4 3081 TS: ~~[illegible]~~ → 81 TM 1–12 13 CRM



This will be a sort of thinking out loud about where I am & where to go now.

Haven't gotten much TM done since Pa's death ~~3~~ 10/31/80.

~~[illegible]~~ I did write a very Good Book Mark : 286.01 – ~291.40

292.01 – 298.01 are ~~a~~ ~~[illegible]~~ a partial analysis of some papers in "Sci" on Chimp learning. I'm not sure as to how relevant to TM this is: It mite lead to a way to design trg seqns. — or to get started on writing trg. seqns. 30.5 ×11 +15

---

CRM ~~[illegible]~~ (1981) 1.01 – 12.40 : This is criticism of a paper by Partridge, Johnston & Lopez : An attempt to Realize a model of Hebb's system of cell assemblys on a computer. 9.01 – 12.40 generalizes their problem & soln. so that it mite be of interest to solve it. Conceivably this could suggest trg. seqns. & genzns. to a UTM — tho t. ~~[illegible]~~ abstraction recognition methods usd by P, J. L are not univl. & are rather narrow, it probly would be easy to extend them.

**Note** A recent issue of Science (~4 23 81) has a review of a book about a bunch of papers about Hebb's work in "Org. of Behav"

Some work on **P.O.F.** (Pay off function) & perhaps some other probs relevant to TM.

---

My impressn. is that t. approach of CRM 9.01 – 12.40 could be usd to develop a TM! Perhaps using formal langs, grammers & ~~&~~ perhaps make a RTM!

— But that t. main problem, as ever, would be devising a suitable trg. seq.

T. present/problems Apparent in (& interpreted approach to TM) devising Trg. seqns) are reviewed in ~~TS~~ TS 1981! 286.01 – 291.40 = "BM".

My impressn. is that I do have some rather lngs, very rough, trg seqns. listed in these notes — 206.27 R — but that for t. most part, there is little detaild work done in "filling in t. gaps." [Also: Note 206.11 – .22]! ≥1 148, 146, 181, 172.

---

My present goal (I guess) is to continue from t. BM of 286.

280.01 – .15 is essentially what I was working on! I want to get as much as possl. of that stuff in rapid access memy. Also, I should be familiar w. all of t. refs. gn. in that ~~[illegible]~~ TM sequence. This putting stuff in "rapid access memy" is ordinarily very time consuming & not particularly pleasant, because it tends to not be very creative (& creation is what is fun). Try to get this unpleasant part over w. as soon as possl. & get into t. good parts!

SN 200.01 – ff may a list of impt. Headings (= ideas) in TM.

279.32 → 280.10 PTS ~~PTS~~

[In 280.01 – .15, there are 3 impt. ideas. ③ Is t. idea of using ~~PTS~~ (partial trg seqns) to test out various search methods & to teach me how to write trg. seqns.

② ~~[illegible]~~ Is t. idea of attempting to test t. apparently VG idea of ~~[illegible]~~ how to discover a seq. of operators of increasing power for a corpus of ↑ length. 267.24ff Also use of few ideas! 270.02 – .09, .12 – .18

① Is a particularly simple way to write trg. seqns. for a not ~~[illegible]~~ nearly very brite TM .... but it may be a useful method never t. less — just to get me started writing trg seqns. 272.28 – .40 w. summary on 273.01

5481 TS: 14

From 280.16 to ~281.27, I started work on a "alg. notation" PTS. [redacted] 281.28 + 282.29 discusses learning t. function "Eval" — which is a kind of gen'l ~~ev~~ evaluator of alg. expressns.
— It is a recursive type of function — ~~def~~ (≡loopy defn.) & is ∴ of particular interest.

282.25 – 284.02 discusses t. concept "1 substring of a certain form": This idea is usd in t "eval" function, & is a general "substitution" or "production" operator. I tried to genz. it to ~~> 1~~ >1 dimension, but w. uncertain goodness!

284.03 – .28 is a continn. of a previous discn. on t. PTS of learning to evaluate alg. expressns.
284.32 – 285.09: 3 possl. ways to get started on writing Tng. Seqs.
286.29 – 287.08 reviews 280.16 ≈ 284.40: in a usefull manner. .... lists 3 impt concepts of presumably hy pc. (string of a certain form", substitution, recursion)

287.09 — 291.10: This seems like a good discn. of t. state of my TM ~~[illegible]~~ rsch at that time: What t. critical problems are & some possl. solns. to them.

1:36 00

Actually 280.01 – 285.40 is an impt. part of it.
286.01 ff is, indeed a V.G. Book Mark.

5.6.81: What I want to do now is to understand t. stuff from 280.01 to 291.40 about as well as I understood it when it was written. My impression is that I had about all of that in Rapid access Memy, when I wrote t. latter part of it.

Some methods of doing this: Read each part carefully & be sure I understand it: Write criticisms & genzns. of each section. Go over t. various sections sequencially again & again untill I have all t. material [redacted] in rapid access memy simultaniously.

.22 : 286.16 Remark on t. diffy of 286.10: Actually, in solns. of many problems, t. pc's of many of t. abss. used are very close to 1, even tho t. defns. of these abss are of much less pc than 1. These "pc's close to 1" are ~~[illegible]~~ almost always conditional pc's — ie. these abss. have very hy proby in a certain environments. see 288.35 – 289.04 → ~~also~~ → 19.25
.25

.26 A start of an outline of 280.0 – 291.40:

My present ~~&~~ immedt. goal: to write tng seqs or PTSs to test out various TM ideas:
list some of them:
1) utility of 267.29
2) 287.10, .19
3) 286.10
4) is 288.02 very impt?
5)?
6) Test utility of method of 272.28 – .40 (summary on 273.01)

One idea is to use an Operator type TM. This is perhaps easily done in Forth form, using t. idea of 267.29 on how to optimumly do searches for Operator TM's in a tng. seq.

.30 SN; t. idea of ↑ can also be genzd. to any type of TM, (not nec'ly an Operator TM), in which one has at each pt in time a Machine $M_i$, then a new sub corpus $S_{i+1}$, & one must search over possl. $M_{i+1}$'s. T. a prior pd of $M_{i+1}$ is than that obtaind from t. seq. $M_1, M_2 \ldots M_i$, using any ~~[illegible]~~ induction system one wants, Bernoulli, Z141 or other langs, or full CB1.

$M_i$ is a machine that is able to adequately deal w. $S_i$.

A subgoal is to write several (or many) PTS's. T. "Eval" funct. of 281.28 is one possl. PTS. goal. — see 281.28 – 284.02 for some development of this idea.
There are ~~[illegible]~~ others suggested PTS's in 280.01 – 291.40 (.ie. 284.32 – 285.09 & ~~[illegible]~~ 290.20 – 291.10.)
Also ~~ref to~~ on 206 ... R where there are ~~[illegible]~~ refs to various tng. seqs.

PTS's can be part of a larger & desired TS or can be simply "study problems"

5.7.81 TS:

Better ~~Better~~ Outline of | T. material outlined by 14.26: This is an outline of t. BM of 280.01 – 291.10.

T. immediate goal is to write a useful tng. seq to test out various impt. ideas.

.03 Some ways to write these tng. sequs. (1) T. use of PTS's (Partial Tng. Sequs) (2) Use the PTS of 281.28 (t. Eval function) (3) some other suggs in 206.. R; 284.32 – 285.09; 290.20 – 291.10 for TS's & PTS's. (4) use of Forth-like notation make partial ordering of abss. clear.

Main immediate use → "study problem"

Remark: A PTS can be part of a larger, more (final) impt. TS., or can be a "study problem".

The way I want to try to solve these TS's or pts's: (1) Is to use Lsrch, (2) Perhaps try t. idea of 267.29 on improved models of TM (extrapolate t. sequence of partially adequate TMS). (3) Perhaps use an Operator TM since Forth ~~[illegible]~~. makes it easy to implement this type of TM.

Some ideas I want to test, using these (P)TS's:

.11 (1) utility of 267.29 & Lsrch, & Forth. L costs are acceptable? (2) 287.10 & 287.19: Can I obtain usable pc's from these (P)TS's, so This gives more detaild things to watch for: Also 286.10 is an impt. detail to watch for: i.e. that pc's of defind abss must usually eventually be >>> t. product of t. pc's of t. component abss. if t. defn. is to be much useful in Lsrch: This criticism may make non-Lsrch searches necessary: but see 81TS 4.22 also 286.10ff and 288.02

(3) T. utility of 272.28 – .40 (summary on 273.01). This is a simpler way to write easy-to-learn tng. sequs. It uses a large SSZ but needs small cc to find solns. [Note 289.05 on trade off betw. SSZ & cc. of search.]

(4) I want to get many leveld hierarchys of abss. defns., so I can see how well t. pc. assignments on t. basis of defns. & SSZ work. T. ≈ Forth formalism seems to make this study natural. Just write t. desired operator for t. final soln. in Forth. — Then write tng. sequs for each component defn., & for each component defn of that, etc. 'till one gets to primative operators.

.22

T. main ideas of this BM are (1) Some immediate TM goals: (P)TS's & some ideas of which to try & how to do them. (2) Some general impt. Q's & criticisms that one should apply to this (P)TS work.

5.10.81 TS! — Brite feeling day



O.K. back to t. ~~[illegible]~~ T.S. to Learn t. "Eval funct. of 281.28":

.01 One way to learn this: First learn A+B=X ; Eu. A & B (for various A, B values) to find X. Then switch to A−B=X & learn this, then A×B=X etc. So t. machine learns/ to work these sub corpi +, −, ×, ÷ etc. Hvr., at first, it isn't able to distinguish betw. them. It just tries various operators untill it finds one that works — Then stays with that operator until it doesn't work, & ~~[illegible]~~ then tries another. ~~The resultant code for t. corpus~~

.09 This is an economical coding method providing t. SC's each keep t. same operator for enuf examples to pay for t. choice of t. operator for that SC. (we could have 2 possl. "−" ops. & "÷" ops — the same op. but w. permuted args.)

All problems are of t. form: A $\{+,-,\times,\div,\ldots\}$ B = X. The problem is to find t. 5th qst. If there are only 4 types of operations involved, after a fair no. of examples, TM will ~~[illegible]~~, upon failure, quickly try all 3 other ops. to find t. one ~~that~~ that works.

.20 After working a fair no. of such problems, TM will have a corpus of problem-operator pairs (there will be a set of 4 ~~[illegible]~~ operators that TM has found useful. It will then note which probs are assoc. w. which operators: Since there are only 4 operator types, t. probs. are put into 4 difnt. groups. TM then tries to see how t. 4 groups can be distinguished: & this would seem (in this case) to be easy to do.

So we end up w. a single operator for all 4 cases. Next, we mite introduce a more operations like ∪ ∩ < > = ≤ ≥ etc. ~~[illegible]~~ t. operations = ≤ ≥ ~~[illegible]~~ > < have 0 or 1 as their values (i.e. "no" or "Yes") — or, special symbols meaning "yes" & "no". Just how TM would go about learning

.29 these ~~[illegible]~~ additional operations is unclear.

.30 Anyway, next we teach things like $(4+3)\times 7 = x$ [or $(4+3)\times(7) = x$]. First, I guess $(4+3) = 7$ should be taught, also $(4) = 4$ etc. ~~Ideally~~ These involve 1 substitution only, then applicn. of a certain operator that it already has found useful. Perhaps t. ~~[illegible]~~ concept of "substitution" should somehow be taught — preferably in a more general way than appears here.

.30 ff can use recursion. Whether this is t. best way to learn this concept, is unclear — but it doesn't make much difnce — rite now ...

5·10·81 TS. 17

I want to try out as many difrnt ways to learn things as possl ... so as to give me much needed experience in this area.

For t. problem learning of 16.01–.29; There are 2 simple models that could be used. In each problem, t. thing ~~needed is~~ to find which of t. 4 operators is to be used. If t. thing is regarded as ① A Bern seq., then for this simplest model, each of t. 4 ops gets a probty of $\frac{1}{4}$. 8:04 8:21

.08 ② For t. situation of 16.01–.09, after it finds t. correct operator, t. probty of t. next op. being t. same is $1-\frac{1}{\ell}$, where $\ell$ is t. expected run length for a single operator. Later, we may find $\ell_{1,2,3,4}$ i.e. difrnt run lengths for difrnt operators, rather than a single $\ell$.

Can I express t. progg. in a Forth-like notation of operators & conditional pc's? First try t. Bernoulli case.

---

.20 Some closely related prob. type that are very similar to t. progg:

known Input is symbol from alphabet of 4 symbols: $X_i$ $(i=1/4)$.

output = $X_i$ in one type of prob. (close)

output = $f(X_i)$ " another " " ". (closer)

In both types of problem, t. input $X_i$ could be either simple Bern. seq. or simple Bern seq. but with run length of expected length $\ell_i$ $(i=1/4)$

* 75.08.

---

5·11·81 ← Brite day. In line w. .20: Say one is devising a seq. of operators $M_i$ so that

$M_i(I_i) = O_i$ ; ~~we want t. $M_i$~~ $i = 1/n$; we want t. $[M_i]$ to have minimal dern. Ideally, $M_i$ is indep of $i$ & of max. pc. Another way would be to have a fixed $M_i = M_0$ "+" $k_i$, where $k_i$ is a small amt. of information. This also codes $I_i \Rightarrow O_i$ in a small no. of bits. T. no. of bits in $k_i$ gives t. uncertainty in t. operators prediction. For lower amts of certainty than 1 bit (but >0 bits) we will have to code $M_i$ from $M_0$ in >1 way. – In general, for more accurate results ($\gtrsim \frac{1}{2}$ bit accuracy) parallel coding must be used.

Regarded in the $M_i = M_0 + k_i$ manner, / $M_0$ plus t. sequence of $k_i$ ~~s~~. is a code for t. "corpus" — so in this way, Operator Induction sequences become much like sequential induction.

Hvr., in one kind (perhaps t. most impt kind) of Operator induction, the ordering of inputs is known to be apri irrelevant, so what we want to do is chose $M_0$ / $\dot{a}$ t. $[k_i]$ such that t. total amt. of info in ~~[illegible]~~ them is minimal (w. modifns for ll codings of both $M_0$ $\dot{a}$ t. "$M_0 + k_i$"'s.)

---

We have have $>2$ kind of T.S. idea! T. idea of 12.20 of operators predicting output as a bern. seq. w.o. noticing t. input.
At next level of development, output is a simple funct of input (small Table Look Up). Other trials betw. I $\dot{a}$ O are compositions of primitive operators. These, as well as t. **TLU** are tried in L cost order.
These ~~po~~ 2 kinds of problems are simpler than solving $1+3=4$, say, so perhaps we should do them first.

Note that for t. bern. seq. soln. t. "soln" is a stochastic operator, while t. more complex operators are all deterministic.

Another interesting note: for t. t.s. ~~=~~ $I_i = a_i$; $O_i = f(I_i)$, we can have either a stochastic (Bern seq) soln., or a deterministic functional soln. A Q. is, at what pts. in t. L srch do these 2 solns. occur?

T. occurrence pt. may depend on SSZ.

[ Well, at present, I have this little trg. seq: starting w. Bern. seq., (simple operator, or unary functs) **TLU,** (Binary functs) $A \left(\begin{smallmatrix} + \\ - \\ \times \\ \div \end{smallmatrix}\right) B = X$, then toward t. more general "**Eval**" operator. Hvr., I want to develope this at hy level ("English" notation) as much as possl. at first. ]

Note: See just how well I can introduce forth-type notation.
It may be that on t. present level, **Forth $\dot{a}$ Lisp** are about t. same — ~~[illegible]~~ — their primitives may be difrnt. — but I'm not much concerned yet w. ~~[illegible]~~ which primitives to use ~~[illegible]~~ Lisp uses to an impt extent, classical ⟨recursive function theory⟩ notation, $\dot{a}$ also its primitives. — The Lisp uses parentheses — which is a minor variation on Polish $\dot{a}$ RPN.

T. technique of learning simple unary operators seems much difrnt from that of using 1 argt TLU. / similarly for 2 argt TLU v.s. usual Binary operators.

5.11.81 TS 

At first Glance TLU & <Functions fashioned from primatives> would seem to be very difrnt.
TLU usually involves a small no. of values for t. argt. v.s. a very large no. of values for "functions".
TLU is constructed much difrntly than functions.

Tho to a mathematician they are very similar, to a simple TM they seem to look like entirely difrnt objects. — Mainly because t. argts are difrnt. & they are constructed difrntly.

Actually, these 2 are quite impt. Everything An Operator T.M. does is a "function", yet t. similarity betw. all functions covers a very small part of their properties — which are mainly very difrnt.

→ So don't be too disturbed if an elementary (or even fairly advanced) TM doesn't recognize <functions fashioned from a few primatives> are also TLU.

---

Hvr, in doing the $A \overset{\times}{\underset{\div}{+}} B = x$ problems we do want TM to devise t. function ($\equiv$ TLU) relating "+" to t. addition primatives, "−" " " subtraction ", etc.

— So to learn these functions properly, we want TM to be able to learn TLU also.

---

well O.K.! : we can just say ~~that~~ "TM can learn TLU", & we can work out just how it does this at a later time, if necy.
— But it is, to some extent, a separate, ("≃ well defined") problem, that is ~~workable~~ work-on-able as such. A sub-problem requiring, perhaps, a special Tng Seq. → Hvr., see 21.10 →
.24

---

.25 : 286.16 (14.25) SN: T. remark of 286.10–.16 may mean that in most cases, t. exact details of ~~#~~ how to evaluate t. pc of a definition are not very impt. That they amount to only 1 a few bits. That we really need to get big [illegible] savings out of using definitions before we can effectively use them to get acceptable costs of srches.

→ 21.25

---

.30 A simple kind of tng seq. that will be good for initial problems. The seq. comes in sections. (≃ sc's). Each section is learned, then t. next section is presented!
e.g. first t. (section 1) Bern seq. w. say 4 symbols & probs .1, .2, .3, .4. | Next, t. identity operator (section 2) $O_i = I_i$ | Then a simple (section 3) TLU function |, relating $I_i$ & $O_i$ (section 4.) next a unary function relating $I_i$ & $O_i$.)

Now just what TM carries over from one section to t. next, is unclear. In t. examples just gn., I'm not even sure that there is any point to carrying any thing over ... i.e. there seem to be no common concepts.

---

T. manner of carryover (if I needed any carryover, & it seems pointless to write long seqs in which carryover is of little value!) could be treated as in 17.08 w. "expected Run length of $L_i$" — T. idea of TS.13 that "carryover" is a centrally impt. idea!

As soon as TM dees that an operator no longer works, it tries to make a new operator using (t. history of) previously successful ops. as a corpus (see 80 TS 267.24 – .40) ← meaning t. set of "i.e. (0682)

---

On t. "carryover" in 19.30 → .40 : t. concepts learned do seem useful — there should be some more complex probs that

51281 will use these concept. So Befor developing t. lang. seqs in any detail (other than English), try to find more complex problems that would need these lower level concepts. Maybe first define t. lower level concepts that have been learnd!

.20 1) Bern seq. learning: stochastic seq. & pure ; → 22.21, 23.20; 27.29

.21 2) simple TLU (Table Look Up) function (operator learning) deterministics (association learning). → 21.10, 27.33

.22 3) Perhaps 2) w. partial Bern. seq. learning for stochastic parts. Essentially, t. basic form of conditional proby — stochastic assoc. learning. → 27.38

This does combine previously learned obss.

.24 4) unary function learning: simple primative functions: –x, |x|, parity of x, sign(x), etc. ; next, combinations of these functs, yeilding more complex unary functs. → 28.02

yeilding more complex unary functs.

Association of these functs w. their labld names.

.27 5) Association of these functs w. their labld names

.28 6) binary function learning (like 4)

7) " " " w. assocn. of labld names (like 5)

8) substitution of functional values. This uses 4) (& 6)).

e.g. 5+(3×7) → 5+21 | or sin cos 2 → sin(–.416)

.32 9) recursion ; 5+ (3×7) → 5+21 → 26. (or sin cos 2 → sin(–.416) → –.404)

continuing: 21.18 – .24 ; 22.07 – 18 ; 80 TS 206 ; of 20.20ff

5.12.81 TS

I did have some objection to having TM learn up to solving simult. linear eqs., because I didn't see that it led anywhere (tho rite now I'm not so sure of this last). At any rate, going up to that pt., (& probly well before it.), I would have lots of nested definitions, & perhaps even "plans", ~~integrated into t. system~~ that t. system had learned, so I could see how t. Q's of 15.11 — .22 work out.

.10 :19.24 Note on "TLU learning" This is more like learning to recognize $\alpha$, than having $\beta$ associated w. $\alpha$ & having $\alpha$ "trigger" $\beta$. — instead of making a table & using TLU in t. usual sense. Furthermore, t. approach of .10 seems more easily expandable to conditional probability, than is t. usual TLU approach.

~64K students in Boston Pub. Scools 10KK 64K

.18 An interesting Q: Say we get t. trg. seq. up to solving equs. (in 1 unk. say): It knows how to do all linear equs & can solve also, ~~many~~ non-linear & transcendental equs. For all types, there are several ways to solve or attempt to solve each equ. Would TM learn which soln. methods are likely to be smallest cc for each type of equ? While its likely that I could teach it this sort of thing using a suitable trg. seq., would it perhaps try to do this automatically — being interested mainly in solns of low $\frac{cc}{pc}$ → 22.07

100 M
6.4
1.6K
965 = DJI Tues

.24

.25 (5.13.81) Perhaps one of t. Main things I was (& am) worried about! Essentially 286. 10 — .16 (see also 14.22, 19.25): That when a human decides to make a new defn. & he thinks (quite correctly) that it is v.g., it may be much better than would be warranted by freq. studies of its successful use in t. past: T. human is apparently bringing more info to bear on t. issue than ~~TM~~ ~~it's~~ a mechanistic TM can readily summon. 286.19, .23 & .26 are some possl. sources of such aux. info for a TM. Giving t. TM access to future problems is perhaps o.k. for a "student", "developmental" TM., but for a "production model" — one solving real problems in RW., I think It will have to somehow induce what future probs. will be like (just as humans do), & use this expected future to help evaluate t. utility of proposd newly ~~created~~ definitions. → Sometimes t. human will have more info than I realized! eg. in deriving substitution! 59.04 Also see 59.11 for a difft kind of example

Ultimately, what I want to be able to do, is get a human soln. to a difft. prob., then try to find a way (a "heuristic") that makes that soln. "a reasonable one" — in t. sense that one can "see" how it could have been found w. reasonable cc. Then I try to devise trg. seqs. so that t. cost of that soln. will be reasonable for TM, using all of t. (apparent) heurs usd by t. human. — T. trg. sequs. are for t. various, heurs, concepts, abss., that seem to be needed for t. human's soln.

T. main problem will be to find enuf "heuristics" for t. human soln. — i.e. an "adequate set" of heuristics — so that TM, using those heurs could solve t. prob. in reasonable cc.

51381 TS:

Perhaps one of t. reasons that I hesitate to do more work [i.e., more detail] 20.20ff is that I fear that it wouldn't lead to anything interesting.

This fear is groundless for at least 2 reasons.

.04 1) 20.20ff even by itself, i certainly w. a bit more concentration, should give hierarchies of abss., so I ~~can~~ can check on some of t. diffys of 15.11 – .22

.07 →[21.24] 2) 20.20ff can be continued in an interesting manner: see refs of Pages 206 for various kinds of continn. w.o. looking at page 206: Consider all t. dif'nt ways of solving eqns. — linear, polynomial, trigonometric, transcendental .... Successive approxn., algebraic manipulation, Soln. of Literal eqns.; Soln. of literal eqns in "closed form" v.s. successive approxn. [Actually "closed form" usually means that one can be sure t. algn. converges .. . In some successive approxn. methods, ~~its not as~~ convergence is not so easy to be sure of!]. The concepts of "solution" of "quantity", of "approxn" of "error magnitude" — These could all be very impt. i all learnable int .tng. seqs.

.18

Keeping .04 i .07 in mind, lets go back to 20.20 i put in a bit more detail: Let's regard each of these (9) pts. of t. tng. seq. as sub-goals — not necly in t. correct order, not necly including T.S. elements for all of t. needed concepts.

As written, ~~20~~ 20.20 ff –.40 looks like t. sort of thing one might write for a human, if one wanted to keep t. $c_j$'s [i.e conceptual jump size] small, [heuristic] ($\equiv h$'js).

Def .21 So o.k. 1): Baum seq. learning: This could be learned by coding as in I i C II, but perhaps simply regard it as a primitive Concept. → 23.20

.22

---

51481 SN One of t. impt. ways around t. diffy of 21.25 is t. ‖ search method of linear regressn. There are at least 2 impt. ~~to~~ concepts involved:

1) That any prediction method is equivt. to a coding method., <i so one has various concepts available that are suggested by a/o used by prediction>

2) That there should be some way to automatically get ~~pages~~ ‖ search i things like it (like it in t. sense of cc), This "automaticness" would seem to be able to be tied w. testing codes (or "concepts") in $\frac{cc}{pc}$ order — since t. sort of ‖ codes should in some sense, be assigned ~~a~~ a/o lower cc a/o hyper pc.

Essentially, t. concept of Lsrch ($\frac{cc}{pc}$ order) is el., in t. sense that it is t. $\frac{cc}{pc}$ that is assigned to each code that determines order of search — ~~a~~ While a more non-el. method would consider $\frac{cc}{pc}$ of a set of codes — or even less el., t. $\frac{cc}{pc}$ of t. entire search!

So I do want a Genzn. of <t. $\frac{cc}{pc}$ concept for individual code trials>.

Hvr., It may be possl. to make an acceptable $TM_1$ using simple Lsrch, then use it as a $TM_2$ to get better search, using a less el. criterion.

One problem in t. linear Regn. ll search is that one doesn't know untill t. end of t. calculation, what t. final total pc is, & one has no estimate of it before that time — so that one can't stop at any pt. because $\frac{cc}{pc}$ > Threshold.

One can, hvr., do a prediction caln. on t. basis of "expected value" of accuracy of predn. & stop t. caln. if cc gets too large. One would stop when $\frac{\text{expected cc needed to complete caln.}}{\text{expected pc of soln.}}$ was > Threshold.

**Actually, I think t. problem of how to proceed may be a Stochastic Part (SP) problem.**

Another possy., is that t. parallel nature of linear regn. coding is illusory — that by coding a short section of t. corpus, then coding successively longer sections, using more coifs & more bits/coif., one could do t. same thing as solving simultaneous linear eqns. w. perhaps t. same (or even less) cc.

Hvr. in this approach, it may well be that linear regn. doesn't begin to be much good as a coder untill a certain threshold of no. of coifs & accuracy of coiffs. is usd.

.20: 22.22 5·16·81 Re. simple Bern. prediction: I had that I dea that: when people are told that they are to be gn. a Bern seq. & they are to try to guess t. next symbol, then they don't simply stick to t. most frequent symbol, but guess symbol $S_i$, w. proby ≈ t. empirical freq. of $S_i$. I that t. reason mite be that people didn't believe it was a Bern seq., so they tried various models to get 100% predn. — & most such models gave t. guess $S_i$ (as "most likely") w. frequency $p_i$ (≈ t. empirical freq of $S_i$).

More exactly, if t. subject uses any deterministic, "near past" model & says: "when, t. last time, a certain sequ. occured, it was followed by $S_i$ & that seq. has occured again ∴ I will choose $S_i$" If he uses models of this type: (even if he generalizes & says "when a certain class of sequs ever occured in t. past $S_i$ followed ... & a member of this class has just occurd ∴ I'll guess $S_i$" —— then he still guesses $S_i$ w. frequency $p_i$ ($p_i$ ≈ t. freq. of $S_i$).

Anyway, I mite build a simpler predn. method into TM that is of this sort, but instead of using t. last case to decide what follows each proposed abstraction, TM would search thru t. corpus & use ≈ t. "straight Rule" or laplaces rule for estimation of probys.

So, one could "code" a corpus, by devising a ~~[illegible]~~ defn. of a set of sub sequences, & then examine, Bern-wise, t. freqs. of symbols that follow that set. Trouble is, this coding method is like linear regn. coding, in that Lsrch seems not to work! One can obtain t. pc of t. defn. of t. set directly from t. pc's of its component abss., but t. final pc of t. code is not available untill the end of t. coding process — so one doesn't know when to halt the search process for exceeding t. cc threshold.

In this, as in linear regn. coding, the cc of t. coding process is, for long seqs, $\propto n$, t. seq. length. It mite, hvr, be worth while to use very short sample seqs at first, to save cc & elim. ~~[illegible]~~ "~~unprom~~ unpromissing coding methods" quickly.

Coding using this method, consists of making t. defn. of t. ~~[illegible]~~ sub-set of seqs, then finding examples of it in t. past, then looking at t. probly distribn. of symbols that have followed it.

O.K.! So one makes t. defn: pc = $P_1$; One starts looking for examples. As one finds each example, one obtains an equivt pc change for t. seq. using those problys for t. sequence as far back as one has examined it thus far. ~~[illegible]~~

As one goes back in time, ~~[illegible]~~ t. $\Pi$ pc $\downarrow$ & t. $\Sigma$ cc $\uparrow$ untill t. Threshold, $\frac{cc}{pc}$ is exceeded — So one can use Lsrch in this method!

