \quad 0, 4, 9, 17, 22, 26 \overset{5}{=} 1, 2, 12, 14, 24, 25$.
d = 1 3 : $0, 4, 9, 14, 15, 17, 22, 25, 26, 27, 37, 38 \overset{6}{=} 1, 2, 12, 13, 14, 17, 22, 24, 25, 30, 35, 39$
$\Rightarrow 0, 4, 9, 15, 26, 27, 37, 38 \overset{6}{=} 1, 2, 12, 13, 24, 30, 35, 39$.
d = 1 1 : $0, 4, 9, 12, 13, 15, 23, 24, 26, 27, 35, 37, 38, 41, 46, 50$
$\overset{7}{=} 1, 2, 11, 12, 13, 15, 20, 24, 26, 30, 35, 37, 38, 39, 48, 49$
$\Rightarrow 0, 4, 9, 23, 27, 41, 46, 50 \overset{7}{=} 1, 2, 11, 20, 30, 39, 48, 49$.

These are ideal except when n = *6.*

Example 2. Starting with the amazing sequence

$$0, 5, 7, 10, 17, 22, 24, 29 \overset{0}{=} 1, 6, 8, 13, 20, 23, 25, 30$$

and discarding duplicates we can obtain an ideal multigrade of order 6 by using various prime values of ***d.***

d = 5: $0, 7, 11, 17, 18, 24, 28, 35 \overset{1}{=} 1, 8, 12, 15, 20, 23, 27, 34$.
d = ʻi: $0, 11, 17, 19, 22, 28, 30, 41 \overset{2}{=} 1, 12, 14, 20, 23, 25, 31, 42$.
d = 11: $0, 17, 19, 34, 42, 53 \overset{3}{=} 1, 14, 20, 33, 39, 52$.
cl = 13: $0, 17, 19, 27, 34, 36, 46, 53, 65 \overset{4}{=} 1, 13, 20, 30, 32, 39, 47, 49, 66$.
cl = 17: $0, 18, 19, 27, 37, 46, 56, 64, 65, 83 \overset{5}{=} 1, 13, 20, 32, 39, 44, 51, 63, 70, 82$.
d = 19: $0, 18, 27, 58, 64, 89, 101 \overset{6}{=} 1, 13, 38, 44, 75, 84, 102$.

Lemma 4. We have $a_1, \ldots, a_m, -a_1, \ldots, -a_m \overset{n}{=} b_1, \ldots, b_m, -b_1, \ldots, -b_m$ if and only if $a_1, \ldots, a_m \overset{n*}{=} b_1, \ldots, b_m$.

Lemma 5. *If n is even and $d \neq 0$ we have $d + a_1, \ldots, d + a_m, d - b_1, \ldots, d - b_m \overset{n}{=} d - a_1, \ldots, d - a_m, d + b_1, \ldots, d + b_m$ if and only if $d + a_1, \ldots, d + a_m, d - b_1, \ldots, d - b_m \overset{n*}{=} d - a_1, \ldots, d - a_m, d + b_1, \ldots, d + b_m$.*

Proof. Let $(A_1, \ldots, A_M) = (d + a_1, \ldots, d - b_m)$ and $(B_1, \ldots, B_M) = (d - a_1, \ldots, d + b_m)$. We want to show that $S_k = T_k$ for all $k \leq n$ if this holds only for the even values of k. By the binomial theorem, we have $S_k = T_k$ for odd k if $s_l = t_l$ for all odd $l \leq k$. Also $S_2 - T_2 = 4d(s_1 - t_1)$, hence $s_1 = t_1$. This implies that $S_4 - T_4 = 8d(s_3 - t_3)$, hence $s_3 = t_3$; and this implies that $S_6 - T_6 = 12d(s_5 - t_5)$, etc.

Lemma 6 (Birck, see [3]). *If $a_1, a_2, a_3 \overset{4*}{=} b_1, b_2, b_3$ then $A_1, \ldots, A_7 \overset{8*}{=} B_1, \ldots, B_7$, where $(A_1, \ldots, A_7) = (s, s - a_1, s - a_2, s - a_3, b_1, b_2, b_3)$, $(B_1, \ldots, B_7) = (t, t - b_1, t - b_2, t - b_3, a_1, a_2, a_3)$, $s = (a_1 + a_2 + a_3)/2$, $t = (b_1 + b_2 + b_3)/2$.*

Proof. Let $f_n = s^n + (s - a_1)^n + (s - a_2)^n + (s - a_3)^n - a_1^n - a_2^n - a_3^n$ and let g_n be defined similarly from the b's. We must show that $f_{2k} = g_{2k}$ for $1 \leq k \leq 4$. A little algebra shows that $f_2 = 0$, $f_4 = \frac{3}{4}(s_2^2 - 2s_4)$, $f_6 = \frac{15}{16}s_2(s_2^2 - 2s_4)$, and $f_8 = \frac{7}{64}(s_2^2 - 2s_4)(7s_2^2 + 2s_4)$. The result follows since $s_2 = t_2$ and $s_4 = t_4$. Incidentally, the factor $s_2^2 - 2s_4$ equals $16s(s - a_1)(s - a_2)(s - a_3)$, so we obtain only trivial solutions if we try to make $f_4 = g_4 = 0$.

Example 3 (Létac, see [3]). If $a_1, a_2, a_3 \overset{4*}{=} b_1, b_2, b_3$ and $a_1 + a_2 = 3a_3$ and $b_1 + b_2 = 3b_3$, Lemma 6 tells us that $s, s - a_1, s - a_2, s - a_3, b_1, b_2, b_3 \overset{8*}{=} t, t - b_1, t - b_2, t - b_3, a_1, a_2, a_3$. But $s - a_3 = a_3$ and $t - b_3 = b_3$, so we can cancel them from both sides. Lemma 4 now yields

$$s, s - a_1, s - a_2, b_1, b_2, -s, a_1 - s, a_2 - s, -b_1, -b_2 \overset{9}{=}$$
$$t, t - b_1, t - b_2, a_1, a_2, -t, b_1 - t, b_2 - t, -a_1, -a_2.$$

This is an ideal multigrade of order 9, and none are known of orders > 9. To find appropriate values of the a's and b's we need to solve four equations in six unknowns. Standard techniques of Diophantine analysis show the existence of infinitely many solutions, one of which is

$$a_1 = 23750, \quad a_2 = 11857, \quad a_3 = 11869, \quad s = 23738;$$
$$b_1 = 20885, \quad b_2 = -20231, \quad b_3 = 218, \quad t = 436.$$

Example 4 [5]. If $a_1, a_2, a_3 \overset{4*}{=} b_1, b_2, b_3$ and $a_1 + a_2 + a_3 = 2(b_1 + b_2 + b_3) = 8h \neq 0$, we have $A_1, \ldots, A_{10} \overset{8}{=} -A_1, \ldots, -A_{10}$, where

$$(A_1, \ldots, A_{10}) = (a_1 - 3h, a_2 - 3h, a_3 - 3h, -3h, a_1 - h, a_2 - h, a_3 - h, h - b_1, h - b_2, h - b_3).$$

The proof is interesting: Lemma 6 tells us that

$$4h, 4h - a_1, 4h - a_2, 4h - a_3, b_1, b_2, b_3 \overset{8*}{=} 2h, 2h - b_1, 2h - b_2, 2h - b_3, a_1, a_2, a_3.$$