25

Could one use t. same thing in linear regn. coding? — Well, maybe. ~~[illegible]~~ Using a history length $l$, one makes a correln. matrix & solves it for t. ~~[illegible]~~ MS error & the expected errors in t. coeffs — to determine total pc. of t. code. If t. $\frac{cc}{pc}$ threshold has not been exceeded, one gets t. correln. matrix for history $2l$, by getting t. correln matrix ~~matrix~~ for ~~t~~ from $t = -2l$ to $t = -l$ & adding it, coif by coif to this correln matrix (which is from $t = -l$ to $t = 0$). One then solves t. eq. & again makes a pc evaln for t. corpus. ~~[illegible]~~ If $\frac{cc}{pc}$ thresh has not been exceeded, 5.16.81 TS.

one ~~way~~ calculates t. correln matrix for from t = −2l to t = −2l, etc. ... This continues untill t. Threshold is exceeded, or untill t. whole corpus has been coded.

One trouble is That The forgg. method is rather inefficient because one ~~one~~ inverts all Those Matrices .... & its really necy to only invert once — t. final one. To get around This, make l large enuf, so That t. cc in computing t. matrix from ~~t to t~~ $t = t_\alpha$ to $t = t_\alpha + l$ is ≈ t. cc of inverting t. resultant matrix. T. result is That t. cc added by repeatedly inverting matrices, at most only doubles t. cc. of t. resultant coding method! By making l larger,

one can make t. method even more "efficient" in This sense — But Then the cc for ~~to matrix~~ < constructing t. corr. matrix + solving t. corr matrix> becomes larger, so we end up trying This particular method only when we have a fairly large $\frac{cc}{pc}$ Threshold — i.e. we ~~s~~ examine This coding method later in t. search.

For linear regn. coding: I would have to make a study (If I haven't already done so & can find t. study) of pc saved by This codetype

.20 as a function of corpus length.

---

[5.17.81] Well! T. forgg. stuff on ≈linear regn. coding still isn't directly usable: For coding for normal Lsrch, t. pc is immediately known for t. proposd code. What is not known is (1) t. cc (2) if t. code fits t. corpus.

.26 For lin. regn. codes: The ~~pc~~ cc (or a good approxn. of it) is immediately known. Also, it's known That t. code(s) will fit t. corpus. However, t. pc is not known untill t. code is completed — The part of it (t. defn. of t. method: ("linear

.28 regn. coding")) is initially known.

.29 Actually, I may have had a diffnt. approach to This entire ~~prob~~ problem: Say D is t. dcrn. of a PEM. (= proby evaln. Method). T. dcrns. of Pems are a prefix set to which Lsrch is directly applicable. A Pem can be usd to ~~...~~ assign a proby to a long seq. of data .... (numerical ~~...~~ a/o non-numerical). So — one does an Lsrch over all Pems on t. Basis of t. pc's of their dcrns.

~~...~~ Two notes: (1) T. ~~...~~ set of all Pems is not r.e. (i.e. countable) (2) Many Pems (like linear regn.) have a ≈known cc for computing them.

5.17.81 TS

The fact that t. set of Pems is not [redacted] r.e. ~~may~~ may not be very impt. — because Lsrch [redacted] may be able to take care of this. i.e. [redacted] one can list all "Pem-candidates" recursively. This list includes all Pems, but it also includes things that are not pems. The "not pems" take ∞ time to be sure they are not Pems. Any Pem (I think) may eventually have its Pemness verifyable, but this can be any finite cc.

On t. other hand, it may be that there are certain things that are, indeed, Pems, but that this fact is not verifyable in finite cc.

---

T. way it looks is that this method of first chosing a Pem, then using that PEM to assign a ~~prob~~ pc to t. corpus, is a most general way of assigning pc to a corpus. One must include as a possl. pem, t. usual way of trying to code t. corpus directly, — presumably by Lsrch.

Appears

If ~~conceivable~~ that t. converse is also true, i. that [redacted] chosing a Pem & using it to code t. corpus (assign a pc to t. corpus & ∴ via "coding thrm", to) is simply another coding method, [redacted] — but I'm not [redacted] sure of t. mechanics of this.

---

.20 One way to use these ideas: At each pt. in time, TM has a set of Pems, w. assoc. known pc's (for their Names). Among t. PEMs is, $P_0$; one that creates new Pems. $P_0$'s (name) also has a pc.

Looks v.G.

One of t. Pems may be a simple, universal coder — But if it is universal, it will [redacted] produce (eventually) some of t. same codes as t. other PEM — ∴ (perhaps) overlap ≡ double coding. It is possl., hvr., that the overlap would be O.K., if t. codes generated by t. universal coder have a special prefix that makes them difrnt from those generated by t. other pems > These pems could "call" one another, or "call" t. "TM system as a whole" — so they can be recursively defined.

So this is probly O.K.

.29 → 27.01

---

(SN) How I got into this: Using even just as simple a trg. seq as 20.20ff, I wanted to do Bern seq. coding & conditional Bern seq. coding. These turned out to be ≈ to lin. reg. coding in that it had characteristics 25.26–.28 & ∴ not directly amenable to Lsrch.

T. method of 25.29 is an old approach, as is 26.20 → — But in both cases [redacted] I didn't have in t. past, a trg. seq. to apply them to — also [redacted] (to some extent) I didn't yet know about Lsrch.

.01: 26.29: A (perhaps) practical way to implement this! Just try ~~to~~ deriving t. working of t. Tng seq of 20.20ff as a human might do it — then put that human soln. into t. ~~form~~ form of 26.20ff — or modify 26.20 so that it includes t. apparent human soln.

Actually, t. problem of searching over "all" pams in pc order is not so easy. A practical example is (≈linear) regressn. for SM. Here we have a long seq. of nos. T. problem is to decide ~~which sets of coifs~~ ~~to~~ ~~try~~ what order in which to try various possl. sets of coifs. (T. cc of each term is easy to compute.) There are really an enormous no. of possys, if one includes first, 2nd & third powers ~~...~~ of all data pts, ~~...~~ also cross products betw. various terms, Also Abs value, etc.

.18 One way would be to use a very large no. of terms, then try eliminating those that have ~~...~~ t. smallest coifs, & see if this gives

.20 higher pc for t. entire code.

Also, by assigning pc's to each possl. term thru aux. info.

Another scheme would be a sequencial strategy of doing expts. & trying new expts on t. basis of info learned — .18–.20 is an example of this.

.01ff Looks quite good. Perhaps even more general — just write out "≈human" soln. to t. tng. seq. of 20.20ff, then try to formalize &/or simplify &/o genz. that "human" sol. — not necly ~~...~~ biased by 26.20ff.

O.n. 20.20ff!

.29 1) Bern seq: Say we have small alphabet. TM's problem is to predict t. next symbol. T. Bern. seq. model is tried first. Next, we try to see if t. previous symbol influences ~~...~~ a symbol, & we

.32 make more & more complex Markov models (— but of course they don't help.) → 34.12

.33 2) Simple TLU funct: given sample of ordered pairs: $\alpha_i, \beta_i$, to find 2nd member. T. rule is deterministic. First TM may use Bern seq. model for $\beta_i$. Next it tries $\beta_i$ as ~~...~~ deterministic funct. of $\alpha_i$. Since this works w. 100% accuracy, there would be a tendency for a human to stop looking. Hvr. TM could look for better models, that have

.37 greater pc for t. entire seq. — &/o less cc. → 34.36

.38 3) problst model of 2) I worked same way — i.e. same order of models tried, but deterministic model is not v.g., so ~~...~~ ~~condition~~ conditional Bern seq. used. There is only one symbol that it

5.18.81 TS

.01 can be conditional on, so this is a hyp. model. → 34.36

.02 4)2) Simple numerical unary functs; TM is gn. pairs of nos: $x_i, x_i'$ for its corpus. Then it is gn. $x_j$ & it has to find what follows.

It tries to find t. 2nd symbol as a funct of t. first, by trying various

.05 functs available to it. → 35.01

.06 5.21.81 Note on Lin. Reg. coding: If one is coding a short segment as a sample of a larger corpus, then the bcost savings in t. short segment can be used to calculate t. probty distribu. that t. larger corpus will have per symbol bc savings using this coding method. We obtain various overhead " reductions using t. larger corpus, & we can estimate t. variance of t bc savings in t. small sample & see if its likely to be "due to chance" — Hvr. this is an impt. theoretical problem that I spent much time on in t. past ... don't remember if I ever got any v.g. results!

My present impressn. is that in t. prob. of .06, it is abs. necy to do something like assume an apripd. Whether this is an adequate assumption is unclear.

Note: In t. Max method, using a uniform Apripd gives reasonable results, & its my impressn. that using any gaussian apripd about zero, its usually easy to approx. just what t. correction is.

. 28 My impressn: That t. big problem was this: One has a certain coding method that yielded a certain bcost/symbol in t. past. What is its expected yeild in t. future ... its varc. ?

Also, more generally, say I used a gn. Pem in t. past & I had a certain Gore in t. past. What is t. expected mean & varc. of this Gore in t. future?

Around 1973 — when I was at MIT, I had t. idea that there was some general way involving t. no. of bits in t. uncoded seq., & t. no. of bits in t. coded seq. Later, hvr, I got t. idea (thru, .34 perhaps, specific examples) that each specific case would have to be dealt w. — that there was no simple formula on how .36 to do this. This work may have been done in 1980.

I think ~.28 was t. PW (≡ Pem wtng) problem. Also, .38 say one had a large no. of PEMs. — so that its likely that at least one would look v.g. when applied to a gn. corpus. — but

5.21.81 TS. 

because of t. large no. of pems used, this "looking v.G." is likely to be spurious. — How does one deal w. this?... e.g. how to wt. t. pems when combining them?

28.38 was, I think, particularly impt., when t. pems were designed ~~by~~ by humans — ẚ ~~oft~~ were ~~d~~ somewhat rhetorical i.e. they had axes to grind. It may be, hvr., if they were made by TM., that TM could be ~~u~~ unbiased ẚ give them honest pc's.

[5.23.81] One problem perhaps faced by 28.34–.36! That when the proposer of a PEM was "axegrinding", one wanted to know t. Max ~~[illegible]~~ damage (error) that could occur, ẚ I think t. idea was that it depended critically on just what form of pem was allowed.

Another impt./possl. discovery (I'm not sure I was certain of its truth at t. time) — probly ~ 1973, was that there were 2 sources of variance in t. boost of a gn. pem ~~[illegible]~~ evaluating a corpus: One/source was t. simple SSZ effect — easily evaluated! Plus one other ~~[illegible]~~ source that I don't remember! ⇒

→ One impt. point: If t. "large no. of PEMs" was N, then t. aprip of each pem ↓ by a factor of ~ N.

---

Rite now, Re: 20.20 ff: one of t. immediate problems is how (if at all) to apply Lsrch to cases in which pc is not known before t. ~~[illegible]~~/trial occurs.

.23 There seem to be at least 3 kinds of search probs. of interest!

(1) NP probs: Lsrch is fine.

.25 (2) Direct induction coding (CBI) Lsrch is fine.

.26 (3) PEM coding. : I haven't really worked this out in any detail ẚ I don't know to what extent it will work. Say t. set of all pems derns is partial recursive. If so, I can do Lsrch on them. Hvr, t. results may not be ~~of~~ useful!

( low $\frac{cc}{P_n}$ first )

Say $P_i$ is t. pc of t. (Name) of t. $i^{th}$ pem. Say $Q_i$ is t. pc of t. corpus w.r.t. t. $i^{th}$ pem ($\equiv$ Pem$_i$). Then I can more or less try t. Pem$_i$ in effective $\frac{cc_i}{P_i}$ order; $cc_i$ being t. cc of creating Pem$_i$ ẚ evaluating t. corpus (i.e. calculating $Q_i$) with it.

$\frac{cc_i}{P_i \cdot Q_i}$ order would be nicer ẚ would make routine Lsrch possl.

Well, maybe just try them in $\frac{cc_i}{P_i}$ order ẚ ~~use~~ wt. them

.38 as $P_i \cdot Q_i$. Just how bad would such a srch be? → 31.01

It should be possl. to use something like 29.25 – 25.20 in this — i.e. use a pem on a short section of t. corpus to try to find pems

52381 T.S.



quickly (or cheaply) that work well w. it. [redacted] In t. case of linear regn. coding, for a first trial, make t. subcorpus usd. of such a length that t. cc. of t. correln. matrix ≈ t. cc of solving it.

For other kinds of Pems, its not clear just how one should do t. search using short corpus samples — But perhaps do t. linear regn. problem in some detail as a "study problem", then try to genz. t. method to all other kinds of Pems.

---

Gen. discn. of 29.23 ff:

It may well be that I could write useful, interesting trg. seqs using only N.P. & Direct induction coding: (I.e. no PEM coding).

.17 Hvr., certainly for a general universal direct induction codes wouldn't work — because t. pc. of a long sequence of numbers (that one would normally use a PEM search on) is much too low for L search — furthermore, t. low pc is "essentially" low & can't be ~~[illegible]~~ increased by suitable [redacted] pretraining.

ex 0 : 0

zero : ∅

My impression is that there are usually 2 difnt kinds of corpi for induction. One is rather short, so CBF can be usd directly on it — another is too long (too small pc) for direct CBF, so PEM search must be usd (or something other than direct CBF).

Q's: is there kind of corpus in which both kinds of things occur?

— Yes — & I'm hoping I can separate out t. 2 components w.o. too much diffy. Is there a kind of corpus in which these 2 kinds are ~~[illegible]~~ "tightly" mixed? — probably — but this would be a very diff't corpus [redacted]! Whether it could be dealt w. would depend on t. extent to which one could "unmix" t. components.

T. Problems of .17 ff are important. Some good ideas of what t. solns. mite look like could be obtaind by doing t. trg. seq. of 20.20, including various Bernoulli-ish type problems.

5 23 81 T.S. ≥1

.01: 29.38: Well, one has ~~a~~ some assurance about how long it would take to obtain a particular soln. (an assurance that is present in normal Lsrch).

e.g. say we use $\frac{cc_i}{p_i} < R$ as a ~~cutoff~~ stop rule.

Then $cc_i < R\, p_i$ so $\sum cc_i < R \underbrace{\sum p_i}_{<1} < R$.

Well, so if a gn. desired soln. exists $cc_j, p_j, Q_j$, ($Q_j$ is pc of corpus wrt $Pam_j$)

then we will certainly find it in $cc \lesssim \frac{cc_j}{p_j}$

Unfortly, while this gives us early solns. w. hy $p_j$'s it does not bias us toward early solns. of hy $Q_j$

Well, maybe it does in t. follg. way: Say $Pam_j$ has in t. past give rise to very many large $Q_j^k$'s (for sub corpi $sc_k$). Then $Pam_j$ get large pc due to useful use, to augment any pc that it gets because its found of useful, hy pc concepts. So if $Pam_j$ ~~is~~ has been useful in t. past, ~~the~~ $p_j$ will get larger so $\frac{cc_j}{p_j}$ will be earlier in t. ordering. While this ordering is completely indip of $Q_j$, it is, non-t.less very useful.

In general, even if I could ~~do~~ do a $\frac{cc_i}{p_i \cdot Q_i}$ ordering

I would end up with an assurance that total cc is $\lesssim \frac{cc_j}{p_j Q_j}$

— which is enormously $> \frac{cc_j}{p_j}$ & certainly too large to be of interest!

So the assurance $cc \lesssim \frac{cc_j}{p_j}$ may be o.k. & may be about as good as we can get!

---

~~[illegible]~~ The idea of this is that an "experienced TM" will tend to ~~[illegible]~~ have hy $Q_j$'s ~~[illegible]~~ for t. ~~many~~ sc's worked by Pams of large $p_j$. These ~~[illegible]~~ $p_j$'s will probly have to be conditional pc's — i.e. obtaind by scanning t. (usly recent) past of t. corpus.

One way to look at t. "Long corpus" needing PEM's: That this is a more "classical" kind of induction, & is more "limited". It usually needs larger SSZ. T. other "straight" CBI kind of induction is for "smaller" ~~samples~~ corpi & is "completely general".

Classical induction will (usually?) ~~[illegible]~~ have codes of t. form: $\alpha^\frown\beta$, where $\alpha$ is a relatively short dcrn. of t. form, & $\beta$ is a section $\propto$ t. length of t. corpus, $\beta$ is "pure noise" as t. corpus expands. $\alpha$ is constant as t. corpus expands. For this reason, $\alpha$ can be estimated & its expected "error" (= bits/symbol of corpus) estimated on t. basis of a short sample of t. corpus. It may often be that t. cc of t. code $\alpha^\frown\beta$ can be divided into $\geq 2$ additive parts: t. part that depends $\approx$ (maybe ln n factor) linearly on n t. corpus length --- which is $\beta$ & t. part that may have a large constant term & maybe a ln n factor.

I vaguely remember having done some work in t. past on difference betw. t. 2 kinds of predictors.

predictors:
Bandwagon
underdog.
verifiability
Moral responsibility of predictor.

5.28.81 TS



.01 A simple, practical way to write [And Solve] Tng. Seqns & PTS's!

1) First do like 20.20 for various ~~[illegible]~~ kinds of goals, fitting in various sub-goals that would seem to be useful for a human student. Put in lots of steps (sub-goals). (This is ≃ t. way 20.20 was obtained).

2) Write down t. soln. ~~[illegible]~~ (or solns.) desired for each sub-goal — at first in English, then in a more exact language. Very often, > 1 soln. will be desired.

3) Dsrb. t. sub-space of search, for each problem. → .27

4) Using t. soln. dsrn. of 2) ~~[illegible]~~ & t. searchspace of 3) ~~[illegible]~~ estimate t. pc. of search for that soln.

5) The Tng. seq., like 20.20, is sequential: [its ~~[illegible]~~ ~~Each set~~ sequence of solutions form a kind of corpus. (each soln. is an "object", & ~~[illegible]~~ this sequence of objects constitutes this "corpus".) If $S_i$ is a soln. for the i th $Sc_i$ (sub corpus$_i$), then one can use any standard extrapoln. device to obtain t. apripd for $S_{i+1}$. This is t. idea of 80TS 267.~~24~~29ff. Z 141, Bern seq., stoch CF gramms etc. can all be used for this extrapoln. The use of a Bern Seq. model (or, more likely Z 141 ≡ "coding w. definitions") will probly give my usual ideas of pc's of various abss.

.27 An unclear step is 3rd: defining t. search space! Just how this is to be done, is unclear. It may be that there are only a few possl. search spaces & its easy to decide which to use. Anyway: In t. Tng. seq of 20.20, just write down t. search space for each prob., along w. its soln. & it may later become clear as to just how t. search space is obtained.

so O.K. Back to 20.20 in t. spirit of 33.01:

t. first 4 items in t. trng. seq. are dealt w. best in 27.29 (1), .33 (2), .34 (3) & 28.02 (4).

I could now go into derbng. t. search spaces, but I want to do this later, after t. problems & their solns. are derbd.

Well, perhaps start out w. a PMTM: For each problem type it is in a known (to itself) mode (oly/model). Abstraction pc's are usable across modes, hvr. — The their pc's can later be modified in each mode when t. (model) SSZ warrants it.

---

Well, back to 20.20:

.12: 27.29 Re: 1) Given this seq. of symbols, to extrapolate it: A human would perhaps try t. freq hypoth first., (like ≈ penny matching), then try cond'l. probys based on t. recent past (≡ Markov Models). These are all "large corpus, Freq-type corpi". Noting that t. symbols were recursively derbable (like t. digits of π) would not be tried.

T. search space mite be t. set of all Markov models, with Markov models of hy pc being considered first. A simpler sub-set of Markov models simply lists various short strings & obtains t. cond'l. pc's of t. symbols to follow them.

If t. symbols in t. seq. to be extrapolated are 0, 1, 2 then t. Markov models, in order of P.C. are 0, 1, 2,

00, 01, 02, 10, 11, 12, 20, 21, 22, 000, 001, 002, etc.

---

Another way to try to extrapolate this sequence is assuming a finite state model of t. process. I think a fair amt. of work has been done on this: Perhaps see review papers by ... Taylor ...

Next, try Z141 ("coding w. defns") or various improvements of it (e.g. non-binary defns (tertiary, quad, etc).

If these more complex search things are tried, I think they will, at present level of TM development, have to be "primitive" (i.e. built into TM a priori).

Note that whether or not TM finds t. "rite model", it will keep on looking for better models, since it can never know it has t. best model.

---

.36 Re: 20.21, .22 (2) (3): These are more or less covered in t. discns. of .12 ff. T. search spaces & t. models are all about t. same.

.01: 28.05 : (4) (20.24): Here TM has $[I_i, O_i]$ pairs. It knows both $I_i$ & $O_i$ are numbers. We can make them real nos. of infinite precision, so if TM is to find $O_i$ as a function of $I_i$, it can only do so by writing pgms, using whatever primative instructions it is gn., w. their (also given) pc's.

So the problem becomes that of guessing t. proper string of instructions.

For many problems, I can list all or almost all of the solns. Hwr, even if I can't by assuming t. solns. I list are t. only ones, t. problem of ~~TM~~ analysing TM's behavior is simplified & is not much difrnt. from as if I could list all of t. solns. w. certainty. By listing t. solns., I can get t. Lcost for TM's finding all of them — Essentially, it is t. Lcost of t. soln. of max Lcost that I'm interested in.

The Unary functions are easy to do: There is just one number as arg., & various functions of that no. are tried in [illegible]

.20 $\approx \frac{cc}{aprip.} = \frac{cc}{pc}$ order.

.21 There could be some point in going into more detail at this time: ~~[illegible]~~ say, if I had a complete set of instructions, t. method of doing controls & ≈ jumps (if any) would have to be decided on. Actually, say t. primative/unary functs were $-x$, sign(x), parity(x), $\frac{1}{x}$. ~~[illegible]~~ To get $|x|$, I could use; (or say "If" was controlable by +1 & −1.)

$|x| =$ If sign(x) then x Else −(x). To get $x^2$ or $\frac{1}{x-1}$ or $\frac{x^2}{x+2}$ I'd have to have binary functions; also, at least t. integers 1 [and maybe 0 so −1 = 0−1].

2 = 1+1 could be defined, (2=1+1 would have much less pc than 1, unless, later, 2 were usd alot). — Then 3 = 2+1 could be defined, etc.

Note that If ... then .... Else is a sort of "funct of 3 args." ...but not exactly. In ≈ Forth $|x| =$ dup If Then Else "−"

Here we start w. x on t. stack, we dup it; If pops stack & decides whether to do nothing or negate TOS. The choices are (1) α (2) whether If Then Else follows α; (3) if If Then Else is usd, what β is & then (4) what γ is.

α β & γ all have to have terminators. "If Then Else" can be t. terminator for α.

5·29·81· TS

01: 20.27 (5) ~~[illegible]~~! / learning names of unary ops. skip this for awhile.

02: 20.28 (6) ~~[illegible]~~ Binary function learning.

Well on second thot, I think it should learn names of unary ops before going on to Binary ops. — Maybe not even learn unary ops w.o. ~~[illegible]~~ their names!

So, t. inputs ($I_i$) consist of numbers and other symbols. The "other symbols" are usd [redacted] to control various operators.

[redacted] Say we are able to put control symbols on t. stack & that "If" means "compare w.t. folg. symbol". So: "If +" would mean, ~~second~~ pop stack & see if ~~[illegible]~~ it is t. symbol "+".

On t. other hand, we may want another kind of "If" w. numerical argts, so t. result could be 3 ways: >, =, <.

So, in fact, the ~~[illegible]~~ If w. non-num. argt. detects only equality or inequality! T. If w. num. argts. detects equality or > or <.

We may be able to use an "If" with only ">" & "≤" — 2 choices. Or, 2 kinds of Num. argtl. IF: (> & ≤) = & (≥ & <).

1:21 [5.30.81] AH··· ~~So~~!···: Befor ~~[illegible]~~ teaching much decision work, have TM. learn logical concepts: ≡(A, A) → Yes; ≡(A, B) → No.

= (10, 3) → No ; = (10, 20 ÷ 2) → Yes.

≡ (10, 11−1) → No, = (10, 11−1) → Yes.

The output (Yes, No) of a logical operator can then be used for control — say control of an "If" branch.

An impt operation is a sequence of conditionals in which control goes to t. first "Yes" in t. sequence. Perhaps this is called a "Case". One common use of this is in "Production Systems" and in a kind of "CF Grammars": in which T. first substitution in a list that is legal, is implemented .... A kind of N way branch of control.

.35 Note on coding for N way (or 2 way) Branch:

We need only have the N conditions in order w. punct. betw. them.

.37 ~~then~~ t. Addresses of where the Branches go to is up to t. O.S. not "TM"! → 37.17

Another kind of operation on t. objects "Yes", "No", are t. Boolean operators.

It maybe possl. for me to work up a good notation for these ideas that would be (1) Good for TM (2) Good for humans in several senses .... not nec'ly, (but possibly) all senses.

One thing I will need, is t. ability to define arby. functions — as pgms or in any other ways. Use of t. λ notation, as in Lisp, is an attractive possy. — My impressn. is that Forth can also do this.

---

An alternate trg. seq. to 20.20ff! Would teach Logical concepts first: Boolean algebra: unary & binary functions. The "Evola" function (20.32) can be first learnd w. logical variables only, (at first). T. pt. of this would be to make it easy to learn "decisions", & "branching" functions.

.17: 36.37: An "If" diffy, not considerd here is how to do loops. One way would be by recursion: to implement:

Define $G(x)$: set $y(x) = x^2$
$x \to x+1$
If $x < 10$, $G(x)$
Else out. ← means do next thing in pgm.

$x = x_1$
$y(x) = x^2$
$x = x+1$
If $x < 10$ goto — else out.

Here $G(x)$ is an operator. It operates on t. vector $\vec{y}$.
$x$ is a param. of t. operator, so $G(3)$ does a difnt thing from $G(5)$.
So this pgm is equivt. to executing $G(1)$.

The idea of asking t. O.S. to do $G(x)$ usually implies recursion is possl. i.e. it's an operation outside t. scheme of 36.35-37.

---

One way to deal w.t. diffy of loops & probably deal w. recursion, is t. use of lables. Every once in awhile, we put a labled pt. in a pgm, so other pts. in t. pgm can jump to it → and/or possibly use lables for srtns also. This would make things pretty much like Ordinary Assembly lang. (the JSR is difnt from JMP, since t. stack & RTN is usd.) - One big difrnce would be that t. "opcodes" (primitive operators) would be carefully devisd.

4/sec. → 32 cps

Actually, these Notation methods will all have to be looked into.

What I should do now is just chose one that seems adequate & extensable, & use it pro-tem. T. fundl. probs. of Trg. seq. construction should be about t. same, in spite of this choice.

Probly best thing to do: write out some solns. for 4) thru 9) (20.24 — .32) in t. manner of 33.01. T. solns. will probly be too complex (by pc). Then use t. ideas of 36.21 — 37.40 (on learning Boolean alg. so that decisions, loops, If's, can be implemented) to ~~write~~ write aux PTS's to bring up t. pc's of those concepts by enuf to be practical. \ partial (or preliminary)

---

.05 O.K.: consider 20.24(4): unary funct. learning: One way I considered: at t. beginning, each subcorpus sc's would consist of a pair $(I_i, O_i)$, with no symbol giving t. operation name — but each sc would have examples of t. same operation: e.g. a sc would be $1,-1$; $23,-23$; $-3^{415}, 3^{415}$; $-1, 1$; ... ;

T. argts would be real numbers of ∞ accuracy. There aren't many simple unary functs on nos: $-X$, $x^2$, sign(x), $|x|$, $\frac{1}{x}$, sin x, cos x, tan x, ~ X (if x is a Boolean integer, each bit of x is complemented) ← Boolean comp. is this called "1's comp of x"?

maybe x+1 & x−1, ϕ, t. constant

Hvr., I don't think this prob. is interesting: The search space consists of unary functs & sequences of unary functs. on reals. [I think ... actually, more complex functs can be obtaind if binary functs are allowd — e.g. $x^2$, x+1, x−1, (x+1)/(x−1) etc. — use of Forth notation can probly easily give us these.

If t. primitive functs are −x, ϕ, 1 (constant), x−1, x+1, sign(x), |x|, x÷y, x×y, x+y, x−y. we can obtain fairly complex functs in forth.

If both −x & x+y are available, we can derive x−y. If x×y & $\frac{1}{x}$ are available, we can derive $\frac{x}{y}$.

|x| is derivable from sign(x),

i.e. |x| = x sign x ; sign ϕ = ϕ, sign +1 = +1, sign −1 = −1. sign(x) is not quite so easily derived from $\frac{x}{|x|}$ or $\frac{|x|}{x}$ because of divisn. by ϕ.

---

If more or less random real nos. are used as argts. in examples, then there isn't much pt. in using > 1 case to illustrate a unary funct. (o.k.) If t. examples are gn. w.o. names, then t. sequence of correct solns. forms this hyer level "corpus", and t. "next trial" for t. next trial t. oprip'd isn't much influenced by this hyer level corpus. — well, it is to some extent: This is of interest. ← except |x| & sign(x) They need 2 examples at least.

.31

TM keeps a ~~record of~~ file of t. solns.: tries them all out on a new problem before trying to create new solns. If a soln has in t. past been usd k times, these past solns. are tried in order of k.

.36

If we use a corpus consisting of a sequence of I, O pairs., & each example is a more or less randomly selected unary function, then t. discn of .31 — .36 can be usd to give a probby of t. next O (O ≡ output).

.01 When we add t. names of t. unary operators, (e.g. −, 3, −3) to t. examples, TM immediately knows that those symbols are not "numbers" & so they must be "control symbols". "Control symbols" are args of Boolian functions — (see 36.21 — 37.40) —

.04 Re: Domain of such Boolian functions is T, F only ......... & is usable to control pgms.

So: Get TM to learn to know what various unary & binary operator symbols signify. We could investigate various functions that don't normally have names (like $\frac{1+x}{1-x}$ or $\frac{1}{1-x^2}$ etc) & give them special symbols.

11:18 26

Because of .01 — .04 TM should find it very easy to learn t. symbols assoc. w. Unary & Binary functions — since only one control symbol appears in each example.

T. next step is learning substitution: examples: $(3\times5)+2 \rightarrow 15+2$ ;

$(3+5)+(6\div1) \rightarrow 8+(6\div1)$
$\rightarrow (3+5)+6$ } — either answ. is acceptable. We have only 1 substitution in each example. Or we may make substitution norm "for t. leftmost subsn. possl."

Next step is recursion — which completes t. "Eval" function!

.18

Re: .01: Knowing things are either "Numbers" or "control symbols" a priori means that data is strongly "Typed" (as in Pascal, say). In fact, data in R.W. is always (or almost always) "Typed" so one knows what kinds of functions can use them as args. — so T.M. usually knows apri. t. type of each data input just as a child does. This ↑, a great deal t. pc of t. functions of t. typd args. —

.22 makes them Much easier for TM to find them.

.23 Actually, this substitution thing isn't so easy. Note that we're using ≈ RPN, so only one thing can be substituted in each expressn.

say $x, y, z, w \ldots$ are Reals & $\alpha^1, \beta^2, \gamma^1 \ldots$ are functions w 1, 2, 1 args resp.

$((((x\ (y\alpha^1)\beta^2)\gamma^1 (z).\epsilon^2)$ here, only $y\alpha^1$ can be substituted for its value.

in $x\ (y\alpha^1)(z\gamma^1)\beta^2)\beta^2)$ either $y\alpha^1$ or $z\gamma^1$ can be substituted.

so, we teach TM to do t. (Rt / left) one ($z\gamma^1$ / $y\alpha^1$) first.

We could use other notations. In such cases, t. trg. seqns may have to be somewhat difnt. — but that's O.K. — so we just would get more experience in a greater variety of trg. seqns.

here re vector:
30¢: •
5¢ inv-
ions.
$10^8$ flips in 1 sec,
for 5¢.
10 sec, $1k =
$10^8 \times 10 \times k \times 20$
$= 2\times10^{13}$ flips
1k flips per trial
$= 2\times10^{10}$ trials

20% int
$3\times10^7$ sec/yr.
1 cent $\times$ $3\times10^7$
so $10^8 \times 3\times10^7$
$= 3\times10^{15}$ flips for 1¢.

$\frac{1}{4}$¢ for w 16 xstrs
so $20\times3\times10^{15}$ flips
$= 6\times10^{16}$ flips per cent.

So just now, I may be able to do unary & binary functs. w. their lables.
Next prob. is how to do substitution
Next prob is how to do recursion

---

I think I discussed a genzn. of substitution in my dizn. of "Strings"
"This substring is a string of type α" : See
282.12 — .20 ← Introduces "string of a certain form"; i.e. idea of substitution.
282.12 — 284.02 ; 285.10 — .20

---

One impt. idea notin ~ 282.12 ff. is t. idea of "equality" ("≡ relation ≡")
That "equality" implies acceptibility of substitution.
Perhaps after teaching names of unary & binary functions, teach meaning of "equality"
In one sense — minus (3) = −3 ; 5 + 2 = 7
In another sense → eq (minus (3), −3) = True : Here we discuss t.
equality of 2 logical variables ; eq ( minus(3), −3) and T .

In another sense, ⇒ if 2 quantitys are =, then either can be sub. for
t. other in an expressn., & t. expression will be equal to what it was
befor t. substitution.

While both are instances of substitution, t. idea of equality-based
substitution & t. idea of 282.12 — 284.02 ; 285.10 — .20 are diffrnt. —
This seems more general. I was thinking of t. "substitutions" used
in CF Grammars a/o in "Production Systems".

The "equality" idea is tied up w. t. idea of "quantity" & t. idea that
an expressn. can have a "value" assoc. w. it.

It may well be that w. human children, they already have something like t. idea
of "Quantity" by t. time that they are to learn arith. & alg. — so that t.
idea has, initially, much hyer pc than it would for a TM w.o. such backgnd.

3 propertys of equality

Another property of equality: Quantitys equal to t. same quantity are = to each other
(transitivity; $a=b, b=c \Rightarrow a=c$), ($a=b \supset b=a$) reflexivity ($a=a$), if $a=b$, $f(a)=f(b)$.

All but substitution are propertys of an equivc. relation : There are 3 propertys of
an equivalence relation: transitivity reflexivity & ... ≡ if a=b then b=a reciprocity?

0 —
180 ✓
90 ✓
180

5
4
3
2
1
0

264
Paid under protest w.o. denying any right to refund of excessive charge payment.

Harvard Trust Co
c/o Cermo

124
Circuit
Rd.
Winthrop
Mass

From a humans idea of "quantity", t. idea of "equality" isn't far — & from these ideas, & human can regard t. idea of "substitution of equal ~~quan~~ expressns" as "reasonable" — i.e. hy pc.

For ATM with much less training than a human, knowing only t. names of various binary & unary functions, t. idea of "substitution" is an enormous heur. jump.

To reduce t. size of this heur jump: One way would be to teach other properties of equality first: like $a=a$; $a=b \supset b=a$; $a=b$, $b=c \supset a=c$.

T. big Q. is — how to we get TM to t. pt. so that substitution of equal expressns within an expressn. is "reasonable" (≡ hy pc)?

$f(a) = f(b)$

$G^2(a,b) = G^2(a',b')$

In human terms: members of equivc. classes are able to substitute for one another in certain applications: for every equivc. class, one should try to find such applicns.

Another approach was 282.12 — 284.02; 285.16 – .20. Here t. idea was t. development of t. substitution concept for use in other applicns, like production systems & cf Grammars.

.19

---

.20 $\boxed{6.2.81}$ Another approach would be to teach TM t. concept of substitu. by giving examples of it, & at giving it a name: e.g.

$$\mathrm{subs}(\alpha, 5, 10, 5x+3-(2+1)) = 5x+3-\alpha$$

an expressn. (α); t. range of positions of t. thing to be substituted (5, 10); 6 7 8 9 10

special meaning of t. word "range".

→ which seems like an awkward notation.

.26 We may want to break it down into 2 kinds of substitu: 1 symbol → ≥1 symbol or >1 symbol → 1 symbol.

---

.27 We will be using RPN, so:

$(((3, 8) + \mathrm{sq}(5\,x)(1, 2\,+)x)$ Here, each unary or binary operation symbol or number indicates a substitutable quantity. I think that's all there are! — so, other than commas, every symbol indicates a substitutable quantity — furthermore, any substitutable quantity in t. expressn. is assoc. w. some single symbol. T. way t. substitution is done; in t. case of ~~Hvr. while this is very fine for RPN.~~ t. numbers, a RPN expressn. can be directly substituted for any number.

.35 In t. case of unary or Binary operators — its more complex. Hvr., there is a way to find t. subexpressn. assoc. w. each such operator. There seems to be a recursive procedure to do this: ① T. range of a Number is itself.

② T. range of a unary symbol is t. range of t. symbol preceding it.

③ T. range of a binary symbol is t. range of t. symbol preceding it and t. range of t. symbol preceding that range.

"Range" is usd in a difrnt way than for functions, which 2 {Domain = input, Range = output}

meaning "plus"? (0682) — or Boolean "AND"?

6·2·81 T.S. 

So we obtain a pair (① & ②) of interlocking recursive defns.

A perhaps ~~to~~ more expensive way to find t. range of ~~a~~ ~~~~ op. symbol would be to

.03 parse t. expressn (Left to Rt.) up to that op. symbol.

---

Using t. forg. concepts it is relatively easy to say that t. operators

$+$ & $\times$ are commutative.

.06 T. associativity of $+$, say is not so easy to express!

$(a+b)+c = ab+c+$

$a+(b+c) = abc++$

so $b+c+ = bc++$ – yes

or $b+c = bc+$ no! ← $a\ b+c\times \neq bc+\times$

or $\boxed{+c} = \boxed{c+}$ ← No!

so t. associativity of $+$ means "+" can be interchanged with its 1st argt.

well --- not always. but $\bullet b+c+ = \bullet bc++$ always : indep of what precedes b.

also $\boxed{\alpha}+c+ = \boxed{\alpha}c++$ just as long as there are at least 2 "objects"

in t. $\alpha$ region. so for $+c+$ we can always xfm it to $c++$,

because $+$ must be preceded by at least 2 quantitys.

On t. other hand, if $c++$ occurs, c must also be preceded

by 2 quantitys at least, so $\boxed{+c+ \Leftrightarrow c++}$; both ways.

now consider $\alpha\beta+c$ & $\alpha\beta c+$

$(\alpha+\beta),c$ $\quad\alpha(\beta+c)$

$\alpha\ b+c\times$
$(\alpha+b)\times c$
$\alpha\beta c+\times$
$\alpha\times(\beta+c)$

$\alpha\beta+c\times \neq \alpha\beta c+\times$

$(\alpha+\beta)\times c \neq \alpha\times(\beta+c)$

$+c \neq c+$

unless followed by $+$

.28 so $\boxed{+c+ \Leftrightarrow c++}$ is always o.k.

.29 similarly $\boxed{\times c\times \Leftrightarrow c\times\times}$ " " "

---

postfix $a-b+c = a+c-b = (c-b)+a = a-(b-c)$
$= c-(b-a)$

RPN $ab-c+ = ac+b- = cb-a+ = abc-- = cba--$

RPM $\boxed{b-c+} = \boxed{c+b-} = \boxed{bc+- = cb-+}$ $ab+ = ba+$

$b-c+ = c+b- = bc+-$ NO

| | | |
|---|---|---|
| $-c+$ | = | $c--$ |
| $+c-$ | = | $c-+$ |
| $+c+$ | = | $c++$ |
| $-c-$ | = | $c+-$ |
| $\div c\div$ | = | $c\times\div$ |
| $\times c\times$ | = | $c\times\times$ |

$c b-+$

$(a-b)-c = a-(b+c)$; $ab-c- = abc+-$ ∴

$(a\div b)\div c = a\div(b\times c)$; $ab\div c\div = abc\times\div$

~~$+c- = c-+$~~ ; ~~$+b- = b-+$~~ :

$b-c+ = c+b- = bc--$

I think there is a kind of duality betw. $(+$ & $-)$ & $(\times$ & $\div)$.

6.2.81 TS 

Getting back to substituting in RPN

say we have $\boxed{a = \alpha}$ — a Number; $\alpha$ an RPN expressn.

If we ever see a in a RPN expressn, we can substitute $\alpha$ for it.

Hvr., if we see $\alpha$ in a RPN expressn, I'm not sure we can replace it by a.

We can: If $\alpha$ is directly follow'd by a unary or binary op.

or If $\alpha$ is follow'd by an expressn that has a value, That in turn is follow'd by a binary op.

what about $\alpha$ a a + + ?

On second thot, I'm beginning to think That whenever $\alpha$ occurs we can subs. a for it!

, 12

(Later ... I'm almost certain This is correct)

.13

| | | |
|---|---|---|
| + a + | = | a + + |
| + a − | = | a − + |
| − a + | = | a − − |
| − a − | = | a + − |

duality $\{ + \to \times,\ - \to \div \}$

.19

distrib law: $a \times (b + c) = a \times b + a \times c$ Alg.

RPN ~~a~~ b c + a x = ~~a~~ b a x c a x +  ; $\boxed{b\,c + a\,x = b\,a\,x\,c\,a\,x\,+}$ RPN

b c + a ÷ = b a ÷ c a ÷ +  $(b+c)/a = b/a + c/a$

$\frac{b+c}{a} = \frac{b}{a} + \frac{c}{a}$

.21

.22

A fairly easy way to Learn t. nomeset / unary & binary functs!

T. trg. seq. consists of things like

Q , A ; Q , A ;

(neg) 3 , −3 ; (sq) 7 , 49 ; etc.

expressn — result

The first order coding is

(neg) 3 , computer negation, result is on stack ; (sq) 7 , seq. of computer ops leaving $7^2$ on stack ; etc.

put these on stack.

~~t. second order cod.~~ t. next order of code defines (gives names to) (multi (computer operation)) functions — so t. 2nd order coding:

(neg) 3 , $\alpha$ , (sq) 7 , $\beta$ ; etc.

$\alpha$ is t. computer "negation" operator

$\beta$ is t. computer "square operator".

For t. 3rd order code, we note that t. only things w. pc ≠ 1 are $\alpha$, $\beta$, etc.

we try to find something They correlate w. to give Them hi conditional pc's.

T. search is rather simple: $\alpha$ always follows (neg), $\beta$ always follows (sq). ... so far ah

$\sin^2 x + \cos^2 x = 1$

~~x sin sq x~~

$\boxed{x \text{ dup sin sq swap cos sq} +}$

$((2+3-1)\times 5)\div 3$

3 − 1 = 1 + 1

3,1 − = 1,1 +

10 + 3 − 1 = 12

10 + 1 + 1

10 ; 3 + 1 −

(10 + 3) − 1

10 + (3 − 1)

10 $\boxed{3\,1\,-}$ +

10, 1, 1 + +

+ 1 − = 1 − +

a b + c − = a + b − c

= a b c − + = a + b − c

a b − c + = a − b + c

a b c + − = a − b − c

− c +

a b − c + = a − (b − c)

a(a − b) + c = a b c − −

− c + = c − −

adequate sized corpus, this is a v.g. code, since code length does not ↑ w. corpus length: i.e.

.82 100% accuracy of predn. w. a fixd length code. (See 82 TS 157.11 for a more detaild exposition of 43.22 ff)

.83 T. forgoing's quite reasonable: Returning to t. Q of how to treat Substitution: Some possl. ways:

1) Make it a primitive op.

2) Using some notation (say whatever notation is usd in) give T.M. examples & teach it what subsn. is — just like learning binary & unary operators in 43.22 ff. A possl. (not v.g. perhaps) notation is suggested in 41.20–.26.

3) ~~Don't give it any particular notation~~ perhaps using t. notation, e.g.

.17 $\alpha,\ 3x+7,\ 43+(2\div\alpha)-18 \rightarrow 43+(2\div(3x+7))-18$

Here, we subs. 3x+7 for α in t. first Left-most occurance of α. 6.50

4) t. notation of 41.20–.26 can be broken into 2 parts: first use t. numberd position to substitute t. symbol ✡ at t. pt. of insert, second, use .17 to substitute t. desired expressn for ✡.

Note: this is only 1 → ≥1 subsn. ; t. ≥1 → 1 subsn. is not yet treated.

≥1 → 1 subsn. is what is needed for "Eval".

5) Substitution (both many ↔ 1) seems to be particularly easy to implement in RPN: (see 43.12) T. main advantages of ordinary alg. notation is that the assocy. of "+" & of "×" are automatically integrated into t. notation. w.o. that feature, in RPN, t. assocy of + is completely gn. by

+a+ ⇔ a++ ; for Mult: ×a× ⇔ a××. (42.28–.29). "a" can be any evaluatable expressn. (i.e. an expressn whose value is a number.)

for + & − (or × & ÷) see 43.13–.19.

1 dot = $\frac{19}{3.1}\times\frac{5280}{21}$ = 1541 ft. = 93 rods, 325 cm. 21 lots.

6) I think substitution is also easy in ordinary Alg. notation. (1 ⇒ many is trivial)

many → 1 will always work, I think if t. expression in which one substitutes is completely parenthd:

1 + 3×4 = 13 ; 1+3 = 4 hvr. 4 ×4 = 13 is false.

On t. other hand, completely parenthd alg. notation doesn't give t. automatic assocy of "+" & "×", & its really very close to RPN — I think I devised a simple grammar for xfmg. one into t. other.

Still, it seems that substitn. is easier in RPN than in completely parenthd. alg. — because in RPN every substitutible expressn. corresponds to one oprtr. A very nice, exhaustive 1 ↔ relationship of symbols in t. RPN expressn & v. versa.

— See 41.27 on this: also 43.12 13, I'm almost sure correct. T. subexpressn designated by any operator in a RPN expressn is gn by 41.35 & routine derbd in 42.03 ← actually 2 possl. routines.

7) [redacted] Try to derb. various applications of subsn. See how each applicn. could be taught. See if t. ~~various applicns~~ Trg. seqns (i.e. sub abss.) leading to each of t. various applicns. have sub abss. in common.

.04 8) Give TM ~~a~~ problems in which substitution is t. correct soln. From this, using reasonable primitive instructions, try to compute t. pc of subsn.

One possl. way: It would seem that using RPN, $1 \to \geq 1$ subsn. would be easiest to derb., ∴ larger pc. After this is learnd, try $\geq 1 \to 1$ / or $\geq 1 \to \geq 1$. subsn. Superficially, this last would seem to involve t. concept of recursion, since t. substituting expressn could be of arby length.

Try teaching it for $\ell=1$, $\ell=2$, $\ell=3$ . . .

First teach $1 \to 1$ subsn., then $1 \to \geq 1$, then ~~∃~~ maybe digrams $\to$ 1 or $\geq 1$, ~~Trigram~~ trigrams $\to$ 1 or $\geq 1$

.19 4gm $\to$ 1 or $\geq 1$ then ngm $\to$ 1 or $\geq 1$ for arby n by recursive defn.

---

.04–.19 / (8) seems not bad to start with: Then maybe [redacted] .01 (7).

(8) could be taught after certain unary & binary operations are recognized.

.22 Say ~~sbs~~ subs. $\alpha, \beta, \gamma \to \delta$ is t. form of t. problems of .04–.19.
(name of operation to be learnd. ; special symbol: TM knows what this means.)
sbs

~~$\alpha, \beta$ are~~ ~~any~~ ~~expressn having evaluable expressn in RPN,~~

~~$\gamma$ is any string containing~~

$\beta$ is to be subs. for $\alpha$ in t. [redacted] leftmost occurrence of $\alpha$ in $\gamma$. if no such occurrence, then null output, or E (error) output. ?

actually $\alpha, \beta, \gamma, \delta$ need not be RPN expressns. They can be genl. strings.

$\alpha$ & $\beta$ are strings; $\gamma$ is a string containing $\alpha$ as a substring. $\delta$ is t. result of substituting ~~a~~ $\beta$ for $\alpha$ in $\gamma$.

---

.28 We start w. $\alpha$ & $\beta$ being both 1 gms. (≡ single symbols).

~~At~~ AH! We mite get a need for recursion here! Say $\alpha \equiv a$, $\beta \equiv b$!

$\gamma_1$ = ac[redacted]

$\gamma_2$ = cac[redacted]

$\gamma_3$ = ccac . . . $\gamma_n = c^{(n-1)}ac$

So, we first teach subsn. for ~~strings~~ $\gamma$ like $\gamma_1$ in which a is t. first symbol

next we " " " " " $\gamma_2$ " " 2nd " .

etc.

At a certain point, TM should get t. general idea, using a recursion formula!

It is very interesting that we can get useful recursion learning at such an elementary level of problems! — [redacted] TM can learn this sort of thing before learning unary & binary functions.

.02 O.K., so after 1→1 subsn., we learn 1→2, 1→3, ... 1→n (via recursion)
Then, we learn 1→1, 2→1, 3→1 ... m→1 via recursion,
then maybe m→n by quick synthesis!

After this kind of substitution is learned, we teach unary & binary functions.

Def

.06 Then evaluation of general alg expressions (EVAL) should follow — but I'm not so sure that the conceptual jump size is reasonable yet. If has the learned sbs function w. 3 string args. of 45.22 How it would apply this to alg expressn. evaln. is not clear yet.

Well, say TM sees the expressn. γ: 3,4,+,8,X ; It scans from Left to Rt.

When it sees +, it naturally computes 3+4=7, so 7 is "around" as a hype object.

That TM should then try sbs 3,4,+ ; 7 ; γ is of reasonable pc.

This gives 7,8X. TM could then evaluate this as 7×8 = 56, which is the final, correct answer.

This, hvr, is w.o. recursion, & its pc is rather low because its the product of the pc of this and this. Yet it is probably high enough so its Least is acceptable.

I can however (at last) see how this will work.

None the less, It may be, that using higher level codes, one could get to the point where recursion was, indeed, the best pc soln. — & that it was very clear that this was a good soln.

.23 Consider the function on strings: Σ ≡ "evaluate the leftmost available function & substitute it in the string". Then Σ to evaluate the string α = 3,4+8X

ΣΣα would do the trick. For longer strings ΣΣΣ or $\Sigma^{(3)}$ would be needed. So in general, the code for the soln. would ("information-wise") [in addition a constant indep of the string to be evaluated] contain only the integer n, & its pc. would be the pc of the integer n.

The problem, then, is to find the "stop rule" for the repeated applicn. of Σ to the

.30 string in question. One obvious criterion is to stop when Σ can no longer be applied (i.e. there are no "non-numerical symbols" (≡ control symbols)). Another is to stop when the result is a pure number — since solns. have (in the past) always been pure numbers, — which amounts to one doesn't have to remember that all solns. in the past have been pure numbers. both

In general, the method of inducing recursive rules must be worked out — & in particular, the method of inducing "stop rules" for the repeated applicn. of an operator.
Hvr. .30 isn't bad i.e. when there are no control symbols left. Another way might be for TM to count the control symbols & make n = to them. Hvr. noticing when none are left after successive reduction of them is equivalent to "counting".

This is beginning to look good! I will be able to get pc's (& ≈ Lc's) of solns & I'll be able to see how pc's of solns. change when I ommit (or add) various examples (or PTS's) to t. tng. seq.

---

O.K., so now lets go back to 46.02 to see how 1→1 subsn. is learnd, then 1→2, 1→3 ... etc. then 1→n subsn. by recursion.

So: look at 45.28 ff:

**So** ; in RPN: a, b, ac, sbs. , maybe notation γ, β, α sbs would be better! so ac, b, a, sbs. → bc.

At this point: what are t. primative operators available to TM for writing these pgms?
— well perhaps keep them in "English" for a while.

~~Also look at 45.28 for 1→1 subsn. learning: I think recursion may be useful~~

.17 First learn $\underset{a}{\gamma}, \underset{b}{\beta}, \underset{a}{\alpha}$ sbs → b for various values of a & b.

T. soln. is always simply t. 2nd arg., b.

perhaps try 6, b, a; sbs → Λ (null......) &c

So: soln = 2nd arg. unless 3rd arg. ≠ first arg. in which case soln = null.

A rather complex thing to learn: perhaps leave out t. "null" effect until more of t. positive stuff has been learnd: so to start γ alway = α. & soln is β.

& this would be easy to learn.

.23 Next ac, b, a; sbs → bc. soln: easy by "correln"

Note that in both .17 & .23: t. first symbol of t. soln. is b. T. 2nd symbol of t. soln. is t. 2nd symbol of γ.

---

poss. simplifns to learn a, β, γ sbs: here α is allways a in all examples to start off. — Also make β always = b.

.30 in, say rsapqrb, b, a; sbs → rsbpqrb We note correlns. for all but (in this case) t. 3rd symbol.

→ by making soln = γ, but with 1 exception, we do have a hyp pc coding method.

$\boxed{\text{SN}}$ so far my main (if not only) coding method has involved "correlation". — i.e. we note that certain symbols of t. corpus are identical in certain other symbols.

.33 Anyway, here the exception symbol is always (in γ), a ; & its corresponding symbol in δ is alway b. If we use these 2 ideas as primatives, then this may be an adequate soln. T. 2 ideas: "correln." & "exception".

.36 $\boxed{\text{SN}}$ we may allow >1 subsn. of b for a in γ i.e. all occurances of a in γ.

O.K. so perhaps via .33 we can do all a→b substitution. Then we learn a → c subsn. then a → d subsn. . Then .... a → any symbol subsn.

So: Does this work?

Lets look at 47.30 — .36 : Is this an adequate soln. of this simplified "substitution" problem?

Just what primitive operations are usd? | $\gamma, \beta, \alpha$, sbs $\rightarrow \delta$

Well: T.M. recognizes t. difrnt. args. $\gamma$, b, a, sbs $\rightarrow \delta$

($\gamma, \beta, \alpha$, sbs, $\delta$) & it knows that it has to find $\delta$, given t. other args.

Then, it tries to "correlate" t. symbols of $\delta$ w. those of each of t. other 4 args. If finds in t. case of that ① at least 1 symbol of $\delta$ is b, (1.5) " " " " " $\gamma$ is a ② No symbols of $\delta$ are a ③ All symbols of $\gamma$ & $\delta$ are t. same, except 1. In $\gamma$ this symbol is a; In $\delta$ it is b. → o.o.o.o ←

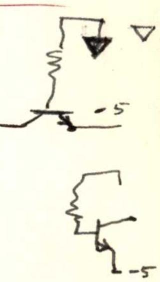
6 pecas = $11\frac{1}{2}$' ! sow 2'/pcc.

Or: In going from $\gamma$ to $\delta$ ; if a symbol is "a" it maps to b; if it

.12 is $\neq$ a, it maps to itself.

This seems fairly simple to learn: but can it then learn it for arby $\alpha$ or arby $\beta$

.14 Then arby $\alpha, \beta$ ? In t. case of general $\alpha, \beta$ ! TM will first note that $\gamma \approx \delta$ for all but 1 symbol. Then it will note that the "non equal" symbol in $\gamma$ is $\alpha$; & t. "non equal" symbol in $\delta$ is $\beta$.



.19 This could solve it. Now: just what are t. primitive operations involvd? "Correlation" TM compares $\delta$ w. each of $\alpha, \beta, \gamma$ to look for systematic correlns. Are any parts t. same, or systematically functional (e.g. like (v→u always)?

.23 Are any parts usually t. same or "systematically functional".

.24

from .12 : If a symbol of $\gamma \equiv \alpha$, Then t. corresp. symbol of $\delta$ is $\beta$.

" " " " " $\neq \alpha$ " " " " " " t. same as t. $\delta$ symbol.

This looks like a complex rule to guess....... This does seem close to .14

.27 ⇢ 49.01

.28 : 80TS 288.29 SN On assigning very hy pc's to new defns: One way is to ↑ ssz. by recoding t. past. Say one has an "acceptable" code for t. past, w. total pc = $P_0$

Now, using t. new defn. to code t. past, one gets $P_1$, & even if $P_1 << P_0$, t. new defn. will give an alternate code for t. past that may be of much interest & can give a very hy pc. to this new defn. — even tho t. parallel code resulting from it is not one of t. hyer pc codes of t. corpus.

However, check this latter reasoning! I'm not sure its correct!

...... → It my be that t. recoding of t. past must have a pc > or $\approx$ t. pc of t. previous existing code befor t. new defn. can get any "points" via this trick.

— For more on this: 59.04 — .10; 59.11 — .27

6-6-81

TS



.01 : 48.27 A possl. way to write this role:
for $i = 1$ to --- N

If $\gamma_i = \alpha : \delta_i = \beta$ else $\delta_i = \gamma_i$

Next

$i=1$ If $\gamma_i = \Lambda$ End

else If $\gamma_i = \alpha : \delta_i = \beta$ Else $\delta_i = \gamma_i$ goto

.07 1$\phi$ $i=1$

.08 2$\phi$ If $\gamma_i = \Lambda$ end Else, If $\gamma_i = \alpha$ ; $\delta_i = \beta$ : Else $\delta_i = \gamma_i$, GoTo 2$\phi$

How to write t. pgm at hy pc? "$i=1$" & "If $\gamma_i = \Lambda$ end, else" are hy pc parts. — also t. Go to 2$\phi$.

T. low pc part is $\boxed{\text{If } \gamma_i = \alpha ; \delta_i = \beta : \text{Else } \delta_i = \gamma_i}$ $\left(\frac{1}{4}\right)^6 = 2^{-12} = 4 \times 1024 \approx (4000)^{-1}$

pc's $\approx$ $p_1$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{4}$ { t. position of this "Else" is t. $\phi$.

a dummy variable.

$\left(\frac{1}{4}\right)^6 \approx (4000)^{-1}$ — a rather large pc. — but there are other factors to reduce it.

$10^9 \approx 2^{30}$

.20 Hvr., t. pgm of .07 - .08 doesn't capture the hy pc aspects of heuristic Rot!
The reasoning of 48.14 should ↑ t. pc of soln. much. This reasoning looks

.22 like (& perhaps is) a "Plan" that results in t. propose "pgm" (≡ "soln."). → 51.11 →

First compare $\delta$ w. each of t. given args. $\alpha, \beta, \gamma$. Are there any

.23 good similarities (see 48.13 - .23) ("correlns") or "systematic functionalitys"?

.22 or .24 So: one codes $\delta$ as $\gamma$ to start, then looks at "errors".

We have to ① find t. symbol that is in error

② " t. correction for it.

for ① T. symbol that is in error, will always be $\equiv \alpha$ This can be discovered by noting t. erronious symbol each time, then trying to correlate it w. t. args $\alpha, \beta, \gamma$. ② T. replacement for t. erronious symbol will be found in a similar way, to be $\beta$.

From t. [illegible]., it is natural to see if being $\alpha$ is a suff. criterion to being an erronious symbol (it is) & to then to

.30 replace it w. $\beta$ — which solves t. problem "in English".