We can add redundant terms and change signs, to obtain

$$a_1 - 4h, a_2 - 4h, a_3 - 4h, -4h, a_1 - \mathbf{211}, a_2 - 2h, a_3 - 2h, -b_1, -b_2, -b_3$$

$$\overset{8*}{=} \mathbf{2h} - a_1, 2h - a_2, 2h - a_3, 2h, -a_1, -a_2, -a_3, b_1 - 2h, b_2 - 2h, b_3 - \mathbf{2h}.$$

Now Lemma 5 applies with ***d*** = ***-h,*** so we can remove the $*$. Adding ***h*** to each term (Lemma 2) yields the stated result. If we also choose the parameters so that $b_1 + b_2 = 3b_3$, the last term A_{10} is zero, hence we have an ideal multigrade of order 8! One solution is

$$a_1 = 71, \quad a_2 = 195, \quad a_3 = -98, \quad 12 = 21;$$
$$b_1 = -127, \quad b_2 = 190, \quad b_3 = 21;$$
$$(A_1, \ldots, A_9) = (8,132, -161, -63, 50, 174, -119, 148, -169).$$

Lemma 7. The multigrade $a_1, \ldots, a_m \overset{n}{=} b_1, \ldots, b_m$ is trivial ***if*** $m \le$ ***n.***

Proof. Add zeroes to both sides, if necessary, so that $m = n$. Then it is easy to prove that

$$(x - a_1) \ldots (x - a_m) = (x - b_1) \ldots (x - b_m).$$

Since a polynomial has unique roots, the ***b's*** are a permutation of the ***a's.***

Lemma 8. *If* $a_1, \ldots, a_m \overset{n}{=} b_1, \ldots, b_m$ ***is an ideal*** multigrade, ***the difference*** $a_1 \ldots a_m - b_1 \ldots b_m$ ***is*** a *nonzero* ***multiple of n!,*** and ***it equals*** $(s_m - t_m)/m$.

Proof. $\sum \binom{a_i}{m} - \sum \binom{b_i}{m} = (s_m - t_m)/m!$, since $m = 12 + 1$. If this is zero, the ***a's*** are a permutation of the ***b's*** by Lemma 7. Hence $s_m - t_m$ is a nonzero multiple of m!. To complete the proof, use the fact that $a_1 \ldots a_m = s_m/t_m + f(s_1, \ldots, s_n)$ and $b_1 \ldots b_m = t_m/m + f(t_1, \ldots, t_n)$, where f is a certain polynomial.

Lemma 9. ***If*** $a_1, \ldots, a_m \overset{n}{=} b_1, \ldots, b_m$ ***is*** *an ideal multigrade,* ***the n*** + 1 ***products (b;*** $- a_1) \ldots (b_i - a_m)$ are all equal ***to*** $(-1)^m (b_1 \ldots b_m - a_1 \ldots a_m)$.

Proof. We have $(x - a_1) \ldots (x - a_m) + (-1)^m a_1 \ldots a_m = (x - b_1) \ldots (x - b_m) + (-1)^m b_1 \ldots b_m$.

Example 5. Lemma 9 leads to almost unbelievable factorization patterns. For example, A. Létac has found the multigrade

$$99, -75, -69, -16, -13, 34, 58, 82, 98 \overset{8}{=} 99, 75, 69, 16, 13, -34, -58, -82, -98$$

by a method different from that of Example 4. Here is the matrix whose entries are the -prime factorizations of $b_i - a_j$:

$2 \cdot 3^2 \cdot 11$	$2 \cdot 3 \cdot 29$	$2^3 \cdot 3 \cdot 7$	5.23	$2^4 \cdot 7$	$5 \cdot 13$	41	17	1
$2 \cdot 3 \cdot 29$	$2 \cdot 3 \cdot 5^2$	$2^4 \cdot 3^2$	$7 \cdot 13$	$2^3 \cdot 11$	41	17	- 7	-23
$2^3 \cdot 3 \cdot 7$	24.32	$2 \cdot 3 \cdot 23$	$5 \cdot 17$	2.41	$5 \cdot 7$	11	-13	-29
$5 \cdot 23$	$7 \cdot 13$	$5 \cdot 17$	2^5	29	$-2 \cdot 3^2$	-2-3-i	$-2 \cdot 3 \cdot 11$	-2.41
$2^4 \cdot 7$	$2^3 \cdot 11$	2.41	29	2.13	-3.7	$-3^2 \cdot 5$	-3.23	$-5 \cdot 17$
5.13	41	$5 \cdot 7$	$-2 \cdot 3^2$	-3.7	$-2^2 \cdot 17$	$-2^2 \cdot 23$	$-2^2 \cdot 29$	$-2^2 \cdot 3 \cdot 11$
41	17	11	-2.3.7	-32.5	$-2^2 \cdot 23$	$-2^2 \cdot 29$	-22.5.7	$-2^2 \cdot 3 \cdot 13$
17	- i	-13	$-2 \cdot 3 \cdot 11$	-3.23	-22.29	$-2^2 \cdot 5 \cdot 7$	$-2^2 \cdot 41$	$-2^2 \cdot 3^2 \cdot 15$
1	-23	-29		$-5 \cdot 1$ i	$-2^2 \cdot 3 \cdot 11$	$-2^2 \cdot 3 \cdot 13$	$-2^2 \cdot 3^2 \cdot 5$	$-2^2 \cdot 7^2$

The product of each row is constant. Also, each entry of the $(i + 1)$st row is obtained by subtracting a constant c_i from the entry above it. For example, all entries of the second row are 24 less than the entries of the first row; yet the row products are the same!

Theorem 1. Every multigrade $a_1, \ldots$, a, $\overset{n}{=} b_1, \ldots, b_m$ can be obtained *by* starting *with* a multigrade ***of order zero*** and ***using*** Lemma. 3 *exactly* ***n*** times, ***with d = 1 each time.***

Proof. We may assume that $a_1, \ldots, a_m, b_1, \ldots, b_m$ are nonnegative, since it's possible to get the multigrade $a_1 - d, \ldots, a_m - d \overset{n}{=} b_1 - d, \ldots, b_m - d$ by reducing all numbers of the initial zero-order multigrade by ***d.***

Two sequences $(a_1, \ldots, a_l)$ and $b_1, \ldots, b_m)$ can be represented by the polynomial $x^{a_1} + \cdots x^{a_l} - x^{b_1} - \cdots - x^{b_m}$. Conversely, any polynomial with integer coefficients corresponds to a unique pair of sequences (al,. . . , a_l) and $(b_1, \ldots, b_m)$ in this way.

Let f(x) be the polynomial corresponding to $(a_1, \ldots, a_l)$ and $(b_1, \ldots, b_m)$. We have $l = m$ if and only if $f(1) = 0$. Furthermore we have $a_1, \ldots, a_m \overset{n}{=} b_1, \ldots, b_m$ if and only if $f'(1) = \cdots = f^{(n)}(1) = 0$. Thus, $a_1, \ldots$, a, $\overset{n}{=} b_1, \ldots, b_m$ if and only if the corresponding polynomial is a multiple of $(5 - 1)^{n+1}$. (This observation is due to Escott, see [1].)