I'm not sure of all of t. details of this reasoning. Could I really formalize them exactly or are there some more tricks that I haven't written down yet? — "ideas betw. t. lines" ☺.

.22 - .30 seems o.k. — let it go for t. present.

Say we can now do $1 \to 1$ subsn. Look at 45.28 ff for $1 \to 2$ subsn. etc.

Well, perhaps not rite now. 45.28 has an alternative way of learning $1 \to 1$ subsn, perhaps. — but I'm not sure ... it seems to need a much larger SSZ than .22 - .30.

Well, what about 1→2 subsn. now?

Since we use t. same function name, sbs., it's natural for TM to try small modifns. of t. defn. for 1→1 subsn — Hvr., it doesn't work very well.

We make $\delta \to \gamma$ for all symbols but $\alpha$; Then we try to replace $\alpha$ by $\beta$, but $\beta$ has 2 symbols in it, so we can't do t. replacement.

However, we can try t. same "PLAN" (cf 49.22—.30). (It succeeded for sbs for 1↔1 so we try same plan for other argts): Good idea!

.10 "Comparing" $\gamma$ & $\delta$; we note that $\delta$ consists of $\gamma$ broken into 3 parts where t. $\alpha$ symbol is: $\gamma_1{}^\frown\alpha^\frown\gamma_3$ Then $\delta = \gamma_1{}^\frown\beta^\frown\gamma_3$.

We'd (perhaps) like TM to have some string manipulation facilities so it could notice such things. $\gamma_1$ & $\gamma_3$ can be found by "correln" w. $\delta$.

.13

In fact, n→m subsn. can probly be devd. by ≈ .10 — .13, directly.

.15 Start out by matching t. first symbs. of $\gamma$ & $\delta$. continue along matching until they don't match. Then $\gamma$ has $\alpha$ in it next, & $\delta$ has $\beta$ in it next, Then

.18 t. rest of $\gamma$ & $\delta$ match.

Perhaps something like .15 would be more readily devd if TM had an operator for concatenating strings.... concat seems relevant, — but I don't see just how to do it.

Perhaps a de concatn. operator. There are some in most BASIC's.

$\boxed{6.7.81}$ Note that t. methods of .10 — .18 may work w. 1 subsn. instance of $\alpha$, only. e.g. if $\gamma$ = a c d $\alpha$ c a b $\alpha$ d d c (2 subsn. instances of $\alpha$), it isn't so clear.

Also note that t. idea of "segmentation" & "desegmentation" for 1, 2 & 3 dimensions & certain kinds of parsing, may be (wired) built-in primitives for humans.

| |
|---|
| 5280 = 16 × |
| 10 × 33 |
| = 320 Rods |
| = 320 × 16½ ft. |
| = $2^5 \times 3 \times 5 \times 11$ |

If we want to treat "Leftmost subsn" as being t. obj. of interest, Then we can get complete subsn. w. a recursive loop. Hvr, Then we have to get t. idea of "Leftmost" in. We could use L to R. examination of strings.... Then t. "first instance of $\alpha$" would be its leftmost instance.

.31 Re: A simple kind of recursion: to "do operation Σ repeatedly until its impossl., Then stop". One way is to have t. "impossibility criterion" (i.e. stop rule) built into Σ — in which case t. "Repeat Σ until impossl." may be regarded as an operation w. Σ & a string as arguments. T. stop rule "built into Σ" is often ad hoc since its based on a certain operation becoming meaningless. Actually, we can have an operator leave its argt. invariant if t. operation is meaningless. In this case, $\Sigma^{(\infty)}$ will automatically have this natural termination — since t. operation converges when $\Sigma\alpha = \alpha$. To save cc, hvr., we have a special "watcher" that looks to see when $\Sigma\alpha=\alpha$, so it just doesn't continue applying Σ again & again & wasting cc.

Substitution: The methods of 50.10 → .18 to find $n \Rightarrow m$ subsn. should be available to any reasonable TM. Just what detaild mechanics are usd is unimportant for t. moment. I think TM should be able to recognize when $\alpha$ is a substring of $\gamma$ — for any $\alpha$ & any $\gamma$. In t. envt. of $\beta \alpha \gamma$; sobs → $\delta$, it should be looking for such "substringisms". 1.0000010 93
Whether it does this by looking for "correlns." or by other primitive operations, need not be decided now. 145

Its certain that at some level, sabsta t. concept of "substitution" would be defined — so I could just plug it into TM & then try to make estimates of its pc. 927 654

.11 : 49.22 One of t. pts of interest: In 49.08 → .20 I made a Ruff estimate of t. pc of finding t. sbs. operator. I got pc ≈ $(4000)^{-1}$. Hvr. it was clear that this particular search was missing most of th. heur devices (including "Plans"), & so it ended up w. a much smaller pc than any reasonable human search would have. $n=10$ $x=10.6$ $\Sigma=106$ $\sigma_n$ 18.548 19.5516 $= \sigma_{n-1}$
Note t. discn. of 49.20 ff, on this point. → (52.26

.17

Perhaps an impt. idea, is that if some of TM's soln's of problems have much less pc than a human soln seems to have, then clearly TM doesn't have some of t. abs, heurs, or whatever, that were usd in t. human soln. — & TM should be given them — either by direct definition or by a pts. So, th. "substitu." deriv problem can be regarded as a challenge to me! can I devise an adequate set of heurs? It can be regarded as a "study prob." — since I'll have to do this sort of thing repeatedly, for more complex problems.

Well, is 50.10 → .18 "adequate"? It looks O.K. superficially, but I haven't put in t. detaild. operations involvd. If I did, then I expect I'd get as hy a pc as a human solving t. problem.

working w. Grace: suggs. folg. methods:

.30 1) $n_\gamma - n_\alpha = n_\delta - n_\beta$ ($n_\Sigma$ is t. no. of characters in $\Sigma$).
which suggests that if $\alpha$ is a substring of $\gamma$ & $\beta$ a substring of $\delta$, then $\gamma$ "−" $\alpha$ = $\delta$ "−" $\beta$ — which turns out to be true & does suggest a final subsn. rule that's correct.
unfortly this eqn. can be writtn many ways e.g. $\left\{ \begin{array}{l} n_\gamma + n_\beta = n_\delta + n_\alpha \\ n_\gamma - n_\delta = n_\alpha - n_\beta \text{ etc.} \end{array} \right.$
— most of t. ways are not as suggestive.

$n_\delta \geq n_\beta$ ; $n_\gamma \geq n_\alpha$.

② → See 50.10 → .18 for heur discn. leading to → can we break t. 4 strings into sub-strings
.37 in a informative way? One way is $\left\{ \begin{array}{l} \gamma = \epsilon^\frown\alpha^\frown\pi \\ \delta = \epsilon^\frown\beta^\frown\pi \end{array} \right.$ any string can be a null string
.38

This breakup implies this eq. & has uniformity more info. | $\epsilon$ &/or $\pi$ can be a null string. We MAY have to allow $\alpha$ &/or $\beta$ to be null strings!
When this eq. is discovered, it is very close to final soln. All that is needed is that we find t. relevant $\alpha$ is t. left-most occurrence of that $\alpha$ in $\gamma$.

6.7.81 TS



SN A method to generate stoch examples for sbs learning:
use poisson distribn. lengths for $\alpha$ & $\beta$ & $\gamma'$. Then generate a random
no. betw. 1 & t. length of $\gamma'$ That gives t. pt. at which $\alpha$ is inserted to create $\gamma$.
From $\alpha$ $\beta$ & $\gamma$ we get $\delta$, to give t. example:

3) t. "correlations" used in 50.10 — .18 are unclearly defined in my mind: I should
clarify this — it seems that these are impt. concepts that I'll want to use
.07 (.07) on many other elementary concept discovery tasks.
other.....

My writing out these possl. ways to solve this concept discovery problem
is mindful of Newell & Simon's work — Of giving humans problems
(like cryptarithmetic) & asking them to say how they were trying to
solve it (into a tape recorder) while they were trying to solve it.
They then would make computer models to try to simulate t. human prob. solving
behavior.

By doing this for a variety of prob. types (I don't know how
much variety N & S have in this area) one would get some idea
of what some basic prob. solving methods mite be .... &
t. "primitive concepts" assoc. w. them.

I mite want to do approximately this for variety of example concept devry
problems — to develope a powerful vocab. of primitive concepts

.26 6.8.81 : 51.17: I think I will want to try to get t. solns. w. pc's ~ human pc's for soln.
This makes it more likely that I have a set of abss. that is adequate
for t. jobs I'll be interested in. I could start out w. relatively low
pc solns expressed in Forth: This would perhaps be of some interest —
to see just how pc's are initially assigned, & then modified to as
t. ssz ↑. However, I suspect that t. "correct" (≅ human) solns
to most problems are not so easily expressed in Forth: That
t. method used by humans uses "plans" & various sub-search techniques
that I don't yet know much about — & that I'll really have to
know these things before I can usefully design workable t.s's.

.01 Heuristic (noun) i possl. defn: A heur. is an abs. that ↓ t. Lcost of t. corpus. It can do this by ↓ cc or by ↑ pc or by both.

A good "plan" will be a heur. It will do this by ↓ search cc ~~by assig~~ — usually by assigning high conditional pc's in a way that leads to a hy pc soln. — Often, the things tried will have very large cc (e.g. some elaborate obs. must be taken befor t. relevant op. is executed), but they will have ~~so~~ large enuff pc so that $\frac{cc}{pc}$ is relatively small.

Actually, .01 doesn't define what one intuitively means by "heuristic". T. intuitive defn. is probly directed toward abss. that are rather general, i are expected to be applicable in many sc's.

A good plan (one often yielding a ~~hy~~ low $\frac{cc}{pc}$ for a sc) will get hy pc. for its name, because it is often used in successful codes — ("successful" means "included in t. finally decided upon code of t. corpus"). ↳86.10 — 1¢ 13/mo 150 stories/yr.

.17

.18 : 81TS270.04 — This doesn't seem to exist! Mayb 80TS270.04

**SN** on "Sequencial Coding": Say we use a corpus broken into $sc_i$'s. When each ~~corp~~ $sc_i$ is gn. we invoke a "plan" to decide what to do about it. The pc's for t. "plan" i for t. coding techniques t. plan uses, are obtaind from t. code (thus far) of all of t. $sc_i$'s thus far (not including t. present one). This is t. idea of 267.29-.40, but perhaps more general. T. present idea is that t. code of t. corpus ~~up to~~ up to and including $CS_{i-1}$, is usd to ~~calc~~ calculate t. apripd. for $sc_i$. In 267.29-.40, this is elementalized a bit, in that ⟨statistics of abss. (~~pc~~ e.g. pc's) that are usd in coding t. corpus up to $sc_{i-1}$⟩ are used to form t. apripd of t. abss. usd to code $sc_i$. ↳ See 56.01

.27 The objection to this "sequencial coding" idea in t. past, was that e.g. linear regn. coding would never be tried for a sc. Hvr. if one uses "plan"s as one of t. abs. types to code a corpus, I think that linear regn. coding would be tried. In t. first place, I think "plan" coding would make it possl. to devr. linear regn. codes in t. first place — for short sc's. — Then, after this initial success it would be usd for longer sc's. — possibly partly valid, but note .32

.32 Well, I think there is some confusion in .18 ff: It involves (in part, perhaps) t. distinction betw. t. 2 kinds of sc ~~corp~~ i: ① T. short sc (29.25), in which one tries to code directly using cBl. & Lsrch. ② T. long sc. (29.26) like a long numerical TS., in which one trys to find a good plan via, perhaps Lsrch.

T. linear regn. code is for t. 2nd kind of problem. T. discn. of 269.29-.40 ff may be mainly for t. first kind (short sc) of problem.

6.9.81 Tk                                        Phoned out from ~ Thursday to ~ 1 3 D later! Wed 6.10.81 ~6pm        54

**PROTEM**

.01 What I may do, protem, is retain t. distinction betw. t. 2 kinds of sc's, and treat each set of sc's separately viz ≈ 267.29ff ⟨ i 53.18 —.27⟩ ~~using~~ pooling statistical data on some abss., & keeping separate statistics on other abss. (depending on ssz's in each set of sc's).

One essential difference betw. t. 2 kinds of sc's is that in Pem coding, t. Pem is allowed to look at ~~t.~~ entire corpus before coding it. In simple CBI (short sc coding) ~~t.~~ TM is only allowed to look

[ a short distance past t. part of t. sc already coded.
This "short distance" (if it exists) corresponds to t. radix size being used.
( I'm not sure this "short distance" idea is correct).

~~1.414 213562~~
~~1.4142 72 136~~
$\frac{\pi}{2}$ = 1.570796327

.15 **A** sort of ~~Gonzh~~ of **53.18 —.27 :** T.M. is solely ~~completely~~ an operator (I, O) device (— w. order of presentation of I, O pairs being considered).

$\frac{e}{2}$ = 1.359140914

The Input will consist of t. problem descr. in some agreed on lang, & Output is t. soln. to that problem. — problems can be of any type.

If t. problem is an induction problem] (i.e. a sequence extrapoln.) t. Input will be a statement to that effect, followed by t. sub corpus to be extrapolated. This "sub corpus" will be understood to be a continuation of previous sc's of this type.

If t. problem is a "large sc", like a numerical time series — Pem coding: like 29.26ff. needing linear or n.l. regressn., this also, will be stated, followed by t. ~~sc~~ sc to be extrapolated.

**In** all cases, the Input is fed into TM, which usually will apply a "plan generator" to it — as a standard method of coding it.

"Obs" are always parts of "ops", but will have pc's of their own. Essentially an "ob" is always part of a structure to control other obs &/or ultimately ops.

(A / The) final code of t. entire corpus is this set of codings of t. sc's.

The total pc of these codes is t. pc of t. Output w.r.t. t. Input — It is a conditional pc. — like "conditional Entropy" etc.

6.10.81 T. form. is a lot like (perhaps identical to) **PMTM** — but the mechanics & t. formalism desired is perhaps clearer [ PMTM also has an "Advice" input — which .15 doesn't have ]

— A possl. way to deal w. an ~~data~~ "advice" channel, using t. .15 formalism: Say the "advice", is t. string α, & it applies to $sc_k$. We concat. α (w. suitable punctuation to $I_k$, so TM knows α is advice for t. $k^{th}$ problem. As such, t. soln, ~~$O_k$ becomes the~~ we want t. $O_k$ of hi-est conditional probby w.r.t. $I_k$⌢α.

An example of t. coding approach of 54.15 ff: Say we are working problems that are amenable to t. GPS type of approach. We first apply t. "Plan" operator(s). There mite be several of these (In 1980 I wrote about several that I had considered ... among them GPS, then a modifn. of GPS by Slagle (≈ Multiple a/o SAINT), then a kind of Plan or 2 that I had worked out.)

So say these plans all have their own pc's — say that GPS has t. largest, so we start w. it first. It looks at t. problem, & if it is of suitable type, it brings out a set of "differences" appropriate to that problem (this is a pc=1 operation). The "search" in normal GPS is completely deterministic (occasionally there may be choices of equal attractiveness, but this is not an impt part of t. system). Probably if t. choices in t. search were assigned pc's (i.e. conditional pc's usually), we could do a cheaper search using Lsrch.

In GPS, one also needs difference reduction operators. These were devised by Newell & Simon on t. basis of logical reasoning — mathematical analysis of t. available operators & combinations of them. TM could do this sort of thing (an advanced TM, that is) but an elementary (naive) TM would probably find these difference reduction operators by experimenting & standard statistical analyses of t. expts — thus yielding often pc's ≠ 1, rather than certainty of certain difference reductions (pc=1).

→ this logical reasoning "method of deriving difference-operator pairs would be an impt ~~discovery~~ methodological discovery of an advanced TM of this type.

Actually, t. pc's involved are not {that t. "desired differences" would ↓} {t. probabilities}, but that this operator would be instrumental in solving t. problem.

In my previous discn. of GPS, I think I did run into t. problem of just what these pc's were t. probabilitys of. I think t. approach of 54.15 clarifys this!

More exactly, the obs & ops need not have pc's (t. idea of pc for an obs. is an elementalizn used in t. Z141 model of induction {"coding w. defns"}). The total corpus preceding t. present corpus induces a apriP on t. possl. solns. of t. present problem ... in t. spirit (organizn) of 267.29. Pc's occur if t. induction model usd is Z141, & probably in some other models ... but need not occur, in general.

6.11.81 TS 

.01:53.27 In t. case of simple digital [redacted] sequence extrapoln., t. idea of "Sequencial coding" is this! After I've made several codes for t. corpus up to t. corpus symbol $x_i$ (t. $i^{th}$ corpus symbol), this set of codes induces an apriod on t. possl. sequences of following symbols. The codes continuing for these possl. corpus continns, is usd to ~~guide~~ give pc's to control t. Lsrch for ~~continuation~~ code continuations that match t. empirical continn. of

.07 t. corpus.

Note that p costs [redacted] rather than $2^{-\ell}$ ($\equiv 2^{-\text{brest}}$) is better to use for Lsrch — i.e., it results in cheaper searches.

T. parp. (.01 ff) means that one saves t. best 100 codes (or best 10 codes ... depending on how long or "factorable" they are) only ... so backtracking is impossl. — (if one didn't save an impt. code needed in this new problem).

.12 On t. other hand, recently, I decided that recoding t. past (or part of t. past) mite be a good way by which [redacted] (humans as well as) machines could justify a very hy pc. for t. name of a newly defnd concept. — So maybe I could integrate these 2 opposing ideas. see .29 ... 1.70 ms

.20 Re: "Sequencial coding": When it's usd, after each problem is solved one impt aspect of "updating" is to change t. params of t. system so that they reflect, as much as possl. (perhaps necessarily perfectly) all t. problistic info obtaind from t. corpus up to now. If Z141 is usd [redacted] much (or a modifn of it), this will take t. form of making new defns., & modifying pc's of old ones. (also a new defn. can be usd in solving a new problem as part of Z141). Modifying pc's of old ones (or even

.29 new ones, can be done, (in addition to t. updating period) during "meditation", by recoding t. past.

.31

.32 : Just how does .01 — .12 differ from t. naive concept of sequencial coding? Well, maybe, if .20 — .31 is considered, that one is allowd to use coding methods other than simple direct coding — e.g. Bern seq. & Z141 can be usd in a ~~[illegible]~~ more ec/economical way — also, perhaps even lin. reg. coding (or any other pam coding method) if. like 29.26 ff)

6.11.81 TS 

.01 Q: To what extent are <codings up to a certain pt. of t. corps> completely expressable as pc's (or conditional pc's) of various abss.?

Well, maybe not "all or maybe not completely expressable" — but perhaps all ~~to~~ humanly derbable ones are! Tho, perhaps "humanly derbable" is too narrow an idea — I'd

.05 certainly want to include subconcious methods

.06 Well, 56.20 – .24 plus 57.01 – .05 is, I think, a critical Q:

can I express any induction methods that I can think of, conveniently

.08 (low cc) as ~~p~~ conditional pc's of various Abss?

It has been shown that all pgms any proby distribn. is expressable by CBI (& conversly). Just what is t. ~~theorem~~ Theorem to be proved in .01 – .08 ?

.12 T. operations ~~[illegible]~~ involved are, (≠), / conditional stochastic branch / plus ~~[illegible]~~ all other computer insts.

T. $\boxed{\text{conditional stoch branch.}}$ This means that we can simulate a stochasticly pgmd ~~[illegible]~~. Pgm Monte-Carlo-wise, but instead, we may do Lsrch if we like, since we can obtain t. numerical pc of each possi. pgm.

This is an N way branch w. n ~~[illegible]~~ pc's assigned to t. branches ($\Sigma p_i = 1$)

It is "conditional" in t. sense that t. values of $\vec{p}$ depends on conditions befor t. branch

.23 was taken & ~~can~~ vary w. those conditions. → 50.03

Now it would seem that .12 would be a method to possibly ↓ cc of various search operations.

I.e. Dewo & Shannon in "Automata Studies" showed that stochastic Automata weren't any better than others in a certain respect; But proby not in cc for certain kinds of things. & of course there is Rabin's result w. random pgms for finding primes.

One known result is that if pc. of 0 = pc. of 1, = .5 then t. resultant pgms fed into a Umc give t. correct universal proby distribn. Hvr. its true if t. proby. ~~of~~ for 0 is any thing > 0 & < 1. Anyway, this is a simple example of a stochastic pgm.

Another kind of stochastic coding is that assoc. w. linear regressn. — but just how this fits in w. t. forgg. is unclear.

$\boxed{6.12.81}$ There are ~~[illegible]~~ many difrnt. ways to assign a pc to a corpus. Some of these ways are universal, others are not. While every Pgm can be mapped to a coding method — (or a specific machine), this sort of "equivalency" is for t. resultant proby distribn. only, & does not consider cc's which may differ considerably.

The linear regressn. coding, using inversion of matrices, v.s. linear regn. coding using many ll codes w.o. solving linear eqns — would seem to be a case in pt. — in which t. matrix inversion method has much less cc. ···· (Tho it may well be that using t. methods of 24.25 ff)

In t. case of Bernoulli codes (& probly 21.41): There may not be much cc difference betw. direct CBI coding & Bern. coding — i.e. cc's may be ≈. In general, if a conditional stoch branch. (57.12) is to be made, this can be done using a special srtn. — which was t. idea of 57.12 ··· Hvr., using CBI, we can list t. possl. branch alternatives & assign codes to them of difrnt lengths (Huffman codes). Hvr., if we do this for a single 3 way branch, say, we will have appreciable error in pc's. Only by coding many branches together, can t. average error in Huffman coding → 0. So in this sense, maybe even for t. simple Bern case, there is a serious advantage of not using CBI directly.

Hvr., in t. Bern case, using Lsrch, I suspect that t. errors in pc's will not be an impt. factor in ↑ L cost of t. total srch. T. total L cost will be a second order effect in t. error of pc — since t. L cost will be min. when t. pc's are exactly correct.

→ So I'm not yet sure that there exist methods other than CBI that have much better cc than CBI.

$\boxed{6.13.81}$ T.S.

(see 33.01–.40 for ≈)

.01 Well: Perhaps returning to t. Sheep: As befor Make tng. seq. like 20.20 ff:
However, Do it mainly in English at first, & try to make t. solns. pretty much what
t. human solns. seem to be. Try to list t. impt. concepts needed — even
.04 if I have trouble defining them exactly. E.g. Th. idea of {substitution
of an expressn for something "equal" to it} should have a much higer
pc than one mite think from t. way subsn. is defined. This is because this kind of
subsn. is one of t. properties of "equality". Just how much more of t.
properties of "equality" are intuitively known by humans & usd to help
.10 solve problems, is unclear to me at present.

(Human) For Many test: have excess of stories so that calibration be usd to calibrate another set of stories.

.11 Another example of a way in which pc's should be ↑ by
bringing in extra info: In the learning of t. operation "Eval"
of 20.20 ff., the use of recursion is made part of a random search
— i.e. recursion is one of t. possl. operations usd in t. search.
— As such since recursion has no greater pc than any other operator, it
has a rather low pc. However, if one understands the "point"
of evaluation of t. expressns., one mite note that one wants something
"equal" to t. original expressn., but that only has one number in it.
From this, it become a GPS problem, because one has a
simple measure of ht. & a simple set of xfmns to ↑ ht.
— Or one has a simple measure of "difference" betw.
.27 1 & >1 numbers in ⟨t. expressn. thus far⟩.

.28 After TM has workd several "eval" problems using GPS,
it should beable to shorten t. procedure, by noticing reggys. in th.
.30 process of finding t. solns. Possibly it may later even find greater
simplifn. (a ↓ of cc) by discovering that t. solns can be
easily expressd as a recursion. e.g. see 50.31 and 46.23
Note: .28–.30 is a start of "$TM_2$" applicn., but of a very
.35 simple kind. We have $TM_2$ watching $TM_1$ & trying to improve it.

Perhaps, after TM devrs any regularity a/o a simple
I → O set, it tries to figure out a way to implement it
at minimum cc. Sometimes this can be difft a/o complicated: It may

depend much on t. statistics of t. [ I ] set ~~for which~~ TM may not ~~know~~ how much of a ssz.

---

.03 : 57.23 SN A new, (approx) way to assign conditional pc's to conditional stoch branches. Code t. corpus, using $x_1, x_2, \ldots x_r$ for all of t. conditional pc's. Then assign values to $\vec{x}$ ∋ t. pc. of t. corpus is max. Trouble is, this assumes t. pc of each data. is 1. To deal w. this, One would then mult. by t. pc of each (or not conditional) pc. defined. I don't know how much in error this pc assignment method is — whether t. error is serious.

---

ail

.17 A Review Of what I've already done w.r.t. implementing t. TS ideas of 59.01 – .10

Ruff outlines

1) Learn Unary & binary functions & assoc. them w. their names.

2) Either Learn or have as a primitive, t. concept of substitution.

3) Meaning & implications of "equality"

4) Learn t. Eval funct. for all alg. expressns (probly in RPN notation) 281.28

TM learning simple Boolean logic (perhaps branches) so it can discover & use conditional/branches in pgms. see 35.21 ff 39.01 – .22

More detail & References:

.22 1) Unary & Binary Functs: some examples: 20.24, || Then 28.02; 35.01 – .20 38.05 – 39.18 ; 43.22 – 44.02 (v.g.)

.24 2) Substitution: How to learn: Early discn. of "a string of a certain type" — This is a genzn. of t concept used in subsn: in 80TS! 282.12 – .20 ; 282.12 – 284.02; 285.10 – .20 Then later: 39.23; 41.20 – .26 ; (44.03 – 52.07) ← this last has some pretty good ways to learn subsn: in particular: 51.30 – 52.07; 50.10 – .18 ← seem v.g.

equality ! 80TS 282.12 ff 40.11 – 41.19

.32 3) The concept of "equality" & of "quantity" (Re: "quantity" — This also involves t. idea of linear ordering ... I don't think I've written about this aspect of "quantity".) Early: 80TS 282.12 ff : Recent: 59.04 – .10. 40.11 – 41.19

RPN 40.27 – 43.21

Eval: 59.11 – .27

4) Eval: 281.28 ff first analysis: Use of recursion to learn it: 50.31 – .40 46.06 – .40 ; Perhaps learning it w.o. "explicit" Recursion (as perhaps humans do) 59.11 – .27 ← Impt., hype way. 59.24 – .35

.01 On possl. contins. of 20.40: In 1980 TS see pp 146, 148, 172, 181 } ("est" "hard") These pp. are clipped together. (206.27—.40 / 211.25 ff) also has refs to other relevant work. See Review (BM) of 207.01 for a overview of ●......

172 gives [redacted] a reasonable direction for present TS. [redacted] to move in

say from understanding Alg notation, to "simplifying" expressns, prove trig identitys, simplify trig expressns, Solve linear equs. Solve some N.L. equs.... (134.20 –.30)

.06

.07 After Learning t. function "Eval": some specific things to learn:

1) Solution of simples equs (Not necly. linear) by alg. manipulation.

2) ~~" " " by addition & subtraction~~

Perhaps soln. of quad. equs. in various ways.

3) soln of equs. by graph drawing a/o successive approxn.

4) soln. of simult. equs. — linear, N.linear, in various ways: subtraction, substitution, successive approxn.

SN 6.15.81 I think human solns of search problems are biasd toward hy pc solns. — i.e. for a gn. Lcost, t. human soln. will usually be of hy pc, & relatively hy cc. This involves usg rather "complex" (i.e. "appearing" in t. sense of having many steps; but each step of very hy pc), abss. of hy pc, but also hy cc. This way to solve problems would seem to be much better (for induction) than [redacted] low cc, low pc solns of about t. same Lcost (or even lower Lcost). Hvr., in any case, Lsrch obtains all solns. for whatever Lcost they have, in ↑ L cost order — so it really doesn't usually make much difference if on has a method that get hy pc solns. somewhat earlier: But at any rate, I think I'll have to think about this a bit more!

---

Well, o.k. say, TM knows how to do "Evals." of alg. expressns. We next have a labled problem (a labled)?: Given an alg. expressn containing "x", ~~[illegible]~~ equal to a number, to find a number that is "=" to x.

Re: these difnt. kinds of problems: We somehow "Tell" TM what t. problem is. Later, when TM has ~~[illegible]~~ matured some, we will present ~~[illegible]~~ demns of problems in some simple lang. Paird w. t. problem itself. From these pairs, TM should be able to learn t. relation betw. t. 2.

6/6/82 { 2 possl. ways to look at "solving X+3=8" ① Inversion of t. operator, +3 & apply it to 8 ② find a no. x ∋ x+3=8. | concept ① leads to algebraic manipulation: ② leads to successive approximation. They may be 2 difnt. concepts of "equality" — 2 difnt. properties of t. relation "=".

.01 Consider problems like: (Alg. notation) : $x+3=7$ : $(x-y)-z \neq x-(y-z)$

To get t. expressn. into t. form $x=n$, where $n$ is a number.

.03 Maybe learn that if $x+\phi=8$, say, $x=8$

Learn this property of $\phi$ ......

To be sure that TM "understands" t. trg. seq. at each point as well as a human: be sure that TM has acquired (or has been given) all of t. manipulative tricks that a human would have at that point for dealing w. t. things in t. trg. seq.

---

Then do more complex ops like $x+3=(7x-9)\div 3$.