The transformation of Lemma 3 corresponds to multiplying the polynomial by $x^d - 1$. Therefore we can deduce al,. . . , a, $\overset{n}{=} b_1, \ldots, b_n$ if we start with the zero-order multigrade that corresponds to $f(x)/(x - 1)$" and apply Lemma 3 ***n*** times until ***d*** = 1.

Theorem 2. ***Let al,. . . ,a,*** $\overset{n}{=} b_1, \ldots, b_m$ ***be*** an ideal multigrade and ***let*** $d_1, \ldots, d_n$ ***be*** any ***sequence of*** distinct ***primes*** that divide ***at least one of the differences*** $b_i - a_j$. ***This*** multigrade can be obtained from a. multigrade ***of*** order zero ***by*** applying Lemma. 3 *with* $d = d_1, \ldots, d_n$ respectively.

Proof. Refer to the proof of Theorem 1. If ***p*** is a prime divisor of $b_i - a_j$, ***we*** have $(x - a_1) \ldots (x - a_m) ==$ *(x* $-$ $b_1) \ldots (x -$ ***b,)*** modulo ***p,*** by Lemma 9, hence we can permute the ***b's*** so that $b_i - a_i$ is a multiple of ***p*** for all i. Therefore $x^{a_i} - x^{b_i}$ is a multiple of $x^p - 1$; therefore *f(x)* is a multiple of $x^p - 1$. The polynomial $x^{d_i} - 1$ has no factor in common with $x^{d_j} - 1$ when $i \neq j$, except x $-$ 1; and we know that *f(x)* is divisible by $(x - 1)^{n+1}$. Hence *f(x)* is a multiple of $(x^{d_1} - 1) \ldots (x^{d_n} - 1)(x - 1)$. We can start with the zero-order multigrade corresponding to $f(x)/(x^{d_1} - 1) \ldots (x^{d_n} - 1)$.

Example 6. Add 100 to each element of Létac's order-8 multigrade in Example 5. This can be obtained by starting with

$$1, 6, 8, 11, 21, 34, 45, 47, 52 \overset{0}{=} 2, 7, 9, 20, 33, 43, 46, 48, 53$$

and applying Lemma 3 with ***d*** = $5, 7, 11, 13, 17, 23, 29, 41$.

Example 7 (Ron Graham). Let $a_1, \ldots, a_9 \overset{8}{=}$ *-al,...,* $-a_9$ be Létac's multigrade of Example 5. The sum $\sqrt{n + a_1} + \cdots + \sqrt{n + a_9}$ is equal to the sum $\sqrt{n - a_1} + \cdots + \sqrt{n - a_9}$ plus $0(n^{-9})$. For example, when n = 1000 the respective sums are

284.44073149798 ancl 284.44073149799.

When $n = 1000000$, they axe

$$8999.99999481749998825924729336515537480\,89$$
$$\text{and}\quad 8999.99999481749998825924729336515537480\,93,$$

respectively. This illustrates how difficult it is in general to compare two given sums of square roots of integers.

Example 8 (A. Moessner). Here are the shortest known multigrades of order 10 and 11:

$$1,13,25,55,75,87,95,-7,\ -11,-19,-61,\ -69,\ -91,\ -93 \overset{10}{=} -a_1,\ldots,-a_{14};$$
$$1,28,31,32,55,61,68,-a_1,\ldots,-a_7 \overset{11}{=} 17,20,23,44,49,64,67,-b_1,\ldots,-b_7.$$

Theorem 3 [7]. *For all n there exists a nontrivial multigrade $a_1,\ldots,a_m \overset{n}{=} b_1,\ldots,b_m$ such that $m \le \frac{1}{2}n^2 + \frac{1}{2}n + 1$.*

Proof. Let $m = \frac{1}{2}n^2 + \frac{1}{2}n + 1$ and consider the N^m sequences $(a_1,\ldots,a_m)$ where $0 \le a_i < N$. Each sequence $(a_1,\ldots,a_m)$ defines a sequence of power sums $(s_1,\ldots,s_n)$. Since $0 \le s_k < mN^k$, there are at most $(mN^1)\ldots(mN^n) = m^nN^{m-1}$ distinct sequences of power sums. If we let $N = m!\,m^n + 1$ we have $N^m/m! > m^nN^{m-1}$; hence there must be two $(a_1,\ldots,a_m)$ with the same power sums that are not permutations of each other.

References

[1] H. L. Dorwart and 0. E. Brown, "The Tarry-Escott Problem," American Math. Monthly **44** (1937), 613-626. See also the related paper by Jack Chernick on pp. 626-633.

[2] Escott, Quarterly *J.* Math. **41** (1910), 152.

[3] Albert Gloden, *Mehrgradige* Gleichungen (Groningen: Noordhoff, 1944).

[4] Hardy and Wright, Theory of Numbers, 521.10.

[5] A. Létac, *Gazeta* Math. **48** (1942), 68-69.

[6] Tarry, *L'Intermédiare* des *math.* **19** (1912), 219-221; **20** (1913), 68-70.

[7] Wright, Bull. Amer. *Math. Soc.* **54** (1948), 755-757.

Appendix C: The L^3 Algorithm

Let $a_1, \ldots, a_m$ be linearly independent n-dimensional vectors of integers. We wish to find a linear combination x = $x_1 a_1 + \ldots + x_m a_m$, whose length $\|x\|^2$ is short, where the coefficients x_i are integers. The algorithm sketched below, due to Lenstra, Lovász, and Lenstra [Math. *Annalen* 261 (1982), 516–524], is surprisingly effective for this problem.

The algorithm makes use of integer n-vectors $b_1, \ldots, b_m$, rational n-vectors $b_1^*, \ldots, b_m^*$, and rational m-vectors $\mu_1, \ldots, \mu_m$. We write b_{ij} for the j-th component of b_i, etc. It proves to be handy to maintain the rational numbers $\| b_i^* \|^2, \ldots, \| b_m^* \|^2$ as separate, redundant variables. An integer variable k records the algorithm's progress towards a solution.

The data structures satisfy the following invariant relations at key points during the execution of the algorithm:

(1) Any integer combination of $(a_1, \ldots, a_m)$ is an integer combination of $(b_1, \ldots, b_m)$ and conversely.

(2) $b_i^* \cdot b_j^* = 0$ if $i \neq j$; $b_i^* \cdot b_i^* = \|b_i^*\|^2$. (This is the dot product of vectors.)

(3) $b_i = \sum 1 \leq j \leq m \mu_{ij} b_j^*$.

(4) $\mu_{ii} = 1$, and $\mu_{ij} = 0$ for $j > i$.

(5) $|\mu_{ij}| \leq 1/2$, for $j < i \leq k$.

(6) $\|b_{i+1}^*\|^2 \geq (1 - \mu_{(i+1)i}^2)\|b_i^*\|^2$, for $1 \leq i < k$.

It follows from (2) and (3) that $\mu_{ij} = (b_i \cdot b_j^*)/\|b_j^*\|^2$. We don't need to compute the values of μ_{ij} for $j \geq i$, since they are constant.

We can establish these conditions initially by setting $k \leftarrow 1$ and doing the following operations for $i \leftarrow 1, \ldots, m$:

$$b_i \leftarrow a_i; \quad b_i^* \leftarrow b_i;$$

$$\mu_{ij} \leftarrow (b_i \cdot b_j^*)/\|b_j^*\|^2 \text{ and } b_i^* \leftarrow b_i^* - \mu_{ij} b_j \text{ for } j = 1, \ldots, i-1;$$

$$\|b_i^*\|^2 = b_i^* \cdot b_i^*.$$

(This is called the 'Gram-Schmidt orthogonalization algorithm'.)

There is a subroutine 'fix(i, j)' which ensures that condition (5) holds at a particular position i, j:

If $|\mu_{ij}| > 1/2$, let r be an integer such that $|\mu_{ij} - r| \leq 1/2$.

Replace b_i by $b_i - r b_j$ and μ_i by $\mu_i - r\mu_j$.

(Thus, $\mu_{il} \leftarrow \mu_{il} - r\mu_{jl}$ for $1 \leq 1 < j$, and $\mu_{ij} \leftarrow \mu_{ij} - r$.)

We will prove below that the vectors $b_1, \ldots, b_m$ will be rather short, if conditions (1) ... (6) hold when $k = m$. Conditions (5) and (6) hold trivially when $k = 1$. Therefore the idea of the algorithm is to try to increase k, and it does this by repeating the following steps until $k = m$:

```
fix($k+1, k$);
if $\|b^*_{k+1}\|^2 \geq (1 - \mu^2_{(k+1)k})\|b^*_k\|^2$
then set $k \leftarrow k + 1$ and fix($k, j$) for $j = k-2, \ldots, 1$
else interchange vectors $b_k$ and $b_{k+1}$, then recompute the auxiliary variables as follows:
    $\mu \leftarrow \mu_{(k+1)k}$; $x^* \leftarrow b^*_k$; $\|x^*\|^2 \leftarrow \|b^*_k\|^2$;
    $b^*_k \leftarrow \mu x^* + b^*_{k+1}$; $\|b^*_k\|^2 \leftarrow \mu^2\|x^*\|^2 + \|b^*_{k+1}\|^2$; $\mu_{(k+1)k} \leftarrow \mu\|x^*\|^2/\|b^*_k\|^2$;
    $b^*_{k+1} \leftarrow x^* - \mu_{(k+1)k} bk^*$; $\|b^*_{k+1}\|^2 \leftarrow \|b^{*2}_{k+1}\| \|x^*\|^2/\|b^*_k\|^2$;
    $(\mu_{kj}, \mu_{(k+1)j}) \leftarrow (\mu_{(k+1)j}, \mu_{kj})$ for $1 \leq j < k$;
    $(\mu_{ik}, \mu_{i(k+1)}) \leftarrow (\mu_{ik} * \mu_{(k+1)k} + \mu_{i(k+1)}\|b^*_{k+1}\|^2/\|x^*\|^2, \mu_{ik} - \mu\, \mu_{i(k+1)})$
        for; $k+1 < i \leq m$;
    if k > l then k + k - 1 .
```

The algorithm terminates because the product $\|b^*_1\|^2 \ldots \|b^*_k\|^2$ decreases whenever k decreases and stays the same when k increases. This product is always a positive integer; indeed, it is the determinant of BB^T, when B is the matrix whose rows are $(b_1, \ldots, b_k)$.

-Examination of the algorithm shows that the vectors b^*_i don't really need to be maintained! All we need is the set of squared lengths $\| b^*_i \|^2$, once the initialization has been done.

When the algorithm terminates we can show that the vectors b_i are not too far from the shortest possible. Condition (6) says that

$$\|b^*_i\|" \leq \tfrac{4}{3}\|b^*_{i+1}\|^2 \qquad \text{for } 1 \leq i < m$$

because of condition (5), hence

$$\|b^*_j\|^2 \leq \left(\tfrac{4}{3}\right)^{i-j}\|b^*_i\|^2 \qquad \text{for } 1 \leq j < i.$$

Therefore we have

$$\|b_i\|^2 = \|b^*_i\|^2 + \sum 1 \leq j < i\ \mu^2_{ij}\|b^*_j\|^2$$
$$\leq \|b^*_i\|^2 + \sum 1 \leq j < i\ \tfrac{1}{4}\left(\tfrac{4}{3}\right)^{i-j}\|b^*_i\|^2 = \left(\tfrac{4}{3}\right)^{i-1}\|b^*_i\|^2,$$

Now let $(x_1, \ldots, x_l)$ be any linearly independent integer multiples of $(a_1, \ldots, a,)$. We will prove that there's a way to permute the x's so that

$$\|b_i\|^2 \leq \left(\tfrac{4}{3}\right)^{m-1}\|x_i\|^2 \qquad \text{for } 1 \leq i \leq l. \qquad (*)$$

Hence it's impossible to find x's that are much smaller than the b's. (In practice, the b's are in fact much shorter than this worst-case bound would indicate.)

To prove $(*)$, we write $x_i = \sum r_{ij} b_j$ and let $q(i)$ be the maximum j such that $r_{ij} \neq 0$. We can permute the x's so that $q(1) \le \cdots \le q(2)$. Now we must have $q(i) \ge i$, for otherwise the vectors $x_1, \ldots, x_i$ would all be linear combinations of $(b_1, \ldots, b_{i-1})$ and they couldn't be linearly independent. By the inequalities above we have

$$\|b_i\|^2 \le \left(\tfrac{4}{3}\right)^{i-1} \|b_i^*\|^2 \le \left(\tfrac{4}{3}\right)^{q(i)-1} \|b_{q(i)}^*\|^2 .$$

But $\|xi\|^2 \ge \|b_{q(i)}^*\|^2$, because $x_i = r_{iq(i)} b_{q(i)}^* + \sum 1 \le j < q(i) r'_{ij} b_j^*$ for some real numbers r'_{ij}, and because $r_{iq(i)}$ is a nonzero integer.

The original L^3 paper described a slightly different algorithm in which the factor $(1 - \mu_{(i+1)i}^2)$ in (6) was replaced by $\left(\frac{3}{4} - \mu_{(i+1)i}^2\right)$. In this case it's possible to prove termination in polynomial time, but the bound corresponding to $(*)$ has 2 in place of $\frac{4}{3}$. Experiments by Lagarias and Odlyzko [*JACM* **32** (1985), 240–241] indicate that the algorithm above produces shorter vectors and takes only about 3 times as long.