---

.16 from .01; $x,3,+,=7$ One could reason (Human-like) by saying: "7 is 3 larger than $x$; so $x$ must be 3 less than 7, i.e. $x$ must be $7-3$ or 4". It may well be, hvr, that that reasoning has no more understanding of t. problem than would be represented by any other means of being able to solve $x+3=7$.

.21 also solve $3,x,+,=,7$. — which looks harder; first we may want

.22 to xfm it to $x,3,+,=7$ then solve it like .16.

To solve this one may also need "assoc." law of addition".

so $x,3,+,3,-,=7,3,-$. from 43.13 : $\{+3- = 3-+\}$

so → $x,3,3,-,+=4$

→ $x,\phi,+=4$ → $x=4$ (by .03).

Say t. assoc. laws of addition-subtraction (& Multiplication-division) are available.

Could TM solve $x,3,+=7$ or $3,x,+=7$ using GPS, say, w. acceptable cc & pc & cost? Well — depends on whether I can devise a suitable set of difrnces.

---

.37 A useful concept: If $x$ is subject to a seq. of xfmns involving no nos. other than constants & ends up = to a constant, one can almost always invert t. xfmns 1 by 1 & get t. value (or values) of $x$. — could TM devn. this concept?

A possl. way to discover 62.37 would by by recursion!

If $xF = \alpha$ is solvable, then $xFG = \beta$ is solvable — its t. idea of peeling of t. layers of complexity 1 by 1.

.64 Another useful concept would be a PLAN (GPS. method) [ had m 1980, in which TM attempts to xfm a new problem to an old problem type of known soln. T. method is improved by ↑ t. no. & types of probs solved, by increasing & improving t. xfmns used as well as t. obs that tell one what xfmn to use. (This business about xfmng t. present problem into a known sort of problems may, concievably use t. methods of GPS).

The xfmn of 62.21–.22 is like .045ff in t. sense of xfmng t. problem into a new prob. of known soln. method.

.19 6.16.81 Dcvry by recursion probably means a trg seq. consisting of 1, 2, 3, 4 ...
.20 layers of functions. A smarter mathematician would look for
.21 a structure that was "factorable" into many layers. We can either wire this heur in, or teach it by induction, or perhaps
.23 find t. Lcost of its discovery w. a trg. seq. that
.24 is not (ad hocishly) directed toward t. dcvry of this heur.

T. Lcost of dcvry via .23–.24 is an impt. thing that I'd like to know: i.e. At what level of cleverness would TM begin to dcvr. such heurs at reasonable Lcost?

.28 T. dcvry of this heur could be a good example of ↑ of pc of a defn. by "recoding t. past": E.g. say TM had learned lots of recursions for various problems using trg. seqs like .19. Then, later, w.o. such a trg. seq. it tentatively considers t. heur of .20–.21 in a particular problem & gives it much hyer pc because it "could have been used" to solve many probs of t. past w. much hyer pc than was obtained in t. past.

Re: .28: There may be some (not exactly "diffs") but additional considerations: Say, in t. past, TM had trg seqs like .19 for this probs. T. defining of t. heur .20–.21 would also enable t. dcvry of seqs in t. past to occur much faster — in t. sense of less SSZ (as well as greater pc, so less $\frac{cc}{pc}$ = Lcost used in t. solns.

6.16.81 TS



{This ↓ of $$Z would seem to be a difrnt kind of very impt value, that TM should try for), Ordinarily, it is assoc. w. simple ↑ of pc.... — perhaps it always is .... But should we not give it "extra points" for this particular desiratum?

In certain kinds of situations, TM will have to "pay" for each example that it gets — e.g. in physical experiments this is always t. case. We want TM to act in such a way to devr. t. "laws of Nature" w. a minimum cc for experiments, + cc of computation.

Note: cc of computation is never negligible, wrt cc of experiments. There is usually t. available possy. of using a lot of cc for compns on Induction to compensate for little &/o poor experimental data. Occasionally there are points in which certain experimental data is essentially absent — e.g. pictures of t. far side of the moon & much Astronomical info. Also Info on hy energy physics can usually be obtaind only by doing hy energy experiments (cosmic rays &/o accelerators).

Hvr, I suspect that [redacted] special & perhaps Genl. relativity & Quantum mechanics could have been worked out in much detail w. only t. data up to ~ 1900!

{ 5.18 to 5.54

{ 4.67– 4–71 copy → GET.D

5+64

To Nip 1) Bolt 2) Sheets

On TS in General: Actually, any problems would be of interest; Just Give t. exact problem dern., then t. complete soln. as findable by a human... including all needed heuristic & conceptual tricks needed, & t. search space.

If possl., find >1 soln. for each problem.

Some impt. Things: ① T. dern. of t. problem: This should be in some small set of standard forms, so TM can understand what needs be done.

② T. dern. of t. soln. will probably be in terms of some "plans". At first, there will be a small no. of such "plans" available to TM. — later he will elaborate them & devise new ones.

③ What is t. "search space" for t. solns?? (so pc's can be estimated).

Actually, t. problems need not be in T.S. order (a partial ordering) at first; This partial ordering can occur later in ~~~~ my analysis.

One v.g. idea is $\boxed{\text{APL}}$ T. functions devised for APL are very economical (≡ by PC). The "Inhumanness" of t. notation is of some interest, hvr. — It suggests that there mite be something seriously wrong w. it.

.07 Part of t. "inhumanness" is t. lack of grouping symbols — like parenths. — So its not easy to visually break up t. expressn. into modules. A possl. second "inhumanness" is that t. modules are not very familiar ones. We are not very familiar w. their properties, their modes of manipulability.

If .07 ff is all there is to t. inhumanness, then I guess its o.k. for T.M.

Re: "Inhumanness": As applied to RPN (& to some extent to normal 2 dim. alg. notation): In RPN t. associativitys of "+ & x" are not made part of t. notation, so specific equivalence / x forms must be learned (or gn) to deal w. this.

In ordinary alg notation, a human can easily scan a 1 dim formula & look for familiar modules, in attempts to parse t. formula in a useful way. I think t. problem of "How to parse an expressn. in a useful way" is one of t. key problems that a notation must help with. And deficiencies in t. notation must be dealt w. by giving or teaching TM appropriate techniques.

One Use of this idea: That one can sometimes use a change of notation to make it easier to scan for certain module types.

.27 A $\boxed{\text{common Math'cal prob.:}}$ How can I parse this expressn. so it is in a "certain form"? e.g. — — so it contains a sub expressn. that is a "certain form." Many problems in symbolic integration & t. symbolic Soln. of diff. equs., ~~~~ (or / simply soln of linear or ~~~~ n.l. equs.) can be put in this form. (e.g. in diff. equs to find an "integrating factor").

D.G. Willis has written some reports that I have on t. "costs" of decomposition of functions. This is a kind of "Parsing" of funtions. He uses a particular "cost" funct. .... perhaps because its math'ically tractable..... I'm not sure just how relevant this is to TM.

Actually, much (if not most) of Algebraic Manipulations are not ordinarily best done t. ~~they~~ way humans do them, but t. way "Macsima" & other similar pgms do them. Macsima uses LISP, hvr., & I'm not sure that t. nature of t. optimum algm mite not be diffrnt if a more efficient (cc-wise) lang. were used.

However, at t. present time, I'm not trying to get TM to solve these problems in optimum ways, but rather, I want to devise a reasonable tng. seq. to ① study TM's solving of it ② Teach me how to write Tng seqs in general. So this stuff about ≈ Macsyma isn't relevant now — Tho eventually, I'll want TM to be able to learn to do various methods/things t. way Macsyma does them.

.10 Getting back to t. Tng. seq: T. problems of 61.07 would probably be an adequate beginning frame work. I suspect they cover much more than I need, to get a good picture of how TM works.

A possible initial outline:

Eval function → evaln. of unary & binary functs.
→ substitution → "correlational studies"
→ possibly recursion (perhaps primitive).

Soln of $f_1(x)$ = known no.

.18 Soln of equs of t. form $f_1(f_2(f_3(\cdots f_n(x))\ldots)$ = known number.

↓

T. concept of "inverse of a function" of unary functs & of binary functs: e.g. $x$, 3, + has inverse $x$, 3, −.

Ability to solve $f(x)$ = known no.

Soln. of equs that are "close" to .18 ; "close" = small cjs. = small Lcost for search.

" " " " " "more distant" from .18 .

.26 → 67.26

$x e^{x^2}$

$x^3 e^{x^2}$

$d(x^2 e^{x^2})$

$= 2x e^{x^2} + 2x^3 e^{x^2}$

$\int 2x^3 e^{x^2} = x^2 e^{x^2} - \int 2x e^{x^2}$

$= (x^2 - 1) e^{x^2}$

$d = (2x) + 2x(x^2 - 1) e^{x^2}$

$= 2x^3 e^{x^2}$

.27 6.18.81 NP : SN, sort of : For TNG seq. study, one way would be to devise a large set of problems, then partially order them w.r.t. one helping solve t. other. Since each prob. can have several difnt. soln. methods, t. ordering is not just a "simple" partial ordering. Hvr., there does exist at least one partial ordering, ∋ if t. probs were presented in that order, t. total Lcost of solving all of them would be minimum. (Here t. Lcost of soln of each prob. is t. Lcost of t. first soln. to be found in t. search — so only 1 (t. "best") soln. to each prob. is used).

Then write out t. more detail solns. of some or all of t. problems, & see what additional problems must be inserted into t. set to allow reasonable Lcosts for each soln.

Actually, t. initial problem set need not be part of a single unified tng. seq. As I work out various of t. probs., I will find how various problems fit (or don't fit) into t. tng. seq. set.

A kind of Heuristic: Say I see a person solve a particular quadratic equ. by "completing t. square". I can easily genz. this to solve **any** quad. equ. w. real or complex coifs or even literal coifs. W. some cleverness

.04 I can generalize further. I don't see how it can be used to solve cubics.

Say we want to solve $f(x) = 7$. $f(x)$ doesn't have a simple low cc inverse, but we may be able to find $G(x) \ni G(f(x))$ has an easy inverse. e.g. say $G(x) = x+2$, so $f(x)+2$ has a simple inverse — i.e. $h(x)$: $(h(f(x)+2) = x)$.

so $f(x)+2 = 7+2 = y$ ; $x = h(y)$.

Anyway after a problem has been solvd, TM should try to genz. it as much as possl. T. directions of genzn. are **controld by**: Ability to use t. genzn. on (1) problems of t. known past (2) **problems expected** in t. future, obtaind by extrapolating probs of t. **past** (3) Perhaps problems of t. future, given to T.M.

.21 This "Genzn." hour is impt. in an early (perhaps V.G.) (1980) "plan" of mine, (verbatim 80TS 68.10–.40) in which one tried to **xfm a problem into** t. <set of problems of known soln method>. This genzn. method is a way to ↑ t. size of this set in a useful way → see 68.03–.05

.24

.26 :66.26: Soln of $f_1(x) = k$ (= known no.). Where **TM** could learn to solve it for each unary funct, $f_1$.

Say $f_1$ is a known unary funct. $-$, $f^0$(identity), $\frac{1}{x}$, $x^2$(?), $\frac{x}{2}$, $2x$, $x+1$, $x-1$. (plus maybe Boolean functs)

If we have $f_2(x, a) = k$
$f_2(a, x) = k$
then TM can learn to solve these for $f_2 = +, -, \times, \div$ (& perhaps Boolean functs) & (almost) all values of $a$ & $k$.

(of course $x \times 0 = 0$ is unsolvable, but we would'nt give TM that problem!) This ↑ sort of "unrealistic" policy is o.k. only for a "study problem" TM — for a real operational TM! I think it would be dangerous to fudge tm. Tng. seq. thusly!

Hvr., w. a mildly educated TM., it mite be possl. for it to devr., that division by zero is a special case.

Another diffty like this (w. a much more diff't soln.) is t. problem of real computers having finite accuracy. To get around this, we can use examples that have integer (or rationals) solns. only .... but this does cut out much of algebra — unless we table probs & solns. w. t. amt of accuracy desird. — which makes t. whole thing much more complicated.

It may be possl. for me to utilize both kinds of AI results (also results in "pattern dcvry" which is not modeled after conscious mind) & perhaps integrate all of them into a large system — a Tng. seq. using CBI.

One possible study to start w. would be Winston's Thesis — t. part that I did via CMI — in an alternate soln. What I would do, would be to express W.'s soln. directly as a CBI sol. I might compare it to my alternate soln., but this is not necy. T. reason to chose Winston's work, is that I have easy access to understanding it via my Tbilisi paper.

One reason not to start on t. AI probs for PTS's: I'd like to get some practice in devising tng. seqns from (Apparent) human solns. of problems. Also find out how to treat dcvry of heuristics, etc. At t. present, it seems like (t. Alg. tng. seq of 61.07 ff & up to t. ideas of up to 68.23.) would be a simpler way to start. Possibly by using (at least) 2 plans: (1) GPS (2) T. plan referred to on 67.21-.24

Th. Winston prob. is learning simple Boolian concepts from examples. He did use a very special kind of Tng. seq. utilizing "near misses". Also he had to have negative as well as positive examples.

—My vague remembrance of W's method: each example was of t. form $\alpha \cdot \beta \cdot \gamma \cdot \delta$. ( $\cdot$ = Boolian "And" &, + = Boolian "or" ).
where $\alpha, \beta, \gamma, \delta$ are simple properties

The first hypoth, is $\equiv$ t. first example / say $\alpha \cdot \beta \cdot \gamma \cdot \beta$.
Any (neg / positive) ~~positive~~ example that (doesn't fit / fits) t. hypoth leaves it invariant.
" (positive / neg) " " (doesn't fit / fits) " " modifys it minimally.
e.g. t. subsequent positive example $\alpha \cdot \rho \cdot \gamma \cdot \beta$ would yield t. new hypoth.

$\alpha \cdot (\beta + \rho) \cdot \gamma \cdot \beta$. In general, a hypoth is of t.

130 form $A \cdot B \cdot C \cdot D$ ; where $A, B, C, D$ are sets of simple properties
( I think Boolian sums of simple properties ).

for A neg. example ~~[illegible]~~ that fits hypoth we again try to modify t. ~~[illegible]~~ hypoth minimally so it wouldn't fit but
I don't know just how this is done. We want t. new hypoth to satisfy all old data (pos. & neg.) as well. say $A \cdot B \cdot C \cdot D$ is t. present hypoth.

6.21.81 Then example $\alpha, \beta, \gamma, \delta$ occurs w. $\alpha \subset A$; $\beta \subset B$; $\gamma \supset C$; $\delta \supset D$.
we could try modifying t. hypoth by ~~[illegible]~~; ~~if it does agree w. corpus,~~ ~~[illegible] B → B~~ subtracting $\alpha$ from A. If this gives a model that

is inconsi w. t. corpus ; try B → B minus β ,

then C → C minus γ ect.

C minus γ = C · ∼γ in this case.

C minus γ = C · ∼γ if dec as is the case.

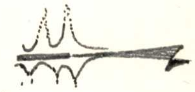
Hvr., if C were constructed by adding in properties, 1 by 1 that examples had, then C → C · ∼γ would remove γ from C & certainly cause t. model to be inconsi w. t. example that caused γ to be included in C.

So, one way to deal w. this is to consider minimally complex joint properties : i.e. properties that are t. "and" of 2 other properties or their negation(s) like α · γ or ∼α · γ or ∼α · ∼γ or α · ∼γ.

Hvr., best look at t. ? bibli paper for how Winston dealt w. this.

See 80TS 169 for disn. of Winston

It would seem that Winston's problem would be too simple to be of great interest as a toy. esp. t. main pt. of interest is that it is a commonly occuring problem — i.e. t. Boolean model of 69.30 is commonly usd. by humans.



WRT A.I. work, the work of Simon Langley et. al on Bacon uses fairly complex, interesting hypoths to induce physical laws. Hvr., these laws & their heurs. may be very far from t. primatives. I may want to look at their heurs & see if they have any things that I will want TM to learn eventually — or whether these heurs would be easily learned by TM as a reasonable continu. of my work on t. algebraic T.S.

2.7182818
28

Make A kind of chart to list t. kinds of probs I want T.M. to solve in ≈ a feasible order. These should be first written as for a human student, then work solns in more & more detail. Note: t. "soln." gives t. routine by which t. student (or TM) was able to solve t. prob. in acceptable cost. ... it includes any necessary heuristics.

6.21.81 TS 

Impt. general Methodological Note

Some past work on such a chart! → 59.01 – .10 : {60.17 – 61.06} ← Bibliogy & review. thru "Eval"

① T. probs of 61.07 (also 61.01 → .06 for some continung. & roots to continuns.)

② 66.10 – .26

③ 67.26 – 68.05

~~Evaln. of unar~~ So a prelimy chart.

.06 1) Evaln of unary functs : $neg(3) = ?$ ; $sq(3) = ?$ $\frac{1}{x}(3) = ?$

2) " " binary functs. $3+5=?$ $4 \div 2 = ?$

2.5) Correlational analysis (to help devr. (3)) — also has other uses.

3) Substitution of expressn$_1$ for express$_2$ in expressn$_3$.

$3, 2, a^3 + 4 \cdot a \rightarrow (3) \quad 3^3 + 4 \cdot 3$.

4) recursion. including "stop rule".

5) Eval function (evaln. of any alg. express. (say in RPN)

.14 6) Soln. of $f_1(x) = 3$ w. $f_1$ = unary funct.

GBAG
GBEG
GAVG
(12)

.20 7) " " $f^2(x, 5) = 8$ w. $f^2$ = binary funct
and $f^2(5, x) = 8$

8) Concept of inverse of a funct. first ; unary functs : like $-$ is inv. of $-$, $\frac{1}{x}$ is inv. of $\frac{1}{x}$ ; $\sqrt{x}$ ($x^2$) is inv. of $x^2$ ($\sqrt{x}$). If $f_1$ is inv. of $f_2$, then $f_2$ is inv. of $f_1$ (— well usually : its true if $f_1$ & $f_2$ are both singlevalued.
i.e. if $f_1$ is increasing, it has an ↑ inverse, & both are singlevalued.

Next; binary functs. like $x+1 \rightarrow x-1$ ; $\underbrace{x^2+1 \rightarrow \sqrt{x-1}}$ ← do I want this?

$\sin^{-1}$ is known, $\cos^2 + \sin^2 = 1$ so: how to find $\cos^{-1}$?

9) Soln. of linear equs. : simpler ones first, then more complex ones in which it is more difft. to get equ. into standard linear form.

33 10) Soln. of some nonlinear equs. $\sin(x) = 3$ $\sin(x) + 3 + 2\sin(x) = 5$.

11) soln. of quadratic eqs. $(\sin(x)+1)^2 = 3$,

12) soln. of 2 simult. linear equs. by a) substitn.
b) "subtraction"

13) genzn. of 12) to some simult. n. l. equs.

14) Genzn. of 12) to n simult linear equs in n unks.

I am thinking of using 2 impt. heurs. ① GPS : This, as I use t. term, is not very clearly defined, but t. spirit of it is: I have $\alpha$ & I have to xfm it into $\beta$. $\alpha$ & $\beta$ differ in t. folg. ways ....., each of these differences can be dealt w. using certain xfmns. I use these xfmns to "reduce" t. differences.

More generally, ~~it~~ if we have $\alpha$ & are required to xfm it to $\beta$ using a seq ~~[illegible]~~ of t. folg. xfmns ......., we "compare" $\alpha$ to $\beta$. This comparison tells how $\alpha$ differs from $\beta$. The object $(\alpha, \beta)$ then tells us (thru experience {& in t. case of New. & Sim, by logical analysis}) how to move $\alpha$ toward $\beta$ using t. xfmns a/o ~~[illegible]~~ substrings of them. How to do this in a general way is an interesting problem.

---

.20 ② Th. "xfm t. problem into a problem of known soln" heuristic. This plan was worked on much in 1980 : its referd to on 67.21–.24 ~~I dont yet have t. refs.~~ → {It is 80TS 68.10–.40}. Its a ~~common~~ plan commonly usd by me. One Big problem is to index large sets of problem solns. so its easy to tell if a problem is in one of them. ~~One impt. part of t. heur:~~

Def ~~Def~~ This heur of 73.20H will be calld ~~$73.20H~~ 73.20H — or H7320. T. GPS heuristic will be calld ~~$~~GPSH / or HGPS

.27 Whenever a prob. is solvd, ① genz. t. soln. in various ways that make it likely that new probs ~~[illegible]~~ (or old : recoding t. past) will use these forms. Also figure out quick tests to tell if a new prob is within t. solved classes.

---

It would seem that t. T.S. ending in 72.37 (soln. of linear eqns) would be a not-bad T.S. to study. That there is probly enuf in it to illustrate various impt. problems in TM. If it doesn't, then I can put in complex eqns. that have to be put into linear form.

Then $3\sin x + 5 = 0$ etc. — i.e. eqns that can be put into ~~[illegible]~~ ~~a~~ a solvable form say using 73.20H ~~[illegible]~~ or eqns that become linear after suitable substitn — again, possibly via 73.20H.

~~Be~~ Also t. continn. 72.33 ff. if necessary.

---

So try putting "up to 72.37" in more detail.

---

③ Impt. Heur I often use is ANALOGY : usd often to genz.

→ 75.01

6·24·81 TS

.01: 73.40: Well lets go back to 71.06 :

.02 1) unary, Binary functs : see 60.22 for [redacted] bibliog. review: for this problem.

6·25·81 adequate bibliog: 38.05 → 39.18 ; 43.22 — 44.02 : ← Both of these ideas are good: they deal w. difrnt ways to recognize t. needed operator by its "name".

This can be separated out from t. problem of finding what t. function is, from 1 (or in t. case of |x| & sign(x), 2) examples.

T. only functional identities that seem a bit difrnt., are |x| & sign(x):

Take sign(x): This needs 2 (rather than 1) data pts. — also, it involves a decision — It looks like an essentially More complex kind of function than neg(x)

6·26·81 Maybe not: consider neg(x): This can be (−1)·(x) or it may involve examination of t. sign bit: [redacted] (if any!) of x.

→ Or we may have [redacted] sign(x) be a primitive.

What I was thinking of was having ">" as a [redacted] primative: input is 2 nos., output is Boolean yes, no. This means we need to use binary functs to learn unary functs — I was thinking of [redacted] first learning unary functs that didn't need [redacted] primative binary functs.

Actually, there are 3 comparisons of interest: "≡", [redacted] which compares any two [redacted] symbols or seqcs of symbols. "=" which compares a pair of numbers, & "<" " " " " " " .

"=" can be used to compare 3 to (4−1), or just 3 to 3 or 7+1 to 9−1 or 9 to 5+5

.22 However, a human in observing [redacted] nos. would usually notice quickly whether it was [illegible] positive or negative — This is a hyp ob. → (see 78.25)

so, inducing [redacted] sign(x) from 2 examples should be not diff't, using this hyp ob.

The moral here, is "stick to English" as long as possl.

Here we have t. 2 examples $+n_1 \to$ [redacted] $^{+1}$ ; $-n_2 \to$ [redacted] $^{-1}$.

We hypothesize t. 2 models x → [redacted] & x → −[redacted] ; to decide betw t. 2 we "look at" x. t. sign of x ← (see 78.25) is a hyp ob., so we try it early, & it turns out to be adequate to decide betw using x → 1 or x → −1.

Hv., φ usually doesn't have a sign. If we make it "+" by convention, sign(φ) = +1.

.35 Similarly |x| could be induced from 2 examples.

It does in common computer representations of integers.

Similarly, all t. ~~many~~ other unary functs. [illegible] & binary functs can be easily induced by trials in English.

SN if t. examples use "random nos" (nos. whose bevs are beyond t. CB of t. machine being used) then t. ~~[illegible]~~ "solns" of t. induction problems I'm using are not very distant from t. correct solns. Actually, I don't need random nos. Any no. that is not total recursive has an infinitely long desc. & is acceptable.

Some constants: 0, 1

Some unary functs: x+1, x−1, |x|, −x, $\frac{1}{x}$, sign(x), maybe 2x & $\frac{1}{2}$x

sin cos

$\sin^{-1}$ $\cos^{-1}$

$x^2$ ; $\sqrt{x}$

self inverses: $\frac{c}{x}$ ; c−x ; $\sqrt{c^2-x^2}$

50 × 2000

2.5 A

6 pp/hr: 100 hrs/vol. 2 k hrs./2 vol.

→ 78.01

ON Problistic Algms

SN AHHA! We cannot ordinarily tell whether a no. is random or not, but we can construct a no. ~~that is random~~ that has a probty of > $1-\epsilon$ of being random, for practically arby $\epsilon$.
This seems related to Rabin's work on problistic algms, in which he constructs a no. that is prime w. probty > $1-\epsilon$ for practically arby $\epsilon$.

Perhaps construct a chess move that has no winning reply w. probty > $1-\epsilon$!
Also: To prove a thrm. in Geometry: Use random values of params to make an example, compute positions
.07 of all resultant pts. If theorem is true within $\epsilon$ for that example, it has a probty of being true $\sim 1-k\epsilon$ where $k$ is some constant.

→ 100.01

.08 SN Suppose we have mc, M, & corpus C, & we know a pri, that ∃ a code, x for C on M, ∋ M(x) = C &
The maximum time between M's printouts of bits of C, is T.
(This is a kind of CB on M). Can we now devise a good search strategy to find x given M & C?. If C has n bits, the max compn. time for C is $\ge n \cdot T$.
Say we used Lsrch.

If x is of length $|x| = m$, then the Lcost of finding C is $< 2^m \cdot n \cdot T$.
— which can be very large.

However, consider trickyer methods of $\approx$ Lsrch: Say one just tries codes at random; One start with 0, & feeds in random nos. when requested by M.
As soon as M puts out an incorrect bit of C, or takes too long betw. bits, we try the opposite of the last input bit (or if this has been tried unsuccessfully already, we go back to the bit before that & change it & try, etc.)
At each input bit, we record how much time has elapsed since the last printout — so we know when to backtrack on new trial branches. We also must retain the total used memory of the machine as of each branch point — or simply a list of memory changes since the last "memory dump" or the last list of changes.

Another possy is to use a reversabler computer ("for backtracking") (like C. Bennet)
There were some complaints that such a machine was "slow" — but I'm not sure its true .... also it not speed but cc that's the relevant criterion.

If random choices are initialy made at each branch, maybe it can be shown that it is unlikely that one will get into a long branch (taking lots of time) that is actually wrong. — Or that this will occasionally happened, but not very often, & so the total amount of cc involved is small.

Well, consider a corpus obtained by linear regressn w. k coeffs & a certain
.37 var.

→ 77.01

N.B. — This is a "long corpus" of essentially low pc; it differs markedly from the hy pc corpi. Usually the code/bits a "long corpus" is $\propto$ the corpus length.

.01 : 76·37 : we may start out our code w. ~~—~~ t. equivalent of a certain set of coifs. We code along ... Then backtrack if the code doesn't work well. This backtracking changes a coiff., say, Then one trys coding forward again. The better t. predictions are, t. less frequently are backtracks (presumably). — There is, of course, t. Q of which coifs to change — & which bits of those coifs to change.

I'm thinking of "M" as a special machine designed for linear regn. coding. The code consists of t. set of coifs, followed by t. correction bits needed for each predicted data pt.

~~Initially, Random~~

One puts t. initial trial coifs in! — also t. Varc. This doesn't take long. Then various trial predns. are made & various trial corrections are made, untill t. data pt. of t. corpus is reproduced. T. ~~farther off t~~ larger t. initial predn. error, t. more random correction trials are needed to reproduce t. data pt. (These are random corrections to a gaussian distribn of given varc. — & t. longer it takes.

Actually, it mite be better to do it w.o. a gaussn. distribn: one has one fewer initial params to adjust, so t. search space for t. correct set of coifs is 1 dimension smaller — I'm not sure that t. result would be t. same, ~~—~~ as w. t. ~~—~~ Gausn distribn, hvr.

.01: 75.40: Simple unary functs:

$-x$, $x$ ; [ sign(x), |x| ] ← perhaps: ; perhaps $\frac{1}{x}$ .

.03 simple binary functs:

$x+y$, $x-y$, $x \cdot y$, $x \div y$.

| | | |
|---|---|---|
| Numerical | $x = y$ ; $x > y$ | } [redacted] Binary logical functs. Range: Yes, No. |
| General strings of symbols | $A \equiv B$ | |
| | $A \equiv B$ | |

10 min

.06

[redacted] baud. 1.6k 600 sec.

Mbit = 120 kbyte

Perhaps more complex binary functs:

$x^2$, $\sqrt{x}$ ; sin, $\sin^{-1}$, cos, $\cos^{-1}$, tan, $\tan^{-1}$.

~~Due~~ Tho there is redundance, I may want to make

$-x$, $x$, sign(x), |x|, $\frac{1}{x}$, $x^2$, $\sqrt{x}$ unary primitive functs.

& $x+y$, $x-y$, $x \cdot y$, $x \div y$ be primitive binary numerical functs.

$x = y$ , $x > y$ } Binary numerical to Boolean, ~~A~~ Y/N

$A \equiv B$ } Binary arby symbol or string to Boolean Y/N .

Numerical constants: $\phi$, 1,

Boolean consts: No, Yes ( ≡ N/Y ) .

Perhaps standard Boolean functs: Unary: identity, negation.

Binary; [redacted] $\cup$, $\cap$, [redacted] exor

$\boxed{6 \cdot 27 \cdot 81}$ Hyp. c. Obs: — Range: No, Yes. Domain: Numbers. { Nos. → N/Y }

.25 1) is $x > \phi$ or $< \phi$ i.e. what is sign of x? or is $x \geq 0$? → N/Y .

2) is $x = \phi$? N/Y

3) is $x = 1$? N/Y .

4) is $0 < x <$ [redacted] (i.e. is x betw $\phi$ & 1?) [redacted] N/Y

5) T. Boolean functs of (.03 – .06) R: a) $x = y$? b) $x > y$? } range = N/y

c) $A \equiv B$? (A & B are symbols or strings of symbols).

The [redacted] obs [redacted] 1) thru 4) can be formed ~~by~~ by combining $x = y$? & $x > y$? with various unary & binary boolean → boolean functs.

6.27.81 TS



Some little probs (w PTS's) that I've ~~sorta~~ partly solved:

1) learning ~~bin~~ unary & binary operations w.o. being giv. t. name of t. op. — when t. op is one in t. compl. ~~mach~~ inst. set.

2) " " " " " " " " " " " " "

3) learning binary & binary ops, ~~[illegible]~~ with or w.o. being ~~[illegible]~~ gn. name of t op., when ~~[illegible]~~ t. op is formed by combining > 1 ~~[illegible]~~ machine inst.

.07 4) [redacted] learning sign(x) (or |x|) when they are not in machine vocably, but t. ob sign(x) exists. (This needs a bit of work). 1, neg(x), are available to TM [redacted]. See 75.22-.35

5) Learning meaning of substitution.

6) Learning "Eval" function based on 5): [I'm not sure I've done this as well as a human — i.e. [redacted] a human would [redacted] give hy [redacted] pc to [redacted] substitn. because "equal things ~~are~~ are "equiv." in many ways". [redacted] — a substitn. is something one/sometimes expects of a equiv. relation.]

---

Re: 4) see 75.22-.35.

Re 6) (A) Another reasonable, hy pc way to learn "Eval": Use GPS heuristic! i.e. Making 1 subsn. ↓ t. no. of ~~[illegible]~~ symbols in an expressn. to be evaluated. Using this single "difference", repeated subsns. continues to bring us closer to t. goal.

.26 After we have worked several of these problems, we look at t. solns. & note that applying "~~[illegible]~~ subsn of any legal sort" to t. expressn "as many times as is poss!" (This latter is t. "stop" ~~[illegible]~~ rule) works, & is a ~~[illegible]~~ single hy pc representation of all of t. solns. As w. all GPS solns, t. Q is — where do we get t. set of differences & assoc. xfmns.? Well, ~~in~~ t. present case, t. "no. of symbols in t. expressn" is, I guess, a hy pc Ob. — so it could be devsd to be a good "difrnce" for this & many other problems.

.40 (B) Another way would be to use t. 73.20ff, of xfmng t. problem into a solvd. problem. This would use a trg. seq. consisting of 1, then 2, then 3 subsns ... etc. — Until TM, using a hyer level analysis ~~[illegible]~~ like .26, would get t. general soln. in (lower ec / faster) a hyer pc. form — i.e. lower L cost form.

6.27.81 TS TS T ST S 80

One Q [redacted] is: How to get from / associating each operation w. its name & going from (learning to)
this to t. idea of substituting in an expressn!

T. first part is like 4, 3, + → 7 ; then consider 4, 3, +, 8 * ("times" ≡ "mult".)

4, 3, + ; 7 is a reasonable thing to be generated by 4, 3, + .

consider t. operator on strings, **4, 3, + ; 7** ; subs.

[redacted] TM may know 4, 3, + ; 7 = gives **T.** (True). (F=False).

t. 2 args of "=" are t. string, 4, 3, + & t. no., 7.

We may be able to "teach" **TM**, that if 2 things are =, then one can subs one for t. other in an expression & t. new expressn. will be = to t. old.

Otherwise, why would TM consider [redacted] with any reasonable pc, t. idea of substituting 7 for 4, 3, + in 4, 3, +, 8 * ?

Sp. rel
Gen rel.
Photo El. effect
Brown. Motion.

.19 [6.28.81] (sun) {From a very old note} It would be good if I could break down TM & various [redacted] sub-probs into [well-defined sub-problems,] so I could work on them w. minimal warm-up time (≡ putting impt. stuff into / rapid-access-memory). (my)

---

.21 **General Plan** : Right now, what I need, is to put these problem solns. into "English" & then progressively refine them. I had pictured that at a certain pt. in TM work, writing tsq seqs for TM would be about t. same as writing them for a human — in t. sense that anything I unconsciously assume is known by t. human, would be known by **T.M.**

17×23
16×24
391
384

Actually, I'm not far from that pt. in writing these elementary tsq seqs, because t. human ("child") would not know much at this pt.

4 20P: out

.30 Also t. present [redacted] technique is about t. same as one would use [redacted] for more advanced problems: ∃ Dscr. of problem soln. (≡ method of soln.) / in English, then progressively more detail of t. soln., then attempts to dscr. all of t. heuristic concepts used in t. discovery. Also t. same expansion from English for other possl. solns methods for t. same problem.

3×107
900
3600 ×100
4
×360
3.30 00000

165×33

5280
5495
528÷16
528 #561

62881TS 

Discn for 80.21: T. early problems in learning unary & binary functs. seem perhaps too easy. Ideally, we would tell T.M. what these various symbols meant. Hwr., this is not possl. to do in this particular model., so we "tell" by examples or (carefully chosen). Learning Sign(x) (like 79.07) is a bit more interesting.

A major subgoal seems to be t. function "Eval" (string → no.). I do want TM to "learn" this as much in a "human" way as possl. I think to do this, t. concept of "equality" is useful, also t. idea of "subsn."

In general, we have "T. idea" of something, if we have (& know how to use) an adequate "property list" of that "something". In t. case of "=", some impt. properties: ① It is a relation applied to numbers.

② Things that are "=" can be subst for 1 another in most expressions.

③ It is an "equiv." relation ($\forall x$ $\underbrace{x=x}_{\text{idempotency}}$, $\underbrace{a=b \supset b=a}_{\text{sym}}$; $\underbrace{a=b,\ b=c \supset a=c}_{\text{transitive}}$)

3×151

453
455
91 5×
2×
13

"5's;
$\frac{1}{5}$" sp.

$\frac{15}{6}$ =
$\frac{x}{}$ 16.
$\frac{1}{10}$" sp = too

.15 Genl. discn: Consider t. 3 probs: learning "Eval"; solving linear eqs.; solving simult. linear eqs. In each case, I could write out 1 or more solns. T. nature of t. soln. will depend on just what TM knows up to that pt.

In t. case of solving simult eqs: There are 3 ways to do this that I can immediately think of ① subsn. ② "subtraction" (2 ~ methods) ③ Graphical & equivt. successive approxn. methods. When I was about to learn these, I did not discover any of these methods so they must be of some difty for a person having t. info that I did at that pt. Trouble is, I'd be hard put to characterz. just what that info state was!

Anyway for simult eq. solving, solns w. certain kinds of available info, will involve very large L cost. I want to see just how little info one can start out with, using an L cost of, say $10^{10} \times 10^{7}$ (1 yr of seconds; ~ no. of bits/sec for human compn.) = $\boxed{10^{17} \text{ bits.}}$ (≈ 1 man year!) = $2^{56.47}$ bits to find a soln. to simult. eqs.

$2^{56.47}$

Note at 5¢ for 10 ns bit. or 1¢ for 1 yr. or $10^{7} \times 10^{6}$ bits for 1¢. or $10^{-4}$ man years for ¢ or $100 for a man year!

If I can find such solns. w. reasonably small amt. of apri info gn. to TM., this will suggest that such a TM is capable of ≈ human (& superhuman) performance.

.32 SN Essentially 2 difent kinds of problem solns: ① Concious mind: hi pc, hi cc.

② Unconcious mind: (say in chess): low pc, low cc. ‖ Usual A.I. research ("Heuristic Prog") deals w. ①: hi pc hi cc.
In actual use by humans, ① & ② can use one another as subroutines. → 89.22

≈ complete

.34

6.28.81

Anyway, I should write out solns like 80.30 to various problems;

.36 like unary & binary functs, |x|, sign(x), desry of subsn., Eval, linear eqs & simult. linear eqs. Each of these solns. can be regarded as a difent PTS. — each can involve a difent backgnd. of knowledge by TM — They need not fit together. They are simply "study problems" for me.

As such, they are also "study probs" for writing pts — which in turn, are study probs. for t. final **TS** for TM.

So each of t. probs of 81.36 can be treated individually. & each can be solvd w. a variety of apri. info states. — This is in t. spirit of 80.19, & a v.g. approach to this work! ("well defined problem(s)")

I already have some solns. of this sort. Many within t. last month or so — but also, I have a lot of older solns. to t. Alg. notation]

→ problem — look at them: see to what extent they are relevant.

SN Re: "Eval" & "Subsn": Up to t. pt. that Eval is givn as a problem, Subsn. has (presumably) been usd. only to substitute expressns from an "=" expressn — even more exactly — to sub. t. "value" of an expressn. for it. — w. this backgnd., t. subsns. needed in "Eval" can be gn. by pc. Actually, after subsn. is learnd, by "recoding t. past", TM can regard evaln. of unary & binary functs as a form of subsn!

General Note on "~~diff~~ easy" v.s. "Hard" T.S's.: An "easy" T.S. is one w. small cjs's: say TM$_A$ & TM$_B$ have learnd a certain set of thrms, defs, etc. TM$_A$ has done this via a TS w. small cjs's, TM$_B$ via a TS w. large cjs's. They both know t. ~~explicit~~ needed concepts equally well, but presumably TM$_B$ has discovered various heurs to work diflt probs. & TM$_A$ has not.

— So for t. future corpus, TM$_B$ will be better at working more diflt ("creative") problems than TM$_A$.

It would be great if I could get t. TS's advanced to such a pt. that t. forgo. could be empirically shown!! → 88.01

"Eval" again: Another possible situation miter be that TM already "understands" t. notation usd as being a sequence of instructions operating on numbers & on previous results — in which case, induction of t. meaning of "Eval" would be very small cjs.

If we in some sense "tell" TM what "Eval" means, what is advantage of having TM learn Eval?: perhaps I can use this as example of t. impt. of learning, v.s. being told. → 90.07

A more general "Eval" funct, would also have a T,F, (Boolian) range

e.g. Evl 3,4,+ → 7; Evl 3,4,= → F, Evl 3,3,= → T.

A way to do a sort of "Branch" using this concept: say R(T) = 1, R(F) = ∅ (function: Boolian → ∅,1.)

Say we want; If x = T then y = 7; If x = F, y = 3

so: y = R(x)·7 + R(~x)·3, or X,R,7,*,~X,R,3*,+ in RPN.

This will do branches, but not loops. Note that this is rather wasteful of cc if it is performed as written — R(x) should only be evaluated once.

One way to get loops is by recursive defns.; $F(x) = (x-1)*f(x-1)$ if $x>1$
$= 1$ if $x=1$

$$\approx F(x) = 1*R(x,1,=) + (1-R(x,1,=))*(x-1)F(x-1)$$

this is easy to evaluate, but not by t. usual "Eval" funct. & Something like t. "$\lambda$" notation may be needed. Or, use a Recursive form of Forth (not nearly Figforth implementation, hvr.) or LISP.

.09 I think t. way I expected to write tng seqs was to write one ruffly in English, then as I refined t. various solns., I would find various concepts that were needed (in t. T.S.) (or that would have to be found by TM at excessive cost) & I would add to t. T.S. — Looks like idea for V.G. System

Also, for t. "final T.S." My main work would be ~~identifying~~ finding a reasonable seq. of problems, then finding 1 or more "solns", to each (including methods of soln. & any impt. heur tricks). Then t. rest of t. work was to be fairly mechanical for me... That I would have a system for obtaining t. tng-seq. & perhaps presenting more

.17 problems for me to solve, to be included in t. T.S. I would be a srtn. to solve various of t. problems.

For some problems, I will not know t. soln. (methodologically, that is) — so I'll just have to put it to T.M. directly or via a very easy tng. seq....

.20 which is t. way most students learn diflt. concepts (?) → Seee 91.20 comment

SN Q: I'd like "Eval" to be able to work w. many valued functions. In this case, a stack would probly be necy. to deal w. a complex expressn of multiple valued functs. In t. case of recursively defined multiple valued functs, perhaps 2 stacks would be needed.

So, t. way this would be done, would be to work backwards. Start w. a diflt problem — like Simult. soln. of linear eqns. Write out several solns. in English. Pick one of t. solns. For each of t. concepts involved: if it is not (what I chose to be a "given" or a "primitive"), consider that concept as a "problem", & I solve each of those problems.

.32 Well, there is some vagueness here: T. problem of solving an eq. is a definite problem to be solvd. Hvr., t. problem of acquiring t. concept of "equality" is not so easily ~~be made~~ defined as a "well defined problem". — well, in t. case of "equality" "acquiring t. concept" means knowing each of its properties & how to use those properties. ~~Probly its~~ In t. case of "equality", t. no. of properties & assoc. heurs is very large, & most of them will be acquired by TM only after [redacted] much try. Hwr. t. basic well known propertys of equality — I can list (equivc. relation & substitution) (3 properties), TM "understands" these properties if he can use them when needed in various problems.

.04 SN Q.: Can I put most (or all) problems in t. form: Eval (expressn)?

or find $x \ni$ Eval (expressn) $= \phi$ ? or Eval (expressn) = T.

Consider t. prob: "Is it true that $\forall x,\ F(x) = \phi$ ?". [F is a certain given function]

If x is restricted to positive integers we can write this as Eval $\sum_{i=1}^{\infty} |F(i)|$

If x is real, ≈ Eval $\int_{-\infty}^{+\infty} (F(x))^2 dx$ if its ≠ 0 or = 0. Hwr. say $F(x) = 1$ if x is rational, $\phi$ otherwise — I. S form wouldn't be true. see 99.01 for more on this

.13 Consider t. equivc. propertys: TM would be able to solve:

[redacted] a = 3, b = a, Eval b [redacted] ? } This is 2 eqns in 2 unks.

a, 3, = → T ; b, a, = → T ; Eval b → ?

1 p = 40 × 64 = 2560 = 1.5 k bytes. 16 k = 10 pp. 128 k = 80 pp.

.20 [redacted]

.21 63081 .04 is an attempt to put all problems [redacted] into a standard form recognizable by T.M. — It may be possl. to figure out such a way: In t. Q.A. formalism, t. "Q" contains t. entire dscrn. of t. problem: but its not clear just how this can be expressd so TM understands it. For t. time being, try to keep t. "nature of t. problem" simple; Dscrb. each problem, assume TM knows what t. problem is, & then I have to write out (a) solution(s). Later, group t. problems together & see if I can find a small no. of problem types, so I can easily tell TM what t. problem is. Actually, even if there are a large no. of prob. types, this may cause no big trouble.

Def .30 O.k. some problems: 1) Eval 3,− ; Eval 3,4,* , Eval 3,4,** (3,4** means $3^4$ (Fortran).)

Eval (some complex expressn ··· purely algebraic a/o partly or wholely Boolian).

2) Find $x \ni$ Eval (expressn) $= \phi$. This could be soln. of "single equs." but also several simult eqns. e.g. find x, y $\ni$ { $3x + y = 2$ & $4x + 3y = 1$ } This can be expressed as a single partly boolian e.g. or a single alg. eq. [ $|(3x+y-2)(4x+3y-1)|^2 = 0$. ]

(x & y are "tables" = Dummy variables?) — so this is mindful of t. "λ" notation.

Both 1) & 2) can be put in "xfmn form" (t. standard form for N & S's GPS)

"Eval" can mean: "subject t. expressn to a string of t. folg. permissable xfmns. Stop when a pure no. is obtaind."

in 2): Subject these 2 expressns to t. folg. legal xfmns. until you obtain expressns of t. form x = a no., y = a no.

On t. other hand we can have some sort of simple notation for desbng t. problems & have TM learn what t. notation means: So we can either have recognition of t. problem descn. built into TM, or have him learn it. Start out w. a small no. of Built in ("primative") problem types.

In my recent work on learning Alg notation: we could look at it as TM having this primative set of ~ machine instructions. T. problem is always to devise a string of them that is applied to t. Q, to yield t. Answer.
As TM learns, he finds that certain sub-strings of insts. are useful, (e.g. $n_1, n_2, +\rightarrow$ inverse of machine operations (substitution)). & they are added to t. list of machine insts. & the pc's.

---

SN "Eval" can also have in its range, a string: e.g. Eval a, b, bccb, sbs → accb (perhaps — depending on what defn. was being used for sbs).
A string can contain pure nos. &/o other symbols.

---

Actually, I think this "various forms of input problems" is not difflt.: in fact, I think I had this soln. some time ago. TM is a "Q.A" machine: It has to learn t. relation betw. Q & A, by being gn. examples. In a certain set of cases, it knows in a genl. way, what has to be done, e.g. when it was given {$n_1, n_2, +$} nos. as input, it knew that it had to find a string of operators to xfm this into t. output.
It can be gn. ways to interpret t. "name of t. problem type" ... which tells TM what has to be done. — or it can learn to do this interpretn. To start off, I will give TM this "name of prob type" interpretation info. Later we will have it learn more complex prob. types. I suspect that there will not be more than 10 such prob. types — & certainly not more than 20.

---

I got into this "problem descn" problem from 83.32ff. I wanted to give TM probs. → their solns would give T.M. various concepts — like t. properties of "equality", for instance.

One approach is ~ 84.13: we give TM info like
.28 a = 3 | b = a, then we ask "what b =". This could be obtained by subsn.
In fact substitution implies transitivity; if a = b & b = c then a = c by subsn.
This subsn. can be done either of 2 ways.

a = b
b = c
→ a = c

Also, we can almost prove ∀ a, a = a: say a = b, where b is any other thing that a is = to. Then a = b again! subs a for b in gives a = a.

a = b
a = b
b = a

If we have proved transitivity & we assume commutativity then a = b implies b = a & transitivity gives a = a.

In addition to t. method of 85.28, say TM ~~has~~ has learned that 3+4=7 and 1×7=7. "would it be able to get to 3+4 = 1×7 from properties of equality?"

.04 So: what is t. problem? well, say I have written out this "English" soln. of a problem, & this soln. ~~needs~~ requires that TM have certain concepts (say properties of equality). How can I give T.M. t. needed info? ① One way is by giving it as a primitive. ② Another is by giving it by means of a trg. seq. of acceptable cost?

10:53:17 SN On Heuristics: During "meditation time" betw. other problems, TM trys to recode t. past in better ways. One such "better way" is a (prob. solving) coding method that reduces t. cost of solns (on t. average). A good "Plan" would do this. Incorporating this "plan" into TM would involve giving it a pc (sometimes a conditional pc — depending on t. situation that arrises) as well as defining it as a pgm. One could then estimate t. amt. of ↓ of cost of ~~the~~ solving problems of t. past by using it to solve sample problems of t. past — or by various methods of theoretical analysis.

Note that cost considers both pc. & cc.

try

.19 well, try ① first! Consider (vaguely) substitution as being a reasonable thing to try for equal objects. {Maybe not such a good example, because if TM knows what subsn. is, it has probly learned it by substituting numerical values for alg. expressns. that they are = to.}

Anyway a "primitive" way to do this, : If we know a=b, then / t. operator a, b, (arby expressn) Sbs has pc > ϕ. — just how large pc should be is unclear. We also have to look to see that b (say) ~~~~ occurrs in t. "arby expressn" so as to save cc.

Hvr., usually (if not always) this subsn. is controled by some auxilliary goal — otherwise one would subs a for b then b for a then loop.

.28 Another way to look at this: Say TM has somehow discovered that subsn. of ~~~~ something for something that it is = to, preserves t. "value" of an expressn. Then, TM should ask — How can I use this fact? in what sorts of probs will it be useful, & how can I (at low cc) recognize such problems? In GENERAL: TM should always do this (if there is cc available) after it has solved a problem. Also, it should ask: how could I have used this fact in t. past? could I have worked t. probs. of t. past more easily now that I know this thing? This is sort of in t. spirit of .10–.19

.33 In dealing w. Human students, a particular concept can be taught by giving students a great variety of problems in which various aspects of that concept are needed. Hvr. .28–.33 ~~is~~ is also very impt.

Well, then, lets regard this as an ~~~~ well defind ~~~~ problem: To teach TM t. various properties of equality, so it can use them in various problems.

Some of t. most impt. ideas about "equality" involve literal expressns — which is something I haven't gotten into yet. In fact, it seems rather diflt. to give problems involving equality if literals are ~~not~~ not usd! "" e.g. If 1+4=5 & 2.5×2=5, then 1+4 = 2.5×2.

Another property of equality: If $n_1 \neq n_2$ then $n_1 \neq n_2$ (i.e. $n_1, n_2, = \to F$)
We can use this property of equality to give various "equality problems." — perhaps problems not using literals.

E.g. Problem: does $3^2+7 = 75 \div 7$ ? This can be solved by noting $3^2+7 = 16$ ; $75 \div 7 = 10\frac{5}{7}$ ; & $16 \neq 10\frac{5}{7}$. because $16 \neq 10\frac{5}{7}$

(Tho there can be diftys w. this in a finite accuracy machine!).

16 k = 
# 8
# k/byte

- Another property of Equality: t. same function of ='s are =.
- Also for every single valued numerical expressn, there is a unique no. that is = to it. / so, to find if 2 things are =, find t. nos. that each one = to, & find if t. 2 nos. are identical. Tho: 0.1111111.... = 1.0000 so identity is not essential to equality for nos.
- So w. t. 3 equivc. properties & substitution: I should be able to make up examples (for a human) to illustrate each of these ideas.

Make up examples for, say, Grace!

If 2 nos. are "=", their difference = $\phi$.
$a-b=\phi \Leftrightarrow a=b$

7/81 **Methodological Note:** "Stick to English" again: In this case, Say we have written in English, a soln. to f "Eval" problem. This soln. involves t. student understanding certain properties of "equality". We then write (for a hypothesized human student) several examples that illustrate those properties of "eqlity". ~~[illegible]~~ It may be that after seeing these examples, t. student is expected to generalize them. If so, then we must be sure that TM would also be motivated to do such genzn. & would know how to do it. I think this process of writing ~~[illegible]~~ TS's & PTS's as if they were for humans, & then refining these solns., is a v.g. idea. — it makes writing ts's for TM much easier (for a human to write, that is!). .29 → 88.17

.30 One [redacted] difty w. writing ~~[illegible]~~ TS's for humans, is that one would want to use RW examples & use "story problems" for examples. This is n.g. for an infant TM, — but I can probably find enuf non-RW probs. & examples to do what needs be done.

7481

.01! 82.26 → One rationale of giving TM concepts viz Trg. sequs., is that t. acquisition occurs in "more generalized form." than otherwise! i.e. Having found a soln. viz trg seqs, it is more likely to be able to solve problems ~ to t. problem actually solvd.

An alternate way of getting this "genzn": To give TM t. soln. directly or viz a short T.S., Then tell TM to genz. T. soln. as much as possl. This Genzn. of any soln. was pretty much what I usd to do when I was learning to solve probs. — See 73.27 on "genzn".

In fact I think t. reason I felt T.S's w. hy cjs's were better than solns w. small cjs's was that TM did, indeed get a more general soln (i.e. abss. that would solve more probs) w. t. hy cjs. solns. This need not always be true, but mite tend to be true for properly designed, hy cjs, trg. seqs.

.16 → 91.07

| 947P |
| --- |
| Maybe necy to press "fast" at least once. |
| No |
| 950 → 10P worked ok. w.o. fast |
| No 1001 → 1005 or 1006. |
| OK. 1006 → 1016. |
| 1028 → 1037 |
| workd |
| 1037 → 1045 |

.17 : 87.29 Int. spirit of 87.20 ("English"), I'll try to write in English what I'd say to a TM that could understand this English, in order to give him impt. concepts needed, leading to soln. of single linear, Then simult linear equs!

.20 Each string of symbols often has a no., "called its "value" assoc. w. it. Some strings have no "value"/ ≢ φ value, others have > 1 value (This may be unnecy to say).

The value of a string can be obtained by having a string of operators operate on t. string.

.23 I will give you some simple strings & Their values. You will try to find a string of operators that act on t. string to give its value. You will try to find such a simple string of ops. that works on all strings I give you. ⟨Hvr, optionally; see 93.33

[ Give examples thru binary functions (numerical only, not Boolean yet) ]

T.M. now "knows" how to get "values" of certain strings.

[ Next, t. concept of "equality": ]

.26 Two strings that have t. identical value (≡) are said to be "equal". i.e.

$(\alpha, \beta, = \to T) \leftarrow ?$ (Just how to express this, is not clear.)

.29 Then for any string $\alpha$ that has a single value, $\alpha = \alpha$.

If for single valued strings $\alpha$ & $\beta$, $\alpha = \beta$, Then $\beta = \alpha$.

.31 " " " " " $\alpha, \beta, \gamma$, $\alpha = \beta$ & $\alpha = \gamma$ Then $\beta = \gamma$.

.32 [ If ~~an expressn. is substituted~~ ~~then~~ an = express. is substituted for a gn. expressn. in a third expressn, t. result will = t. third expressn.

.33 ⟨for .32 ~~t. idea of substituting~~ a. if we use RPN, Then substitution of an expressn That has a "value" is always legal.⟩ — I guess we could use "subsn." as a primitive.

.01 Next, give TM an expressn to find value of that involves ∆ 39 min: a.g.
.02 2 subsns. — e.g. 3, −, − Now I could give TM only t. subsn. rule of 88.32−.33 — 1045 1053 NO
.03 but in narrower form! i.e. {If expressn. $\alpha$ occurs in expressn ∆ = 8 n.g.
$\beta$ as a substring, & t. value of $\alpha$ is $\eta$, Then if $\eta$ is subs. for $\alpha$ in $\beta$ 1053 → 1102
.05 giving $\gamma$, Then value of $\beta$ = value of $\gamma$.} 1118 → 1120 wait till 1127 or 1128
.06 The advantage of t. more general form of 88.32−.33 is that t. genl. form will enable TM to eventually solve more diff't. probs. Hvr.,
.08 t. more genl. form will (usually) make t. actual correct soln. of lower pc.

Another Q is whether to include t. other properties of equality (88.29−.31) — t. remarks of .06−.08 hold a fortiori.

.11 Say TM has learned unary & binary funcs (w. names). — so he can get values of some simple strings. We then give him .03−.05 & t. (2 subsn) probs of .01−.02 — Like 3, −, 7 & 3, 4, +, − & 3, −, 4 + . What happens?

For one thing, I'm not sure about just what TM's generalization Algm is in t. present case. 90.01 91.30 95.27

.21.
.22 : 7.6.81 81.34 → : A) Conc. mind using uncon. mind as srtn: examples: ① recognition of face of person is unc. mind srtn. used by conc. mind ② evaln. of chess positions may be unconc. but division into conc. unconc. in chess is unclear, ③ finding distance, direction of objects by visual & acoustic means: are unconc. srtns. → .31

B) Unconc. mind using conc. srtns: ① "Get more info from RW" is conc. srtn. ② Manipln. of alg expressn. into several forms. Done by conc. mind. Results used by unconc. mind. ③ "Curiousity drive" sometimes could be command by unconc. to conc. mind to investigate certain aspects of something. (choice of projects for "basic rsch" could be of this form & couldn't be explained by conc. mind.) ④ In general, conciousness behavior that one can't account for very well rationally could be use of conc. as srtn. by unconc. mind.

.31 7.7.81 → I used to use my uncon. mind as srtn. by working on a problem before going to sleep. Befor sleep, I had to have t. problem clearly in mind ... preferably as a "well defined problem"; Also I had to have all of t. relevant facts in mind; Also, I had to really want to solve it. ..... When I woke up, I'd either have a soln. to t. problem or an impt. new clew for its soln. — usually something new to work on.

One impt. Q: on awakening, is t. info in Short term or Long term Memory? — i.e. if in STM, then if I don't put it in LTM by immediate rehursal or by writing it, I will lose it.

91.01

.01 : 89.21) : Remember "knowing a concept" means .. knowing how to use it in various circumstances. All this means is knowing t. conditional pc's. (unconditional pc's are a special case of conditional p.c.'s). I think normally, "conditions" are evaluated by an "OB".

SN An Ob usually has a single Boolian T/F as output, but it can have a lot of bits as output. In fact, it can have a binary no. as output .... all t. bits of a binary fraction. This set of bits can be used either as a set of TF statements, or as an approxn. to a Real no. — e.g. One could use a boolian operator to ask if this "Real" is > .037 — it maps an/set of ordered boolian vars into a single Boolian variable.

---

→ Things that are "conditions" for "condl probty": (1) Nature of t. problem (or sub problem) (2) state of partial soln. of t. prob (or sub-prob). In t. case of GPS type probs., t. "nature of t. prob." is defined by t. final goal s and t. set of permissible xforms. T. state of partial soln., is t. one (or several) xformed object(s) than one has now to xfm. to t. "final goal" form..