bool := #: *atom.* { there's a special kind of atom called a *bool*)
proof (b : bool) := #: *atom.* { and another called a *proof* of a *bool* ; logicians say $\vdash$ *b* }

for_all(t : atom; p: (x : t)bool) := #: *bool.* { $(\forall x)p(x)$ is a *bool* }
generalize (t : atom; p: (x : t)bool; q : (x : t)proof (p(x))) := #: *proof (for_all(t, p)).*
    { if we can prove $p(x)$ given x, then $\vdash (\forall x)p(x)$)
specialize (t : atom; p: (x : t)bool; x : t; q : proof (for_all(t, p))) := #: *proof (p(x)).*
    { and conversely }

exists(t : atom; p: (x : t)bool) := #: *bool.* { $(\exists x)p(x)$ is a *bool* }
existence(t : atom; p: (x : t)bool; x : t; q : proof (p(x))) := #: *proof (exists(t,p)).*
    { if we can prove *p(x)* for some x, then $\vdash (\exists x)p(x)$ }
choose (t : atom; p: (x : t)bool; q : proof (exists (t,p))) := #: *t.*
    { if we can prove $(\exists x)p($ x), *choose* gives us an x such that $\vdash$ *p(x)*)
thus(t : atom; p : (x : t)bool; q : proof (exists(t,p))) := #: *proof (p(choose(t,p, q))).* *{see?)*

trick(a, b : bool; p : proof (a)) := *b: bool.* { maps a proof of *a* into *b*)

implies (a, b : bool) := *for_all(proof (a), trick (a, b)): bool .* { $a \Rightarrow b$ }
implication(a, b: bool; p : (q : proof (a))proof (b)) := *generalize(proof (a), trick(a, b), p):*
    proof (implies (a, b)). { ifwe can prove *b*, given a proof of *a*, then $\vdash$ ***a*** $\Rightarrow$ *b*)
modus-ponens (a, b: bool; q : proof (implies (a, b)); p: proof (a)) :=
    specialize (proof (a), trick(a, b), p, q): proof(b). { $\vdash$ *a.* $\Rightarrow$ *b* and $\vdash$ *a* yields $\vdash$ *b)*

∧(a, b : bool) := *exists(proof (a), trick(a, b)): bool.* { *a A b* }
conjunction(a, b : bool; p: proof (a); q : proof (b)) := *existence (proof (a), trick(a, b), p, q):*
    proof (∧(a, b)). { $\vdash$ *a* and $\vdash$ *b* yields $\vdash$ *a A b* }
first-conjunct (a, b: bool; p : proof (∧(a, b))) := *choose (proof (a), trick(a, b), p): proof (a).*
    { $\vdash$ a **A** *b* yields $\vdash$ a)
second-conjunct (a, b: bool; p: proof (∧(a, b))) := *thus (proof (a), trick (a, b), p): proof(b).*
    { $\vdash$ *a* **A** *b* yields $\vdash$ *b*)

imp_refl (a : bool) := *implication(a, a, (q : proof (a))q): proof (implies (a, a,)).* { $\vdash a \Rightarrow a$ }
imp_trans (a, b, c : bool; p : proof (implies (a, b)); q : proof (implies (b, c))) :=
    implication (a, c, (r : proof (a))modus-ponens (b, c, q, modus-ponens (a, b, p, r))):
    proof (implies (a, c)). { if $\vdash a \Rightarrow b$ and $\vdash b \Rightarrow c$ then $\vdash a \Rightarrow c$ }
and_symm (a, b : bool; p : proof (∧(a, b))) :=
    conjunction (b, a, second-conjunct (a, b, p), first-conjunct (a, b, p)): proof (∧(b, a)).
    { if $\vdash$ *a* **A** *b* then $\vdash$ *b* **A** *a* }

lemma1 (a, b: bool) :=
    implication (∧(a,b), ∧(b, a.), and_symm(a, b)): proof (implies (∧(a, b), ∧(b, a))).
    { $\vdash$ *a* **A** *b* $\Rightarrow$ *b* **A** *a* }

Appendix E: CHEX3

A CHEX script about relations and sets.

This example of CHEX is intended to follow the general definitions in the 'second CHEX script'. Let's imagine that all those definitions have already been given.

1. First we introduce a type called '*elt*' on which relations and sets will be defined. Everything that follows could be made more general by including '*elt: atom*' as the first parameter to every function, but that would make this script unnecessary verbose.

elt := #: *atom*. { there's a special kind of *atom* called an *elt* }
rel := *(x, y : elt)bool*: *(x, y : elt)atom*. { a relation maps a pair of *elt*s to a *bool* }

2. Various properties of relations ase fundamental in mathematics. The three most important properties are defined here: A relation might be reflexive, symmetric, and/or transitive.

In the comments, we write 'x R y' if $r(x, y)$ holds.

rpred$(r : rel;\ x : elt) := r(x, x)$: *bool*. $\{x \mathrm{R} x\}$
reflexive $(r : rel) :=$ *for_all*(*elt*, *rpred* (r)): *bool*. $\{\forall x\ (x \mathrm{R} x)\}$
use_reflexivity$(x : elt;\ r : rel;\ p$: *proof* (*reflexive* (r))) :=
 specialize(*elt*, *rpred* $(r), x, p)$: *proof* $(r(x, x))$.

spred2 $(r : rel;\ x, y : elt) :=$ *implies*$(r(x,y), r(y,x))$: *bool*. $\{x \mathrm{R} y \Rightarrow y \mathrm{R} x\}$
spred1 $(r : rel;\ x : elt) :=$ *for-all* (*elt*, *spred2* (r, x)): *bool*. $\{\forall y$ *spred* $(r, x, y)\}$
symmetric$(r : rel) :=$ *for_all*(*elt* , *spred1* (r)): *bool*. $\{\forall x \forall y$ *spred* $(r, x, y)\}$
use_symmetry$(x, y : elt;\ r : rel;\ p$: *proof* (*symmetric*(r)); q : *proof* $(r(x, y)))$:=
 modus-ponens $(r(x,y), r(y, x)$, *specialize* (*elt* , *spred2* $(r, x), y$, *specialize* (*elt* , *spred1* $(r), x, p)))$: *proof* $(r(y, x))$.

tpred3 $(r : rel;\ x, y, z : elt) :=$ *implies* $(\wedge(r(x, y), r(y, z)), r(x, z))$: *bool*.
 $\{x \mathrm{R} y \wedge y \mathrm{R} z \Rightarrow x \mathrm{R} z\}$
tpred2 $(r : rel;\ x, y : elt) :=$ *for_all*(*elt*, *tpred3* (r, x, y)): *bool*. $\{\forall z \ldots\}$
tpred1 $(r : rel;\ x : elt) :=$ *for_all*(*elt* , *tpred2* (r, x)): *bool*. $\{\forall y \forall z \ldots\}$.
transitive $(r : rel) :=$ *for_all*(*elt* , *tpred1* (r)): *bool*. $\{\forall x \forall y \forall z \ldots\}$
use-transitivity $(x, y, z : elt;\ r : rel;\ p$: *proof* (*transitive* (r)); $p1$: *proof* $(r(x, y))$;
 $p2$: *proof* $(r(y, z))) :=$ *modus_ponens*$(\wedge(r(x, y), r(y, z)), r(x, z)$, *specialize* (*elt*, *tpred3* $(r, x, y), z$, *specialize*(*elt*, *tpred2* $(r, x), y$, *specialize*(*elt*, *tpred1* $(r), x, p)))$,
 conjunction $(r(x, y), r(y, z), p1, p2))$: *proof*$(r(x, z))$.

3. An equivalence relation is reflexive, symmetric, and transitive.

rs_rel$(r : rel) := \wedge($ reflexive (r), symmetric *(r)): **bool** .* { reflexive and symmetric }

equiv_rel$(r$ *: Tel) :=* $\wedge($ *rs_rel (r), **transitive*** (r)): ***bool*** . { and transitive too }

refl_equiv *(r : Tel; p : proof (equiv-rel (r))) :=*

*first-conjunct (reflexive (r), symmetric(r), first-conjunct (**rs_rel**(r), transitive (r), p)): proof (**reflexive** (r)).*

*s ymm-equiv (I : Tel; p : **proof** (equiv-Tel (r))) :=*

*second-conjunct (reflexive(r), symmetric(r), **first_conjunct**(rs_rel(r), transitive (r), p)): proof (symmetric(r)).*

trans_equiv *(r : Tel ; p : proof (equiv-Tel (r))) := second-conjunct(Ts-Tel(r), transitive(r), p): proof (**transitive** (r)).*

4. Next we define the notion of a set, which is sort of a one-dimensional relation. In the comments we write '$x \in s$' if s(x) holds.