.19 Condl. pc. tells when to use a concept. .... it is a complete statement of when to use it. → One kind of condl. probty that is very highly conditioned by "T. Type of problem" is condl pc of various "PLAN"'s. → Note 91.30

---

.20 SN On Giving TM a soln. to problem directly. T. best way to do this is to factor t. soln. into concepts that would have been adequate if TM did a regular Lsrch using that augmentation to its set of concepts. Ordinarily, hvr., one doesn't know what condl. pc's to use (i.e. quantitatively, t. conds under which to use those concepts — a complete quant descn. — see .19 ↑ ). If these concepts are all concepts that TM already has, and has adequate condl. pc's for them, then this "given" soln. is about as good as TM solving t. problem itself ... (tho by solving itself — it gets better condl. pc's and it may find a better soln. than t. one we are giving it. — In which case it would be better to have it have both solns. — i.e. let it search for a soln. for a reasonable time. If it finds t. soln. I would have given it — let it end at that pt. If it doesn't find it in acceptable time, and/or finds another (better) soln. in that time, then give it our additional soln. in as factored a form as possl.

Another possy: If TM doesn't yet have all of t. concept factors needed for t. soln., "give" it only those needed concepts and let it find t. soln. by conventional Lsrch.

7781 TS

**Methodological Note** 91

.01: 89.40 Uncons. Mind, cont : In line w. these ideas: (an old idea) be sure to know all t. time, just what t. most impt. problem(s) being worked on wrt TM, if any. Each day, be sure t. progress has been integrated enuf, so I can state t. problems in as "well defined form" as possl. If there is >1 problem, be able to list them in order of general impte. & order of ~~[illegible]~~ urgency.

.06 → 112.01

.07: 82.30 / 88.16 SN why we would rather have TM learn various (& fund) concepts like "eval" rather than being gu. them as primatives. Having TM learn these concepts means it must

.09 have t. more fund. concepts to base that learning on. One of my fears is that I will not put certain impt. concepts into TM & that these concepts will be

.11 essential for solving various impt., diflt. probs. By being sure that TM can learn, e.g. eval., I must put/have put impt. fund. concepts into TM, (i.e. basis of prim. concepts) & it becomes less likely that I'll run into t. difty of .09–.11.

.19

.20: 83.20 : Methodological Note: on 83.17 I was thinking of a system to solve t. problem of devising TS's for TM. As decd at that pt. T. system would use me as a srtn. As I read it now it would use my concious mind (89.22, .31) as a srtn., & I in turn would use my unconcious mind as a srtn. to help solve these probs

.24 – using t. ideas of .01–.06. → 112.01

.25 In line w. .01–.06: **What is t. Present Problem?** :

Immediate backgnd: 80.26 – 81.10; 83.09 – 84.20; 86.04 – 87.40; 88.17 – 89.21; 90.01 – 90.19.

(83.09: e.g. idea for a system to devise TS's.)

.30 89.21 / 90.19 : T. general "Action Algm." of TM: This consists of devising strings of concepts in pc order, taking conditional pc's into account, then trying them out on t. problem. The "condl. pc's" take into account t. nature of t. immediate problem & state of partial soln. of t. present prob. (q.v. 90.01–.19)

(not exactly; often t. next element in t. string can't be chosen until t. previous one is executed.)

In view of .30, consider t. prob. of 89.11: If TM has been gu. t. concept "subsn" it knows t. conditions under which to try it..... Including t. nature of t. prob. & t. present state of soln. In t. situation of 89.11, T. problem up to that pt. has always been (& continues to be) evaln. of expressns. At that pt., an "adequate" understanding of subsn. would be to look for (ob) a substring that one knows how to evaluate, – if one is found, then substituting its value for it has pc ≈ 1. If no such substring is found, subsn has pc = 0, & other xfmns (if any) have pc's > 0. Thus others xfmns may all have condl. pc's = 0, in which case TM gives up. or sees it t. present state of soln. is an acceptable soln.

Hvr., if t. pc's are empirical, pc's can't be zero. Only in t. case of subsn., where subsn. is impossl. if there is nothing to be subs., can we get a pc = ∅.
Or if t. pc is externally given as ∅.

Another impt. Q is how t. pc's get modified w. experience.

Also: What about Obs? when are they usd? Do they have pc's or are they always usd algorithmicly — i.e. w. pc = 1 or ∅? In t. case of t. ob. "see if any substrings can be evaluated" — This is a rather complex op . . .

.10 → how could it have been invented? → In general: how are obs invented?

Another possb is that TM isn't really that good at evaluating unary & binary expressns — so it really doesn't know where they begin or end.

e.g. consider t. substring 5, 3, 4, +, 6   TM's method of evaln.
.16 might be: "Look at t. first no. in t. string. If it is followed by a non. no., do t. operation assoc w. that non-no. If there is a no. following t. first no., do t. operation corresp. to t. next non-no. on t. first 2 nos." — in which case it would give 5+3 = 8 as
.20 t. evaln. of this expressn. T. correct "evaln." is t. 5, 7

→ One way of dealing w. pc of obs: also, maybe, how they are invented!
We look at an ob as an obligatory part of t. code to be inserted into a cmc that occurs befor any one of a set of symbols of t. code can occur. This ob determines t. condl. pc. of that set of possl. symbols.

.26 → A subsn. algm. that would work o.k.: "starting from t. left, move rt., & find t. first symbol that is not a no. If its a unary op., evaluate it & t. no. befor it
.28 & subs.; If it is a binary op, eval it & 2 nos. before it & subs."
Actually, this subsn. algm. could be learned rather directly. {see 94.01 for Notes on Backtracking.}

$$F^2(x) + A(x)f(x) = G(x)$$
$$F^2(x) = F(x)\cdot A(x)$$
$$\frac{F\,F(x)}{F(x)} = A(x)$$

.30 At this pt. we could get involved w. "Backtracking", if t. subsn. algm of .16 – .20 was usd. — we'd have to backtrack till we get to t. algm. of .26 – .28 perhaps — or any other one that was as good. Hvr., at this pt., I think TM's "teacher" should be careful that early concepts are learned properly so that little or no backtracking is necy.

→ T. problem of how obs are invented! .10 seems to be a serious problem. Ordinarily new trial obs are made by combining hy pc obs & ops. of t. past

7781TS



.01 T. Apparent state of t. problem at present:

See 91.25 for Bibliog review! 83.09 is ev. ideal. 88.17 — 89.2[redacted] is v.g.

If this set of probs is not enuff, see 80TS 206 for possl. extensions of this T.S idea

[redacted] T. [redacted] T.S. I'm thinking of is alg notation (Eval) in RPN, than single linear ≥ than several simult. linear eqns. 88.17 ff is a start on doing this (TS) at a hy level in English. Say I was able to do this in English ≥ put in arbly more detail when required by TM for acceptable solns.

for a hypothetical ≈ Human student

Next problem is how to ~~do~~ implement each of t. learning tasks for each concept. Normally, [redacted] [redacted] t. problems gn. to TM will be of several difrnt forms. I have to figure out a way to get TM to solve each type of problem.

The general action algm of TM will always be of this same form (91.30)

.16 The condl pc's (→ see 90.01 — .19 for discn. of condl. probys.) [redacted] (≥ all pc's will be somewhat condl., since they will usually depend (at least) upon t. nature of t. problem) — will depend on t. [redacted] nature of t. problem (e.g. t. "Type of problem" say: General GPS, or solving

.18 an eq., or 84.30 — .40 [redacted] or "find a string α ∍ F(α) is (true/false)" or, "find t. shortest string α, ∍ f(α) is true" or "given α [redacted], β, δ are true, (α β δ are gn. strings), find δ ∍ F(δ) is true".

.19 e.g. if 3x+y=0 ≥ 5x+y=7 what is x, y?

→ see 99.01 — .40 for a list of problem types

.20 • Well, this, then, is somewhat of a problem! Say we get t. pc of a certain concept under one set of circumstances (e.g. problem type 1). Then what shall its pc. be under difrnt. circumstances? Well, use ≈ [redacted] Z141 formulation: [redacted] we collect data on that abs. ("concept") for various conds. If t. ssz for one cond. is zero, we pool data from other conds. As ssz ~~for~~ for each cond ↑, it gets more wt. for its own. type of "condition".

Also: There are some cases in which t. pc's for t. difrnt "conditions" can be related "logically" — so t. info on them can be pooled more effectively. [redacted] This is not done in Z141, but this "logical analysis" mite be applyd to

.28 Z141-type probs. to get insight on how to do this.

So, it would seem that I have most of th. [redacted] diftys under [redacted] (Some) measure of control. I should then try to review the entire soln. as I see it, in some detail, so as to clarify just what work needs be done.

.01 — .19 is somewhat of a review.

10,35 00

Action Algm.

≈ 11°c'.

= 97.6°F

— 11 × 9/5 +

97.6 = 77.7

of ≈ °c' × 9/5

+ 77.8

.5°c'

= 77°.F

| 9/5 c' | °c' | F | Δ |
|---|---|---|---|
| .9 | .5 | 77 | 76.1 |
| 19.8 | 11 | 97.6 | 77.8 |
| 8.1 | 4.5 | 82 | 73.9 |
| 6.3 | 3.5 | 83 | 76.7 |

so Δ = 76.95 = ΔT

so ≈ 77°.

so °F ≈ 9/5 c' + 77.

3.8°c' 12:28P

~ 84°F only.

.33 [7.9.81] On dealing w. various (TYPES) kinds of Q's. (.18 — .19): an Alternate means!

Normally, one ~~test same~~ is looking for t. same op. string to solve all prob. types. This involves conditional probys (.16 — .19). An alternate way: for each problem type, one has a difrnt. string of ops. That is supposd to solve all probs. of that type. Ordinarily I think it is trivial to decide what "type" a prob. is. TM can probably be told this! i.e. each problem can be coded w. its "type" index when I give that prob. to TM. Pooling of data (condl pc's) of abss for difrnt prob. types can be done like .20 — .28.

→ Actually, it seems ~~[illegible]~~ that [redacted] .33 ff can be made a special case of t. previous method that uses only 1 string to solve all probs. T. first thing t. single string does is decide on t. problem type; then it allocates t. problem to a suitable sytn!

7.9.81

[redacted] TS



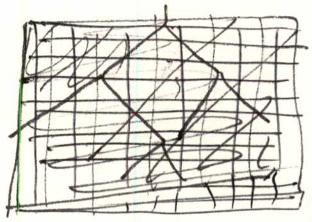
7.9.81

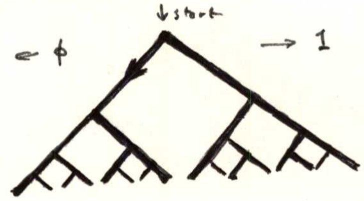
**BACKTRACKING!** One V.G. Model for backtracking occures in normal Search for Induction Codes.

In "normal srch" one uses, some stop rules to terminate potentially infinite compns that produce no output.

One tries an input string to Umc, M, — all zero's to start. As soon as an incorrect bit comes out (i.e. ≢ t. proper bit of t. thing one is searching t. code of) one tries t. opposite bit of t. last bit tried. If it produces an error, one backtracks one bit & tries t. opposite of t. one tried at that pt., etc.

This is just straightforward tree search. Go down, & keep to t. left as much as possl.






We assume that t. Machine has t. "sequencial property": That it is a "process":

I.E. If [redacted] $M(x) = \alpha$ Then $M(x^\frown y)$ must be of t. form $\alpha^\frown\beta$.

$\beta$ may $= \Lambda$. Hvr., note .22 !

---

T. Q is: Can I use take any situation in which backtracking is needed & put it into this form? Or Can I use t. foregg. backtrack model as a heuristic guide to solve all (or even most) other Backtracking problems?

---

.22 **N.B.** It may well be that in Lsrch t. order of trials in Backtracking is considerably modified by t. various (condl) pc's involved. — So superficially, it would not look like ordinary Backtracking! One ordinarily does not try to chose ϕ's first, but rather t. branch of max pc.

96.38 — on a possl. applcn. of "Backtracking" of Prissort.

Would **Bourbaki** be a good source of tng. seqns.? Its supposd to be a very well written set of books ... introducing new concepts in a very logical order — — But not necly t. order usually taught in schools.

30% vs. 20%

McKean
Smith [redacted]
Wylie [redacted]
Tower
Senn.

---

I think t. main prob. may be simply writing t. "English" Tng. seq. & being sure that all of t. relevant concepts have been explicitly Tht of.

---

7.10.81 I've been thinking of all probs being solved by t. same single (string) operator (or in 93.33 a difrnt string for each "Problem Type"). For problistic induction, hvr., we [redacted] will have a set of strings each w. its own pc. This set of strings can be t. output of a stoch grammar (say, as in Z141, perhaps ≈ a SF. stoch grammar). T. Answer to t. Q posed to TM, will then be a stochastic set of strings that are t. outputs of t. stochastic set ~~of~~ operators.

15R→1.5k →4.5k

Laxton's Progress w. pers! v.g. ~~[illegible]~~ to a cst

---

**SN** T. ALPHA-BETA heuristic is normally usd for 2 person GAMES & can reduce search trials by a factor of $\sqrt{N}$ (at most) (N is original preheuristic no. of trials). Could one use some ideas in this heur. to reduce no. of trials in finding induction codes? Study t. α-β heur. w. this in Mind! Perhaps think of non-gaming tasks as "Games ~~[illegible]~~ w. "Nature" as t. opponent". This is sometimes done via game theory. It assumes a maximally malevolent "Nature" & is certainly very conservative. But if it gets us $\sqrt{N}$ — it may still be worth while! Note: $\sqrt{N}$ means we get Twice as much length of uncoded corpus that [redacted] we can code for [redacted] a gn. cc.

---

.27 89.21 ~~[illegible]~~ branched off into an impt "aside," on condl. pc's. Th. conclusion, is that: If we gave TM 89.03-.05, then we would also have to give TM its condl. pc's., along w. t. Ob. that gave rise to t. condl pc's. Hvr., 89.03-.05 is not an "xfmn having a condl pc." in t. usual sense; It is a kind of "fact" — a "thrm". Such "facts" can often be somehow converted into xfmns & assoc. condl. pc's. — or, such facts mixd w. other facts & xfmns can produce new useful xfmns & assoc condl. pc's. (cpc's).

Def cpc = conditional pcost[redacted] cpcs = plural of cpc.

---

In many problems (E.g. GPS-type probs), there is a set of legal xfmns that one ~~[illegible]~~ is allowd to use on a partial soln. string. ~~One can~~ T. user of TM (≡ "client") can ~~[illegible]~~ give TM xfmns to put into that set. This can be done with or w.o. giving TM assoc. cpcs. If no cpc's are gn., a default uncondl. pc. will be given corresponding to t. "principal of indifference". ~~Or it may be that t. xfmns in that set have no~~

Well, say TM has a list of "facts": How does he use them? Given a new fact, how does he find out how to use it?

200 K BBl
$ 6M

E.g. T. properties of equality: ( equive. properties + substitue).

108

On t. other hand, if TM "learns" t. properties of equality ~via suitable T.S.~, perhaps they will be automatically in a form TM can make use of.

25 primes: 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, 41, 47, 53, 59, 61, 67, 71, 73, 79, 83

It may well be that ordinarily, giving a human a "fact" is not a very good way to [redacted] help him — that it is normally a very indirect hint. — That xfmg [redacted] "facts" into usable form is normally a fairly diff't. problem. I.E. knowing "facts" & knowing how to use t. "facts" are 2 quite difrnt. things!

Getting back to "equality": How to get TM to know things that would be equivt. to "knowing about" or "understanding" equality:

---

Hvr. "facts" may still be of interest — i.e. they may be a sort of "primative form" of info. that can be xfmd into somethings useful. Hvr. it can be xfmd into various things for use in various problem types.

→ One mite want to code t. "facts" in an optimumly short code or set of codes

-.09
8.2

---

O.K.: consider t. "facts" about "equality" ( 3 equive. propertys + subsn).
Say T.M. knows how to take some (expressns) strings & get numbers from them that are t. "values" of those (expressns) strings. ~~those~~

+1 =
-1 =
8.-09
7.9
6.4

---

SN: say TM knows how to evaluate $n_1, n_2, +$. Then ~~then~~ we can generalize this evaln. by allowing $n_1$ [redacted] a/o $n_2$ to be an expressn. that has a "value".

→ 7.
6.5→7
6.4→0

This may be a standard way to generalize! i.e. allowing 1 or more parts of a [redacted] rule to be something more general than they were originally.

$6.5\times10^{-9}$ +1 -1 → $7\times10^{-9}$
$6.4\times10^{-9}$ +1 -1 → 0.
Casio

---

SN: Another way to think about it: $n_1, n_2, +$ is defined so that $n_1$ & $n_2$ are expressns. that have "values". From this, the recursive defn(s) follow — particularly when $n_1, n_2, \left(\begin{smallmatrix}+\\-\\\times\\\div\end{smallmatrix}\right)$ are defined.

T. way ~~one would~~ to teach this to T.M.: teach $n_1, n_2 \left(\begin{smallmatrix}+\\-\\\times\\\div\end{smallmatrix}\right)$ for $n_1$ & $n_2$ being numbers. This results in a certain pgm. Po w. an expressn ~~as in~~ to be evaluated, as input, & a value as output. Next, we give examples in which $n_1$ a/o $n_2$ are simple expressns evaluable by Po.

.38 T.M. must ~~then~~ modify Po so that it can evaluate these. This amounts to a kind of Backtracking: (see 94.01 on perhaps optimal Backtracking). Here we want "minimal modifns of Po" as trials for t. more genl. expressn.

.01 In t. spirit of 101.29–.31 I could go thru 88.17 ff. using any kinds of Q's I liked. This is a good way to do it ~~promupt direction to go in~~ since it would seem to make it a lot easier for me to write TS's. Ultimately, I want most of my creative energy to go into t. construction of TS's — That these TS's should be pretty much designed for "RW-[redacted] (innocent) (deprived) Humans." — That putting these ≃Human TS's into a form for TM should be an essentially routine process.

SN Re: ① "equality" in 88.17 ff., t. only quality of equality that I actually use is [redacted] subsn.

.11 ② I may want to use equality in a more genzd. sense: That 2 strings can be equal — whether or not they have values or could in principal have values. E.g. 2 strings can be equal because of subsn.

.14 i.e. If $\alpha = \beta$ ($\alpha, \beta$ strings) then $\gamma = \alpha, \beta, \gamma$, subs. (for any string, $\gamma$.) This form of "equality" makes it more useful for manipulating literal Algebraic expressns.

Properties of equality: ① if $\alpha \equiv \beta$ then $\alpha = \beta$. I guess this is t. same as $\alpha = \alpha$. ② $\alpha = \beta \supset \beta = \alpha$; ③ transitivity ④ some form of subsn. postulate. ($\alpha, \beta$ in .14)

Hvr., I'd like it to be for any subsn. of an = expressn. [redacted] So far, I have defined subsn. only for, say t. first occurance of t. thing substituted.

5.38P: (Refrig. tried to go on. actually may have gone off.) 5.40 ref did go on ~ 20 sec after I turned it off, then on Max (≡3). 6M. 5.47. It just tried off or on! or both.! 6m. 5 49P on, 5 55P maybe off, 5 57:53on, 6:04:off 20 sec, 6:08:20 on 2, 6:12:25 off, on 6:14:31 off, 6:20:37 off, 6:28:06 off, 6:30:51 on, 1:50 p 100 on

.22 Some possl. forms of $\boxed{\text{subsn.}}$, $\alpha, \beta, \gamma$, sbs [redacted]: ① $\alpha \leftarrow \beta$ in t. first occurance

.23 on t. left in $\gamma$, of $\beta$; ② Same as 1) [redacted] but first on Rt. ③ $\alpha \leftarrow \beta$

.24 in the $n^{th}$ occurance of $\beta$ in $\gamma$ from t. left. ④

Q: if $\beta$ doesn't occur in $\gamma$, [redacted] what is value of $\alpha, \beta, \gamma$ [redacted] sbs?

I'd like to get a useful notation for subsn.: some of t. operations I may want to do w. it: [redacted] subs. all occurances of $\beta$ [redacted] in $\gamma$ by $\alpha$.

[redacted] say $\alpha, \beta, \gamma$ sbs [redacted] means ① (.22) Then {$\alpha, \beta,$ , sbs} can be regarded as an operator. If we apply it to $\gamma$, ∞ times, (i $\alpha, \beta, \gamma$ sbs $\equiv \gamma$ if $\beta$ isn't a substring of $\gamma$) Then we get
Another way would be a "Do" loop" using t. "n" notation of .23–.24.
We may want to change [redacted] order of argts. in "sbs" function, to make applicn. of sbs, ∞ times, [redacted] easier to do. ⑤ subs. t. first discovered occurance in $\gamma$ by $\alpha \leftarrow \beta$

Suppose I want to change t. $m^{th}$ symbol of type T ~~[illegible]~~ in $\gamma$ to $\alpha$.

.26 Well, best leave t. details of t. sbs notation "open" until I find which kinds of operations I will want to do w. it.

SN Another thing sbs is used for is in "production systems" like c f Grammars or even concext sensitive Grammars.

7·15·81 TS



Going back to 88.17 in t. spirit of 102.01, & 101.29–31.

Line 88.20: For a human, this is (perhaps merely?) to get his head in t. rite place: to get t. rite part of low access memory into rapid access memory.

An "introduction" .... — But maybe there is more content!

.05 On t. other hand, if "function" is one of TM's primitive concepts, 88.20 says that "Value" is t. name of a funct. from strings to nos. It is sometimes single, sometimes multiple valued, & it's domain is >1 string but not all strings.

445P: NACL 595P. 9P 10:40P

If .05 is t. case, then after I have given TM. this info, it has certain Facts in memory. We might put these facts into a "Semantic Net".

Hvr., for this particular case: we mite just store t. fol. facts:

1) "Value" is a function. 2) Function: is a set of objects of which "Value" is one. it is in t. set of functions.

3) Because "Value" is a function, it has a domain & range. In this case, they are strings & nos. resp. (T. range can be other things also: say T/F (Boolean) — & perhaps strings, but I'm not sure of t. Utility of this last.)

4) T. funct. can be single or multiple valued. (Range).

5) Domain: >1 string, < all strings.

$c = d$

$a = b$

$f(a) = f(b)$

$f(a,c) = f(b,d)$

From this info only what kinds of Q's can TM Answer?

"Value" has a property list. T. first property is that it's a function. Then, referring to "function", t. next properties are about its Domain & Range

$a = b \quad a \neq b$

Line 88.23: From th. solns of these probs: TM learns that $+, -, \times, \div$ are all functions & it has some (tentative) functional forms for them. In all cases thus far, t. Domains & Ranges have been pure numbers.

$a = b \supset b = a$

$a = a$

$a = b, b = c \supset a = c$

$a \neq b, b \neq c$

(SN) looking at it from an ~ human pt. of view: We know how to "evaluate" $n_1, n_2, +$ if $n_1$ & $n_2$ are numbers. Now, if $n_1$ is $n_3, n_4, \times$ & $n_3$ & $n_4$ are nos. then we know how to evaluate part of $n_3, n_4, \times, n_2 +$ or we don't know how to evaluate, but if $n_3, n_4, \times$ were a number, we would know how to evaluate it. There is a no. assoc. w. $n_3, n_4, \times$ — so trying that no. to replace it, is reasonable.

$6.5 \quad 10^{-9}$

$6.49999 \quad 10^{-9}$

Is it reasonable to try to get TM to do reasoning of this sort?

7.15.81 TS

Rite now, I'm a bit confused about t. meaning of 3, 4, + & 3, 4, +, 5 x.

3, 4, + is a string; If we apply t. "Eval" operator to it, we get 7.

We can also say, that t. function "Eval" ~~maps~~ takes t. string 3, 4, + into 7.

---

7.19.81 One approach: After TM has learned 3, 4, + (−, ×, ÷), Eval so he now has a pgm that will do this → 

| |
|---|
| −1 |
| 7 |
| 12 |
| $\frac{3}{4}$ |

i.e. simple unary & binary funcs.

we give him 3, 4, +, 8, x Eval. T. old pgm doesn't work any more. 2) He trys to minimally modify it to get a pgm that will work.

I would like TM to try to use t. old pgm as a subr., as a "minimal modifn."

Some other suggested approaches for TM!

.12 (b) Try to find out how t. new problem differs from t. old (in which t. pgm did work) — so we can still use t. old method on old probs. & realize that a special new pgm is needed for t. new kind of problem.

.13 (c) Try to break up t. new problem into parts that are workable by known (or more known) means.

---

Re: (b) (.12): If t. problems are sequential, then Time can be used to distinguish betw. old & new types of probs. (t. specific time when t. new prob types started would be a threshold). If there is no ordering info, this can't be done & TM must find an ob. to distinguish betw. old & new type probs.

One ob that will usually work! T. old probs have fewer symbols!

3(−) has 2 symbols, 3, 4, + has 3 symbols. 4, 3, +, 8 − has 5 symbols (≡ new type): hvr. 3(−)(−) is only 3 symbols & is still a "new style" problem.

(− is a unary operator meaning negation.)

→ A Better ob: t. Old type has only one non-numerical symbol. This always works.

Phone Sonn
Phone
Councilmen

3(−)(−)

---

One reason why it would be good in t. present problem for TM to recognize probs it could solve correctly: This (makes / may make) it possl. to do heur. .13 of breaking up a large prob. into sub-probs. Hvr., even w. that recognition, its not clear just how TM would consider t. relevant substitution

---

Another possl. soln. method! Use of t. Plan of (73.20) of xfmng t. problem into a new problem that mite be solvable.

This involves knowing about "equality", & that one can substitute sub expressns into an expressn. & retain equality.

Another way to train a human is by telling him how to work problems. One can write a trg. seq. of this sort — & at each point, write down just what t. human knows & what problems he can solve at that pt.

T. problems solvable will include those involving Genzu. via. Lsrch, of technique "told" to him by trainer.

One of main problems for me, is figuring out just How TM is able to use "info" obtaind by solving probs. in t. past, to solve probs in t. future: e.g.

.17 after learning $n_1, n_2, \left(\frac{+}{*}\right)$, how can it use these abss. to solve more difflt. probs?

one possl. approach for me: write out t. soln. of t. difflt. prob. — see just how parts of t. soln. of t. earlier prob. can be usd in it (as subs or whatever), then devise a formal means to → legitimize t. transfer.

After learning this & having been told about subsn. (as a Primitive op.),

It still would have an enormous search to learn "Eval":

The needed learning is somehow tied up w. TM. being able to recognize "parts" of a string that it could evaluate:

For Using Causal Knowledge.

Another view of this last: in t. operator $\{\alpha, \beta, \gamma, \text{sbs.}\}$; TM has to find $\beta$ in $\gamma$ that $\alpha$ should be subs. for. How to get this (or these) from t. $\left\{n_1, n_2, \left(\frac{+}{*}\right)\right\}$ that TM "knows" it can solve, is not at all clear. If TM can't do it, then its clear that he is lacking an impt. abs (or several impt. abss) that mite be needed in other, future probs. as well.

3, ft, + → 7

is

One (perhaps) useful concept is t. "Domain & Range" of an op.

doesn't seem to be usd in t. present context.

D → R

.30 For t. op of .17, its Domain is $n_1, n_2, \left(\frac{*}{+}\right)$. I.e. t. Cart. product of these 3 sets. This gives a set of strings.

In general, I think that to implement t. sbs op., we need to obtain $\beta$ as a substring of $\gamma$ in some way: consider t. set of strings in .30: somehow "and" it w. $\gamma$, to obtain a set of members of .30 that are also substrings of $\gamma$. Then $\alpha$ is a function (eval) of $\beta$.

T. forgg. process (t. set defined in .30; the "and" of it w. $\gamma$, then $\alpha$ as a funct (eval) of $\beta$) seems like a useful process: In future problems, we will want to know what objects a function can operate on..

{ We will want to occasionally use this set to find out how to xfm an object so that ~~this Range set~~ it gets into t. Domain set, so t. Function can operate on it.

I suspect that t. cpc (conditional pc? (probably)) of t. desired operation is still very low! ← Later: I'm not so sure

O.K.: Say T.M. has worked w. t. operator of 105.17 for a while & it knows it as a function, & it knows that it does not work for certain things. [θ maps things like 3.1, 2.2, + into 5,3] Call this operator, of 105.17, θ. D(θ) is t. domain set of θ. $D(\theta)\tilde{\cup}\gamma$ (Modified "And") is t. subset of D(θ) that are substrings of γ ......

∪ means "OR"! (01082)

Def Def .09

.11 We then try t. $\mathrm{Eval}\,(D(\theta)\tilde{\cup}\gamma)$ ③, $(D(\theta)\tilde{\cup}\gamma)$ ②, γ ①, sbs

Say S(γ) is t. set of all substrings of string S. Then $D(\theta) \cup S(\gamma)$ is what we want.

This expressn may not be so low in pc. We start w. ①; (should be $\tilde{\cap}$ (And)) ② is t. result of an operation on ①, ③ is a result of an operation on ②.

While t. pc may not be so low, t. cc for .11 ↗ may be somewhat hy! So I'm not sure about whether $\frac{cc}{pc}$ = Lcost is acceptable.

for this sort t. $\theta(D(\theta)\tilde{\cap}\gamma)$ would be more likely (0.1082) — But it wasn't what I was thinking about. See 82 TS 160.30

One reason that I want solns. of acceptable Lcost! (That if this is true), then it's more likely that t. set of abss. used in that soln. may be adequate for future probs (i.e. will yield an acceptable Lcost for future probs.).

.25 7.21.81 Going back to .11 ff: While D(θ) & S(γ) are nice mathical notations, implementing directly $D(\theta) \cup S(\gamma)$ is not a v.g. way to do it. Also, we don't want t. whole set anyway — any one member (say t. first one found) would be adequate — Hvr., in t. present problem, there is no overlap of domain strings in γ, so if we got a set this way, we mite want to do t. implied substitution in .11.