*set := (x: **elt**)**bool**: (x : **elt**)**atom**.*

in(x : elt; s: set) := proof (s(x)): atom. { $\vdash$ *x* $\in$ *S}*

5. The most basic relation on sets is that of set inclusion, written '$\subseteq$'.

incl_pred *(s, t* : set; *x* : ***elt***) := ***implies***(*s*(*x*), *t*(*x*)): *bool.* { *x ES* $\Rightarrow$ *x Et* }

set_incl(*s*, *t* : *set*) := ***for_all***(***elt***, ***incl_pred*** (*s*, *t*)): *bool.* (∀x . . . }

is_incl *(s, t : set) := proof* (***set_incl***(*s*, *t*)): *atom.* { $\vdash$ *s* $\subseteq$ *t)*

*set-move (s, t : set; p : **is_incl**(s, t); x : elt; q : **in**(x, s)) :=*

*modus-ponens (s(x), t(x),s **pecialize** (**elt**, **incl_pred**(s, t), x, p), q): in (x, t).*

{ if x $\in$ *s* aid s $\subseteq$ *t* then x $\in$ *t* }

6. Set inclusion is reflexive and transitive.

set_incl_refl(*s* : set) := ***generalize***(***elt***, ***incl_pred***(*s*, *s*), (x : ***elt***)***imp_refl***(*s*(*x*))): ***is_incl***(*s*, *s*).

{ $\vdash$ *s* $\subseteq$ *s* }

*step1 (s, t, u : set; p : **is_incl**(s, t); q : **is_incl**(t, u); x : elt) :=*

*implication (s(x), u(x), (r : in(x, s))**set_move**(t, u, q, x, set-move (s, t, p, x, r))): proof (**incl_pred**(s, u, x)).* {if $x \in s$ and $s \subseteq t$ and $t \subseteq u$ then $x \in u$ }

set_incl_trans(*s*, *t*, *u* : ***set***; *p* : ***is_incl***(*s*, *t*); *q* : ***is_incl***(*t*, *u*)) :=

*generalize (elt , **incl_pred** (s, u), **step1** (s, t, u, p, q)): **is_incl**(s, u).*

{ if $s \subseteq t$ and $t \subseteq u$ then $s \subseteq u$ }

7. Another basic relation on sets is the notion of set equality.

set-eq(s, t : set) := $\wedge$(***set_incl***(*s*, *t*), ***set_incl*** *(t, s)): **bool**.* { s $\subseteq$ *t* and $\subseteq$ s }

is_eq(*s*, *t : set) := proof* (***set_eq***(*s*, *t)): atom,.* { $\vdash$ *s = t* }

8. Set equality is reflexive, symmetric, and transitive.

set_eq_refl (s : set) :=

conjunction (set_incl(s, s), set_incl(s, s), set_incl_refl(s), set_incl_refl(s)): is_eq(s, s).

$\{\vdash s = s\}$

set_eq_symm(s, t : set; p: is_eq(s, t)) :=

and_symm(set_incl(s, t), set_incl(t, s), p): is-eq(t, s). { if $s = t$ then $t = s$ }

step1 (s, t, u : set; p : is_eq(s, t); q : is_eq(t, u)) :=

set_incl_trans (s, t, u, first-conjunct (set-incl (s, t), set-incl(t, s), p),

first-conjunct (set-incl (t, u), set-id (u, t), q)): is-incl (s, u).

{if $s = t$ and $t = u$ then $s \subseteq u$}

step2 (s, t, u : set; p: is_eq(s, t); q : is_eq(t, u)) :=

step1 (u, t, s, set_eq_symm(t, u, q), set_eq_symm(s, t, p)): is_incl(u, s).

{if $s = t$ and $t = u$ then $u \subseteq s$}

set_eq_trans (s, t, u : set; p: is_eq(s, t); q : is_eq(t, u)) :=

conjunction (set-incl (s, u), set-incl(u, s), step1 (s, t, u, p, q), step2 (s, t, u, p, q)):

is-eq (s, 26). {if $s = t$ and $t = u$ then $s = u$}

9. Now we will study the properties of an arbitrary equivalence relation. For simplicity, we call it 'E'. (For generality, we could have added two parameters, '*E: rel*; *assumption: proof (eq-Tel (E))*', to each of the functions that follow.)

E := #: rel. { E is a relation }

assumption := #: proof (equiv_rel (E)). { E is an equivalence relation }

I(x, y: elt) := proof (E(x, y)): atom. $\{\vdash x \mathrm{E} y\}$

E_refl(x : elt) := use_reflexivity(x, E, refl_equiv(E, assumption)): I(x, x). $\{\vdash x \mathrm{E} x\}$

E_symm(x, y : elt; p: I(x, y)) := use_symmetry(x, y, E, symm_equiv (E, assumption), p):

I(y, x). {if $x \mathrm{E} y$ then $y \mathrm{E} x$}

E_trans(x, y , z : elt; p: I(x, y); q : I(y, z)) :=

use-transitivity (x, y, z, E, trans_equiv (E, assumption), p, q): I(x, z).

{if $x \mathrm{E} y$ and $y \mathrm{E} z$ then $x \mathrm{E} z$}

10. If x is an element then *E(x)* is the "equivalence class" of x. The rest of this script is devoted to a study of the elementary properties of equivalence classes.

In the comments we shall write '$[x]$' for x's equivalence class. It is easy to prove that $x \in [x]$:

proposition0 (x : elt) := E_refl(x): in(x, E(x)).

11. Next, we prove that x E y implies [x] = [y].

step1 $(x, y, z : elt; p : I(x, y); q : I(x, z))$:= *E_trans* $(y, x, z, $ *E_symm* $(x, y, p), q)$: $I(y, z)$.
{if $x \mathrm{E} y$ and $x \mathrm{E} z$ then y Ez}

step2 $(x, y : elt; p: I(x, y); z : elt)$:=
implication $(E(x, z), E(y, z),$ *step1* $(x, y, z, p))$: *proof* (*implies* $(E(x, z), E(y, z))$).
{if $x \mathrm{E} y$ then $x \mathrm{E} z \Rightarrow y \mathrm{E} z$}

lemma1 $(x, y : elt; p: I(x, y))$:= *generalize*(*elt*, *incl_pred* $(E(x), E(y))$, *step2* (x, y, p)):
is_incl$(E(x), E(y))$. { if x E y then [x] $\subseteq$ [y])

proposition1 $(x, y : elt ; p: I(x, y))$:= *conjunction* (*set-incl* $(E(x), E(y))$, *set-incl* $(E(y), E(x))$, *lemma1* (x, y, p), *lemma1* $(y, x,$ *E-symm*$(x, y, p)))$: *is_eq*$(E(x), E(y))$.
{ if x E y then [x] = [y])

12. Conversely, if [x] = [y] , we must have x E y .

step1 $(x, y : elt; p:$ *is_eq*$(E(x), E(y)))$:=
first_conjunct(*set_incl*$(E(x), E(y))$, *set_incl*$(E(y), E(x)), p)$: *is-incl*$(E(x), E(y))$.
{ if $[x] = [y]$ then $[x] \subseteq [y]$ }

step2 $(x, y : elt;$
$p :$ *is_eq*$(E(x), E(y)))$:= *specialize* (*elt*, *incl_pred* $(E(x), E(y))$, x, *step1* $(x, y, p))$:
proof (*implies* $(E(x, x), E(y, x))$). {if $[x] = [y]$ then $x \mathrm{E} x \Rightarrow y \mathrm{E} x$}

proposition2 $(x, y : elt ; p :$ *is_eq* $(E(x), E(y)))$:=
E_symm$(y, x,$ *modus-ponens* $(E(x, x), E(y, x),$ *step2* $(x, y, p),$ *E_refl*$(x)))$: $I(x, y)$.
{ if $[x] = [y]$ then $x \mathrm{E} y$ }

13. An equivalence class is a set that equals [x] for some x.

Eclass $(s : set; x : elt)$:= *set-eq*$(s, E(x))$: *bool*. { $s = [x]$)
is_Eclass $(s : set)$:= *proof* (*exists* (*elt* , *Eclass* (s))): *atom*. { $\exists x (s = [x])$)

14. Each equivalence class s therefore has a "representative" element x, called its *Erep* and denoted $\hat{s}$.

Erep $(s : set; p :$ *is-Eclass* $(s))$:= *choose* (*elt* , *Eclass* (s), p): *elt* . { $\hat{s}$ is an element)
IErep$(s : set; p:$ *is_Eclass*$(s))$:= *thus*(*elt*, *Eclass*$(s), p)$: *is_eq*$(s, E(Erep(s,p)))$.
{ $\vdash s = [\hat{s}]$ }

15. If s is an equivalence class and x $\in$ s, we have $x \mathrm{E} \hat{s}$.

step1 $(s : set ; p :$ *is_Eclass* $(s))$:=
first_conjunct(*set_incl*$(s, E(Erep(s, p)))$, *set_incl*$(E(Erep(s, p)), s)$, *IErep*$(s, p))$:
is_incl$(s, E(Erep(s, p)))$. { $\vdash s \subseteq [\hat{s}]$)

proposition9 $(s : set; p :$ *is_Eclass* $(s); x : elt ; q :$ *in*$(x, s))$:=
set_move$(s, E(Erep(s, p)),$ *step1* $(s, p), x, q)$: $I(Erep(s, p), x)$. { if $x \in s$ then $\hat{s} \mathrm{E} x$ }

16. Now we're almost ready to prove the principal theorem of these notes, the fact that equivalence classes are either disjoint or equal.

meets(s, t : set) := exists(elt, (x : elt) A (s(x), t(x))): bool. $\{\exists x\,(x \in s \;A\; x \in t)\}$

disjoint-or-equal (s, t : set) := implies (meets (s, t), set_eq(s, t)): bool.

{ if s meets **t** then s = **t**)

common-elt(s,t:set; p:proof (meets(Q))) := choose(elt, (x : elt) $\wedge$ *(s(x), t(x)), p): elt.*

{ if s meets **t** , this is a common element)

common-first (s, t : set ; p : proof (meets (s, t))) := first_conjunct (s(common_elt (s, t, p)), t(common_elt(s, t, p)), thus(elt, (x : elt) A (s(x), t(x)),p)): in(common_elt(s, t, p), s).

{ the common element is in s)

common-second (s, t : set; p : proof (meets (s, t))) := second_conjunct(s(common_elt(s, t, p)), t(common_elt(s, t, p)), thus (elt, (x : elt) A (s(x), t(x)),p)): in(common_elt(s, t, p), t).

{ the common element is in **t**)

17. This is it: The main theorem at last!

step1 (s, t : set; p: is_Eclass (s); q : is-Eclass (t); τ : proof (meets(s, t))) := pτoposition3 (s, p, common_elt(s, t, r), common_first(s, t, r)): I(Erep (s, p), common-elt(s, t, r)).

{ if equivalence classes s and t meet, their common element is equivalent to $\hat{s}$)

step2 (s, t : set; p : is_Eclass (s); q : is-Eclass (t); r : proof (meets (s, t))) := proposition3 (t, q, common_elt(s, t, r), common-second (s, t, r)): I(Erep (t, q), common-elt (s, *t, r)).* { and also equivalent to $\hat{t}$ }

step3 (s, t : set; p : is_Eclass (s); q : is-Eclass (t); τ : proof (meets (s, t))) := E_trans(Erep(s, p), common_elt(s, t, r), Erep(t, q), step1 (s, t, p, q, r), E_symm(Erep(t, q), common-elt (s, t, r), step2 (s, t, p, q, r))): I(Erep(s, p), Erep(t, q)). { hence $\hat{s}$ is equivalent to $\hat{t}$ }

step4 (s, t : set; p: is_Eclass (s); q : is-Eclass (t); τ : proof (meets(s, t))) := proposition1 (Erep(s, p), Erep (t, q), step3 (s, t, p, q, r)): is_eq(E(Erep(s, p)), E(Erep(t, q))). {hence $[\hat{s}] = [\hat{t}]$)

step5 (s, t : set; p: is_Eclass (s); q : is_Eclass (t); τ : proof (meets (s, t))) := IErep (s, p): is_eq(s, E(Erep (s, p))). { but $s = [\hat{s}]$)

step6 (s, t : set; p: is_Eclass (s); q : is-Eclass (t); r : proof (meets (s, t))) := IErep (t, q): is-eq(t, E(Erep(t, q))). { and $t = [\hat{t}]$)

step7(s, t : set; p: is_Eclass(s); q : is_Eclass(t); τ : proof (meets(s, t))) := set_eq_trans(s, E(Erep(s, p)), E(Erep(t, q)), step5 (s, t, p, q, r), step4 (s, t, p, q, r)): is-eq(s, E(Erep(t,q))). {so $s = [\hat{t}]$)

step8 (s, t : set; p : is_Eclass (s); q : is_Eclass (t); r : proof (meets (s, t))) := set_eq_trans (s, E(Erep(t, q)), t, step7(s, t, p, q, r), set-eq-symm(t, E(Erep (t, q)), step6 (s, t, p, q, r))): is_eq(s, t). { and in fact $s = t$)

proposition4 (s, t : set; p : is_Eclass (s); q : is_Eclass (t)) := implication(meets(s, t), set_eq(s, t), step8 (s, t, p, q)): proof (disjoint-or-equal(s, t)).

{ qed }

Appendix F: CHEX4

Supplement to the second CHEX script: Negation.

contradiction := *for_all*(*bool*, (*b* : *bool*)*b*): *bool*. { all ***bool*** can be proved)
by-contradiction (*b* : *bool*; *p* : *proof* (*contradiction*)) :=
    specialize (*boo1* , (*b* : *bool*)*b*, *b*, *p*): *proof* (*b*). { this proves *b*, if $\vdash$ *contradiction* }

not(*b*: *bool*) := *ior_all*(*proof* (*b*), (*p* : *proof* (*b*))*contradiction*): *bool*.
    { $\neg$ *b*, constructive negation: all proofs of *b* lead to a contradiction)
impossible (*b* : *bool*; *p* : *proof*(*b*); *q* : *proof* (*not*(*b*))) :=
    specialize (*proof*(*b*), (*p* : *proof* (*b*))*contradiction* ,*p*, *q*): *proof* (*contradiction*).
    { $\vdash$ *b* and I-- 1 *b* is contradictory)
hence-not (*b*: *bool*; *p*: (*q* : *proof* (*b*))*proof* (*contradiction*)) :=
    generalize (*proof* (*b*), (*q* : *proof* (*b*))*contradiction*, *p*): *proof* (*not*(*b*)).
    { if $\vdash$ *b* leads to a contradiction, then $\vdash \neg$ *b*)

modus-tollens(*a*, *b* : *bool*; *p* : *proof* (*implies* (*a*, *b*)); *q* : *proof* (*not*(*b*))) :=
    hence-not (*a*, (*r* : *proof* (*a*))*impossible* (*b*, *modus-ponens* (*a*, *b*, *p*, *r*), *q*)): *proof* (*not* (*a*)).
    { if *a* implies *b* and $\neg$ *b*, then $\neg$ *a* }

double-not (*b* : *bool*; *p* : *proof* (*b*)) := *hence-not*(*not* (*b*), (*q* : *proof* (*not* (*b*)))*impossible* (*b*, *p*, *q*)):
  · *proof* (*not*(*not*(*b*))). (if *b* then $\neg\neg$ *b*}

negation(*t* : *atom*; *p*: (*x* : *t*)*bool*) := (*x* : *t*)*not*(*p*(*x*)): (*x* : *t*)*bool*. { $\neg p(x)$ }
weakly-exists (*t* : *atom*; *p*: (*x* : *t*)*bool*) := *not*(*for_all*(*t*, *negation*(*t*, *p*))): *bool*.
    { $\neg\forall x$ -*p*(*x*)}

step1 (*t* : *atom*; *p*: (*x* : *t*)*bool*; *q* : *proof* (*exists*(*t*, *p*)); *r* : *proof* (*for_all*(*t*, *negation*(*t*, *p*)))) :=
    specialize (*t*, *negation*(*t*, *p*), *choose* (*t*, *p*, *q*), *r*): *proof* (*not*(*p*(*choose* (*t*, *p*, *q*)))).
    {if $\exists x\, p(x)$ and Qx $\neg p(x)$ then we have an x with both $p(x)$ and $\neg p(x)$ }
step2 (*t* : *atom*; *p*: (*x* : *t*)*bool*; *q* : *proof* (*exists* (*t*, *p*)); *r* : *proof* (*for_all*(*t*, *negation&p*)>>> :=
    impossible (*p*(*choose* (*t*, *p*, *q*)), *thus* (*t*, *p*, *q*), *step1*(*t*, *p*, *q*, *r*)): *proof* (*contradiction*).
    { and that's a contradiction)
3 *step3* (*t* : *atom*; *p* : (*x* : *t*)*bool*; *q* : *proof* (*exists* (*t*, *p*))) :=
    hence-not (*for_all*(*t*, *negation*(*t*, *p*)), *step2* (*t*, *p*, *q*)): *proof* (*weakly-exists* (*t*, *p*)).
    { hence $\neg\forall x \neg p(x)$ }
fact1(*t*: *atom*; *p* : (*x* : *t*) *bool*) := *implication* (*exists* (*t*, *p*), *weakly-exists* (*t*, *p*), *step3* (*t*, *p*)):
    proof (*implies* (*exists* (*t*, *p*), *weakly_exists* (*t*, *p*))).
    { we have proved constructively that 3 implies $\neg\forall\neg$)

nonconstructive (*t* : *atom*; *p*: (*x* : *t*)*bool*) :=
    #: *proof* (*implies* (*weakly-exists* (*t*, *p*), *exists* (*t*, *p*))).
    { nonconstructivists assume that the converse is also true)

trivial (*b* : *bool*) := *proof* (*implies* (*b*, *b*)): *atom*. { $\vdash b \Rightarrow$ *b*)
prop1 (*b*: *bool*) := (*p*: *trivial*(*b*))*b*: (*p*: *trivial*(*b*))*bool*.
    { a mapping that takes a tautology into a. specific *bool*)

prop2 (b : *bool*) := (*p* : ***trivial***(b))***not*** (*b*): (*p* : ***trivial***(b))***bool***. { similarly for its negation)
trick(b: *bool*) := ***for_all***(***trivial***(b), ***prop2*** (b)): *bool*.
{ for all proofs of *trivial (b)* there's a proof of $\neg$ *b)*
step1 (b : ***bool*** ; *q* : *proof (trick(b)))* :=
specialize (trivial(b), ***prop2*** *(b),* ***imp_refl*** *(b), q)*: *proof (not* (b)).
{ if so, there's a proof of $\neg$ *b)*
step2 (b : bool; p: proof (not (not(b))); q : proof (trick(b))) :=
. *impossible (not(b), step1 (b, q), p)*: *proof (contradiction*).
{ we can't prove both $\neg\neg$ *b* and *trick(b)* }
step3 (b : bool; p : proof (not (not(b)))) :=
hence-not (trick(b), step2 (b, p)): proof (weakly-exists (trivial(b), prop1 (b))).
{ hence $\neg\neg$ *b* gives weak existence of *b* }
step4 (b : ***bool*** ; *p: proof (not (not(b))))* := *modus-ponens (weakly-exists (trivial(b), prop1* (b)),
exists (trivial(b), prop1 (b)), *nonconstructive (trivial(b), prop1* (b)), *step3 (b,* ***p***)):
proof (exists (trivial(b), prop1 (b))).
{ and we can apply the nonconstructive axiom to get what we want: }
excluded-middle(b: bool; p : proof (not (not(b)))) := ***thus***(***trivial*** *(b), prop1 (b), step4* (b,p)):
proof(b). { $\vdash\ \neg\neg b$ implies $\vdash$ *b}*

Appendix G: CHEX5

Supplement to the first CHEX script: What we've always wanted to prove.

1 := *succ*(0): *nat*. { $1=0'$ }
2 := *succ*(1): *nat*. { $2=1'$ }
3 := *succ*(2): *nat*. { $3=2'$ }
4 := *succ*(3): *nat*. { $4=3'$ }

lemma(x : *nat*) := ***eq_transitivity***(***nat***, ***sum***(x, 1), ***succ***(***sum***(x, 0)), ***succ***(x), ***sum_ax2***(x, 0),
eq_functionality(***nat***, *nat*, ***sum***(x, 0), x, ***sum_ax1***(x), ***succ***)): *is* (*sum*(x, 1), ***succ*** (x)).
{ $\vdash x+1=x'$ }

2-plus-2 := *eq-transitivity* (*nat*, ***sum***(2, 2), ***succ*** (*sum* (2, 1)), 4, sum-ax2 (2, 1),
eq_functionality(***nat***, *nat*, *sum*(2, 1), 3, ***lemma***(2), ***succ***)): ***is***(***sum***(2, 2), 4).
{ $\vdash 2+2=4$ }