Also, I think this idea of $D(\theta) \cup S(\gamma)$ is useful in other problems as well, — tho again, I think there may be a better way to implement it than the way implied by that expressn.

.32 To do this properly, I'd want to see just how TM (or a person) would discover a v.g. (low cc) way to implement $D(\theta) \cup S(\gamma)$. This is just as important a part of TM's education as learning various hy pc. abss — It's t. learning of a heuristic device.

→ Note: That S(γ) & $D(\theta) \cup S(\gamma)$ & $\mathrm{Eval}\,(D(\theta) \cup S(\gamma))$ (AND not OR) are all multiple valued need not cause any diffy. Conceptually, t. soln. of .11 is O.K. — which is what determines t. pc. ← How we implement it is a separate Q that determines cc ...... Th. problem of .32 ↗

(I'm not sure pc & cc can be completely separated this way! 01082)

Hoard
Shop
eggs
car.
Brub book
(should color) Tie
Post
Glue.

Phone dear Gold Cos.
Crackers
warm Refrig.
Beer in frig.
~~Pastry~~

7.21.81 TS

N.B. Apparently I got $\cap$ & $\cup$ mixed up! .... I used $\cap$ for "OR" .... $\cup$ for "AND"



What should be done: Write out this particular soln. & try to genz. t. heurs. used, as much as possl. If I want these to be "primitive" heurs, they should be as genl. as possl.

O.K.: First, T.M. learns to solve [redacted] $n_1(F^1)$ ← unary funct

then " " " " $n_1, n_2 (F^2)$ ← binary funct

[redacted] T.M. at this point (w. no negative cases) might be able to factor t. domain into $n_1 \otimes F^1$ ($\cap$ should be "or") $n_1 \otimes n_2 \otimes F^2$ (⊗ ← cartesian product), but I'm not at all sure that this is a particularly hy pc. [redacted] dern. of t. domain.

Or it may need 1 or more negative cases.

Well: $n_1\ F_1$ $n_1\ n_2\ F^2$

T. first char. is always a number. T. 2nd char. is a no. or a unary funct name ($F^1$, hvr.) ← only a few possys.
If its a funct name, its t. last symbol in t. string. If its a number, then t. third char. is a binary funct. name.

Numbers have 2 parts: 1) Type symbol: that says its a number.
2) T. number itself. — This can be in various notations: Fixed ~~acc~~ accuracy integer; Variable accuracy integer; Fractions, Floating pt. of fixd or arby accuracy, etc.

Functions [redacted] names can be single symbols or they can have

.25 ① a part that tells how many args, [redacted] a part that has ② a name for t. function.

16 − .1 × 16 · α = 1.6 α
19 − 1
− 19 × .25 α

so $n_1, F_1$ looks like: n | number | F | 1 ← 1 = t. no. of args F has.

$n_1\ n_2\ F_2$ " " n | number | n | number | F | 2 ← 2 = t. no. of args. F has.

| a | b | c | d |   |   |
|---|---|---|---|---|---|
| e | f | g | h | i | j |
| $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ |

16 − .1α
− 19 + 1
+ 19 × .25 α

b, f & h [redacted] are unpredictable & random.

a. e are always n. — (I guess this is "Type" symbol)

(17.5 → 19) × .7 = 13.3

If $c_3$ is a function name, $c_4$ is 1

.31 " " " n, $c_4$ is a random no., $c_5$ is a funct name & $c_6$ is 2.

− 2 1.6 α
+ 4.75 α
= 2.65 for all!
− 2 + 3.15 α

T2281 I think t. forgoing (.25 − .31) is reasonable induction to expect from a Tabula Rasa machine! I.e. the method of doing t. correlations observd are something that we would want to "Wire into" our initial machine.

so T.M. Does get a good [redacted] model for to ~~...~~ Domain of $\theta$ (106.09)

At this point, T.M. has t. operator D; & D($\theta$) is a natural thing to do.

I'm not sure S($\gamma$) is so useful (106.11 R), but D($\theta$) $\cup$ S($\gamma$) is a useful concept.

We want a good cc for D($\theta$) $\cup$ S($\gamma$) — but we need not have a good cc for creating S($\gamma$) or even D($\theta$).

7.22.81 TS 0.10.82 108

TM has a good ($pc$?) for $D(\theta)$ —
$S(\gamma)$ and $S(\gamma) \cup D(\theta)$

~~The~~ also sbs: Now sbs has 3 ~~ar~~ string args. t. arg $\gamma$, is an arby string. $\beta$ is a substring of $\gamma$, and $\alpha$ is (usually) some function of $\beta$.

.06 Viewd in This way, if we have any [redacted] function on strings, $\theta'$, mapping strings into strings, Then sbs, and $\theta'$ do result (sometimes) in an operation on any string $\gamma$ — i.e. $\theta'(D(\theta') \cup S(\gamma))$, $D(\theta') \cup S(\gamma)$, $\gamma$, sbs

This will not work ~~well~~ well if t. set $D(\theta') \cup S(\gamma)$ are overlapping at all.

In t. present case, there is no overlap, so its ok. — But it may

.12 be nec'y to Backtrack (94.01) a bit in t. future, when there is overlap.

Whenever sbs is considered [redacted] operating on $\gamma$ (its last arg), we must find some way to define a ~~set~~ subset of t. substrings of $\gamma$. (later, we will want a way to specify 1 substring of $\gamma$). Then t. substitution is some function of that substring (or those substrings).

So one way to work sbs $\gamma$: Get substrings of $\gamma$, get funct on t. substrings.

~~another~~ This involves 2 things; ① $\gamma$, ② t. function

If t. function has all substrings as its Domain, There is no problem.

[redacted] Hvr, Given "sbs", $\gamma$ & some function that does not have all substrings as its domain, — we use .06 → .12 to implement $\alpha, \beta, \gamma$, sbs

.25

.26 T. resultant conclusion of t. forgo. dizcn. is that probably t. subsn. operator of 106.11 would have a reasonable pc, so 106.11 would probly be an acceptable soln. to t. "Eval" problem for any 2 level functions.

N.B. It is not a soln to a 2 level funct. $Eval_1(Eval_1(\gamma))$ is needed for 2 level functs.

Def ~~[illegible]~~ Say t. operator of 106.11 ~~is~~ $\equiv Eval_1^*(\gamma)$.

~~and~~ We give TM several probs in which $Eval_1^*$ works o.k. (i.e. 2 level functs)

Actually, $Eval_1(\gamma)$ works for 1 level functs, $Eval_1(Eval_1(\gamma))$ works for 2 level functs, call This $Eval_2(\gamma)$

Then, for 3 levels, $Eval_1(Eval_2(\gamma))$ will ~~work~~ be a soln. — so call This $Eval_3$. {Also $Eval_3^* = Eval_1(Eval_1(Eval_1(\gamma)))$}

After working probs w. successively hyer level functs, Eval 2, 3, 4 ... are defined, & TM should get t. idea that we "apply $Eval_1$ again & again until we get a ~~[illegible]~~ no." more

7.23.81 TS

The Mechanism from 108.26 to .40 is mainly in English. T. Q. is, is it a "real soln." i.e. can th. "handwaving" parts be made rigorous?

Well, first note that applying $Eval_1$ to 2 level probs does not solve them.
= $Eval_1(Eval_1(\gamma))$ in this case.

.04 $\theta(Eval_1(\gamma))$ will solve those 2 level probs, hvr.

.05 Hvr, $Eval_1(\gamma)$ does xfm t. problem into a prob. known to be solvable. (73.20 has refs to this "Plan"). This is a good way to solve this problem, but it does involve t. developement of this "Plan" in some detail. Also, TM must know that subsns of this sort always xfm an "Evaln" problem into an equivalent problem. This may involve TM's knowing t. "properties of equality".

.10

If we don't use .05 – .10, then I think we need $\theta(Eval_1(\gamma))$ as a soln. to 2 level functions. The pc of this is somewhat low. t. argt from about 106.01 to 108.25 is to show that $Eval_1(\gamma)$ has reasonable pc. $\theta(Eval_1(\gamma))$, then shouldn't be very low in pc, but it may be low enuf so that it is probly done by "subconcious mind".

Friday:

~~Phone for~~

① Get NY Times, WSJ! Phone various mut. funds that are promising! Get list of current investments.

Phone Various commodity brokers;

② Look up ph. nos. in old W.S.J.'s & NYT's.

Tell them about depository receipt.

③ Perhaps do ① anyway; I may have to delay the "Cap. Gains on Ag" trick & I may want a place to keep money anyway – since I can't get ~~...~~ contracts for exactly* look.

7.24.81 ⟨Looks at t. forest rather than t. trees⟩: 2 things I

.21 want to do: ① Draw up a T.S. in English, then expand out t. various parts into more & more detail, until it is adequate for a TM. ② Examine t. workings of TM using a reasonable part of a TS. See just how cpc's are assigned.

Presumably, I would do ① first, then ② would naturally follow.

Hvr, it would be possl. to devise a "soln" to ① that wasn't really adequate for a TM — i.e. too much apri info stuck into it a/o t. cost of solns. of some parts of it are far too large. From such a "soln" I could still get a good look at what ② was like.

Mainly, in ②, I want to see just how TM builds up complex concepts from simpler ones — just how cpc's can be assigned so this will work out ok. — e.g. is Z141 adequate? Do I have to go to much more complex langs?

.01 WRT 109.21 (1): So far, t. English TS is like this!

.02 (1) TM starts w. a set of unary & binary operations in its machine code, & so it can easily learn the notation for these ops from usually only 1 example using "random" nos. <Actually, if random nos. take too much time (~~too many~~ bits needed), just use more examples [redacted] using nos. having fewer signift. figs. — t. total amt. of info input may be t. same — as will perhaps be TM's search time for t. soln>.

.16 (2) Also TM has t. concept of "substitution": furthermore, it knows a lot about substitn, so that the ideas of ($\tilde{}$105.30 — $\tilde{}$108.25) are of hy pc — so t. operator $Eval_1(\delta)$ is of hy pc.

---

At this point there are 2 ways to get a soln. for t. more goal. "Eval" function: (3) & (4)

.20 (3) 1) use of t. plan of 109.05 — .10. (see e.g. 73.20) of xfmg. a problem into a class prob. of known soln.

.22 (4) 2) Use of t. GPS heuristic: [see 79.26 — .30 for discn of GPS] Here, whenever we make a

.23 $Eval_1(\delta)$ [redacted] subsn., we make t. resultant string smaller

.24 than t original & i.e. "closer to t. goal". Ordinarily, GPS is rather diflt for TM to use, because t. "set of differences" has to be devised & this can be a very complex task. In t. present case, hvr, it is fairly natural to use the "differences" of .23 — .24.

.27 Note: This may be involve a suffly simple modifn. of GPS so that 'differences' are easier

i.e. it may involve a simpler ("scalar") "hill climbing" rather than t. vector hill climbing of GPS. Also: for problems of higher depth, t. solns. are t. those of lower depth — so it's a special kind of a u.g. method. It may be worth while defining it as a special (heuristic) (rather than a subclass of GPS) — to be used in "simplifier" & many other prob. types.

After TM has been gn. [redacted] Eval probs of hyer & hyer levels, &

.29 (5) it solves them all, it should "get t. general idea" & be able to solve "Eval" probs of arby. level. I haven't worked out t. details of this, hvr. I expect that [redacted] the technique needed is of very general applicn., & so I may want to make it primitive — but if primitive, try to make it a generally applicable as possl. The idea here seems to be t. idea of Recursion. By looking at t. codes for t. solns. of more & more complex problems, [redacted] [redacted] i.e. $Eval_1(\delta)$; $Eval_1(Eval_1(\delta))$; $Eval_1(Eval_1(Eval_1(\delta)))$, etc.

.36 it should be able to extrapolate easily.

---

In addition to t. subproblems desc b. in .01 — .36 I also worked on t. prob. of discovering substitution by examples. See 60.24 for bibliogy op to that pt.

Consider t. system of 110.01–.40! I want to desrb. each of t. parts in suff't. detail so that its possl. to tell if they fit together.

Items: ① (110.02): Learning Unary & Binary Functs.

② (110.16): T. concept of substitution w. assoc. ideas of $\approx$105.03–108.25 on how to use subsn.

③ (110.20) xfmg. a new problem into an old prob. of known soln.

④ (110.22) GPS for ~~a~~ simple "difference" or "improvement" function — A simple Hill Climbing Heuristic.

⑤ (110.29) → Ability to do "recursive Ganza" to "get t. goal idea" in this particular case.

---

O.K., lets Do each of t. parts in greater detail:

① When presented w. inputs of t. form $\overbrace{\text{Eval}; \underset{n_1}{4}, \underset{n_2}{3}, \underset{F_2^*}{+}}^{I_i} \overbrace{\rightarrow 8}^{O_i}$

~~TM is able to solve~~ This for $F_2$ (binary) f. (or ary) funct.

TM is able to search for & find a pgm. to solve these probs. T. pgm. found gives conditional pc's in t. presence of "**Eval**" & other symbols.

Also, TM is able to characterize t. ~~set~~ set of input strings (~~..~~107.01–.31).

This characterizn. is, I think, often useful — so TM does it. It amounts to compressed coding (which is always good), & it gives TM more useful experience from t. corpus (in t. world of probs) than if ~~he~~ he just tried to solve t. **(I,0) problems only**. i.e. TM is able to use abss. from t. ~~...~~ min. codes of **this** "input" corpus, to ~~a~~ help solve I,0 problems.

→ Also, by characterizing this input corpus TM is able to get t. domain of t. operator, $\theta$ ~~...~~ usd in ~105.03–108.25, in "substitution".

→ So, by characterizing input, TM is sucking out every last bit of info in its corps. ←

So now TM has a soln. to this problem (t. operator $\theta$ (106.09)) & t. domain of $\theta$ (107.25 (also 107.01 → .31)).

---

② **T.** concept of substitution: T. function, **sbs,** from strings to strings: $\boxed{\alpha, \beta, \gamma, \text{sbs.}}$ (i.e. $\alpha \leftarrow \beta$ in $\gamma$)

The concept is a bit ~~...~~ ambiguous: there are several possl. meanings! (102.22ff)

Hvr. note remark of 102.36.

Another way of looking at t. ideas of $\approx$105.03–108.25! we have t. string $\gamma$ & t. operator sbs. In ~~GPS~~ their disn. of GPS etc., N. & S. consider t. sub problem of "~~How can we~~ $\boxed{\text{apply}}$ sbs to $\gamma$?". In general they often get into ~~...~~: "How can I ~~..~~ xfm $\gamma$ so that operator $\theta$ (say) can be applyd to it?".

Hvr. "Appy xfmn $\theta$ to $\gamma$" is in GPS, one of t. "output suggestions" resulting from a certain "difrnce" being observd, → (114.1)

7 2681 TS



91.06 } on Subconscious Mind
81: 91.24 } On Negative Reenforcement:

Why it is often N.G. for Humans & Animals:

Humans usually categorize large neg reenft. as assoc. w. danger: A life-threatening situation. As such, it is to be avoided w. hy ° of certainty. This means that t. situation ~~==~~ in which it occured is to be avoided. This "situation" will be in a very general sense, because we want to be sure we include t. relevant situation. The penalty for overgenzn. is small, compared to t. penalty for undergenzn!

Say we give t. human an intellectual H.C. problem, & we give him a large penalty for a $-\Delta G$ trial. He may then genz. This so as to avoid H.C. problems or avoid problems that look in any way like t. one being worked.

An example is a Malpractice suit for Md's. We'd like t. Md. to subsequently / to simply be more careful in t. relevant area. Instead he ~~=~~ (from our pt. of view) overgenzs. & avoids t. entire area of Medicine in which t. Malpractice Suit occured.

Note that ~~a~~ a not very perceptive external observer would regard t. reaction to neg reenft. as over genzn — in fact, it is overgenzn. from t. trainer's pt. of view, but is just t. right amt. of genzn. from t. pt. of view of t. payoff function of the Reenforcee.

Example of reaction to neg reenf. Trainer teaching dog to jump over fence on command "Jump fence". Dog tries & does wrong thing. Trainer whips dog. Henceforth, Dog avoids ~~that~~ that trainer. In presence of trainer, dog cowers in corner, trying to avoid trainer. Does not try to perform at all upon command by that trainer

1) why neg. reenft is ~~often~~ usually bad for learning.

2) that t. H.C. (TSOPS) method is better since it makes soln. for very hy level goals of = ease

3) Write actual paper on this example of a TS.

It learns to avoid danger.

— A situation in which overgenzn. is appropriate — yet it is over genzn. for positive learning.

Can we get t. M to learn write by both + & − reenft?

Genl. Admin:

It would be well to write an actual Technical Report a/o "Paper" on t. TS. of 110.01 → .40. This would be partly as a Bookmark for me, to state clearly what t. problem is, give part of a soln., ā tell why t. various parts were done that way, tell how t. TS ā soln. methods can be expanded.

Possibly Get Marvin a/o Levin as co-authors. Marvin, because (a) he's a very good writer (b) to get him to understand this problem ā get him to help work on it. , Levin, for (b)

Politically, Such a paper by these 3 authors would have much effect on t. A.I. community.

Also Marv. would be v.g. on Tng Seq. writing: it does involve t. ideas of "heur pgmg" — how we "really" solve probs, etc.

What I mite do is write a rough report, then get M. ā L. to help fill it out. This need not be a "paper" in a journal. It could be an MIT report or, if long enuf ā complete enuf, a Book.

Maybe get Peter Gaks to help?

---

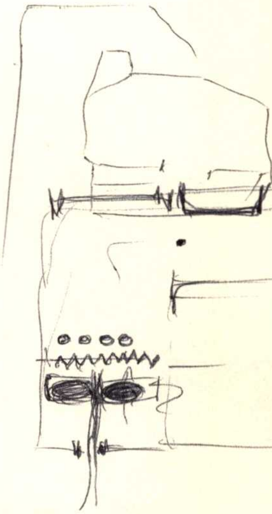
One of t. impt. ideas of the paper is Methodology: "Top down" Tng Seq. writing: To make an "English" desn of t. TS. first, then expand each section of t. TS desn. ā expand each part of t. expansion, etc., until we get what seem to be reasonable "primatives". This sounds very much like writing a computer pgm.

7.26.81 TS



On t. idea of having pc's (a cpc's) for **Obs**! Previously, I had that of making various obs be mandatory at certain pts in t. pgm (i.e. cpc = 1). Perhaps not necly. At each pt in t. pgm., one has a cpc of making one of several obs or simply one of several ops (($\Sigma$ pc of obs) + ($\Sigma$ pc of ops) = 1)

In t. absence of any recent obss TM works w. t. ob. outputs of t. past/ — mainly, t. most recent ones, — since These are t. best presently available.



.11 : 111.40 → **I** think I mite be able to get TM to t. point where it will try t. proper substitution — but for it to realize that ~~it is~~ t. result is something useful, is another problem.

For a 2 level problem, doing a subsn. xfms it into a 1 level (usually — but sometimes a 2 level) problem.

If it has become a 1 level problem, it is likely that, since TM knows it can solve 1 level probs, (— or more exactly, that $\Theta$ can be applied to 1 level problems), it's likely that TM will try $\Theta$ on to 1 level prob. & solve it — i.e. it will get an

.21 answer That $\equiv$ t. "$O_i$" value.

On t. other hand, using to heurs (2)(110:20) & (3)(110.22) presuppose a difrnt. problem type! In .11 - .21, t. soln. is for t. situation in which TM is gn. (I, O) pairs & must find a genl. xfmn. betw. them of hy pc. & low cc. Here we have to find a seq. of xfmns <out of a certain set of xfmns> that xfm $\delta$ into a "number".

— So strictly speaking (2) & (3) are not directly relevant in t. present problem.

— they would be if t. problem were formulated difrntly, hvr.

If neither heurs (2) or (3) are used,

.30 $\Theta(\text{Eval}_1(\delta))$ is an acceptable soln for 2 level probs —

.31 as is ~~Eval, ($\delta$)~~ $\text{Eval}_1(\text{Eval}_1(\Theta))$.

Note: Superficially, TM mite be that to know that he has to get a single number as t. result of his xfmns on $\delta$ — (in which case, t. H.C. sub-method of (3) could be usd) — hvr. strictly speaking, this is not so — TM has to obtain t. desired output string as if he didn't know what it was.

— An apparent exception to this is linear regression coding, or many kinds of predictive coding.

Another possy. is that TM could induce t. Range of "Eval" & decide

1AM
9AM 8
2PM 5
8PM 6
1AM. 5

30 mw
900
100k
= 4 MN

81
10n

$5^2$:
$\frac{25}{4}$ = $6\frac{1}{4}$

that it is always a single no. Viewed in this way, t. H.C. heuristic of 110.27 is usable. The heur of 110.20 ("Xfmny problem into a solved problem") is not usable, hvr. — it mite be w. suitable backgnd — exp. if TM had a better understanding of what needed to be done, & understood some properties of "equality", etc.

So — using 114.30–.31 as solns. to 2 level probs would enable TM to solve Eval probs of successively higher levels.

Not so obvious! If TM is just looking for xfmns to reduce $\delta$ to a single number, it can easily do this by selecting out any of t. single nos. in $\delta$. ! — which would clearly not be a soln. Anyway, this → would work & conceivably t. heurs of 110.20 & 110.27 mite be made to work.

In noting that $\Theta(\text{Eval}_1(\delta))$, $\text{Eval}(\text{Eval}_1(\delta))$ etc. are solns. of Eval probs of various depth — Just how does T.M. "get t. genl. idea?"; One way would be to notice that Eval, was to be applied repeatedly, until t. result was a single no. Actually, I think a single no. is not in t. range of $\Theta$ & .. not in t. Range of Eval, — Tho it could be, if I gave examples like Eval: 13.731 : 13.731
I, Oi

Anyway, say I have an Ob that can determine if $\delta$ is a single no. I could use such an ob. to control t. invocation of Eval, or as a "stop rule".

If $\text{Eval}_1(3) = 3$ say then t. solns. of t. Eval probs can be of t. forms $\text{Eval}_1^{(n)}$ $(n = 1, 2, 3, \ldots)$. $\text{Eval}_1^{(n)}$ w. n = largest n needed thus far would be a hy pc soln., tho not necely best cc. This would be a bad soln., because it wouldn't work for probs of greater no. of levels.

Hvr., If we look at t. set of solns. $\text{Eval}_1^{(1,2,3,4,5,\ldots)}$ & we consider these solns. as members of a simple lang (say a FSL), then it would seem that $\text{Eval}_1^{(\infty)}$ would be a soln. — tho again a soln of excessive cc.

On t. other hand, T.M. should keep an eye out for heuristics: Clearly if Eval, is applied to a string again & again & no change occurs, we should stop.

Look at t. Lsearch for t. soln! trials like $Eval_1^{(n)}$ will have pc's of ≈ $k^n$, where k is close to 1. So if TM finds $Eval_1^{(10)}$, say, doesn't work, he will quickly try $Eval_1^{(11\ or\ 12\ or \dots)}$, since they are of very hy pc.

Now in "**Meditation**" mode, TM may look at his acceptable solns. to various of t. problems & find that he could have saved much cc by stopping when t. xform. string yeilded a single no. **I don't know** just how he'd go about doing this, hvr.

→ Hvr, if $Eval_1^{(10)}$ has always worked o.k. thus far, TM will use $Eval_1^{(10)}$ on γ, even tho γ has 12 levels of functs in it! — So T.M. will not extrapolate well if it does things this way.

7.27.81 ← next day, ~9PM

T.Q is: **IS** it reasonable for TM to induce a "stop rule" by "noticing" that t. correct answer was obtaind when $Eval_1^{(n)}$ yeilds a single no.?

Perhaps TM should be looking for "stop rules". One way to do this: TM looks at all of t. successful solns. — Then varies them, inserting lots of obs near the end of each run — so as to be able to tell when it would end. For TM to be able to "notice" that t. single numerical output signals "stop", this particular **ob.** has to be of fair pc.

**TM** mite notice that after a single no. is obtaind, doing $Eval_1$ again does not change it — so there is no pt. in continuing. This involves, as befor (.20), fair pc for this ob.

**If** $Eval_1^{(10)}$ was t. biggest size op needed to solve all probs this far, then $Eval_1^{(10)}$ could be used as t. single op for all cases. Hvr., an $Eval_1^{(n)}$ w. a suitable stop rule can have more pc. (for hy values of n) & can certainly have less cc, for t. entire corpus so $\frac{cc}{pc}$ [redacted] ≡ Lcost could be much better than $Eval_1^{(n)}$ w. fixd n — so it would be found sooner (perhaps) in t. Lsrch.

}  Hvr, see 117.19 ff for a quantitative analysis.

**Methodological Note**: If a human can be expected to notice ~~stop rule & use~~ t. termination of t. operator when t. result was a single no., & devise ~~a suitab~~ t. corresp'g. stop rule — leads to ↑ pc & ↓ cc, then perhaps we should expect **TM** to be able to do this also.

To implement this, we then have [redacted] being able to notice this "single no. situation" as a "sub-goal" for either a t.s. or to be inserted into TM as a "primative".

7278 | TS



The T.S. as of now!

.02 1) First learn t. operator $\theta$ ! Unary, binary & t. identity function [redacted] ($no_i \to no_i$ (Eval $no_i \to no_i$; Eval $3.781 \to 3.781$))
Also learn $\theta$'s Domain.

.03 2) Learn $\theta(\mathrm{Eval}_1(\gamma))$ & $\mathrm{Eval}_1^{(n)}$ & $\theta(\mathrm{Eval}^{(n)}(\gamma))$ as soln. to problems.

— This involves giving T.M. "sbs" as a primitive, w. assoc. ideas That make it a useful concept. e.g. [redacted] $\theta, \gamma$ → A set of [redacted] substrings of $\gamma$ That are in t. domain of $\theta$.
op ↑ string

.10 3) The use of t. ob of 116.20 as a "stop rule" for [redacted] $(\mathrm{Eval}^{(n)}(\gamma))$

1) Phone Co: — ask about Touch tone "I got this T.T. phone for Gift".

**72981** (SN) On "sbs": Usual use: find a substring/of $\alpha$ $\gamma$ having folg. properties (or That is in t. folg [redacted] set of strings), & substitute something for $\alpha$ That is a function of [redacted] $\alpha$.

on .10 This stop rule saves ~~some~~ cc (usually < a factor of 2, hvr [redacted] — but depends on t. corpus) & it may or may not ↑ pc .... but it will certainly ↑ pc if t. [redacted] problems have enuf levels in them. Also t. ratio of cc ↓ will [redacted] be larger if we have some probs. w. very many levels in them.

.19 The cc saved using a stop rule:

1) using stop rule, cc of doing t. corpus: Say $n_i$ is t. no. of subsns. necy. in t. $n_i^{th}$ example. Say there are $m$ examples &

$\boxed{N \equiv \mathrm{Max}_i(n_i)}$ ($\equiv$ t. largest $n_i$ overall;). Say $A$ is t. cc of doing 1 subsn. & $B$ is t. cc of examining each resultant string to see if it's a single no. or not.

(Usually $B \ll A$ since it takes little cc to realize t. string is not a single no.).

So $\approx \boxed{(A+B)\sum_{i=1}^{m} n_i}$ = cc of doing all of t. corpus, using t. stop rule,
of $B$
(assuming $B$ is same whether result is "yes" or "no".)

---

If no stop rule is used, cc of entire corpus is $\boxed{A \cdot m \cdot N}$

So ratio $= \frac{A \cdot m \cdot N}{(A+B)\sum_{i=1}^{m} n_i} \approx \frac{m \cdot N}{\sum_{i=1}^{m} n_i} = N \Big/ \left(\frac{1}{m}\sum_{i=1}^{m} n_i\right) \equiv \frac{N}{\bar{n}_i}$

So ratio of cc is ratio [redacted] $\frac{n_i\,(\max)}{\bar{n}_i}$, $\approx \frac{\text{cc. of no stop rule}}{\text{cc. of w. stop rule}}$.

more exactly, ratio = $\boxed{\frac{A}{A+B} \cdot \frac{n_i\,(\max)}{\bar{n}_i}}$.

---

Ratio of pc's: This caln. is less certain:

Say pc of $\mathrm{Eval}_1$ is $k$, then pc of $\mathrm{Eval}_1^{(N)} \approx k^N$ (i.e. no stop rule) ($\frac{k}{N+1}$ maybe better estimate! see 118.17–.32)

Dart

Say pc of the ob That limits recursion of $\mathrm{Eval}_1$ is $\ell$. — Then pc of t. operator using t. stop rule, is $k \cdot \ell$.

Lcost : no stop rule = $\frac{m\,A\,N}{k^N}$ $\left(\frac{m\,A\,N \cdot N+1}{k}\right)$ viz 118.32

Lcost using stop rule = [redacted] $\frac{(A+B)\sum n_i}{k \cdot \ell}$

.01 $\frac{\text{Lcost: no stop rule}}{\text{Lcost: stop rule}} = N\left(\frac{m}{\sum n_i}\right)\frac{A}{(A+B)}\frac{K\cdot l}{K^N} = \left(\frac{A}{A+B}\right)\left(\frac{n_i \max}{\overline{n_i}}\right)\cdot\left(\frac{l}{K^{N-1}}\right)$

is occasionally v. impt.

usually most impt factor.

An impt Q is: is k close to 1? If it is as small as $\frac{1}{2}$, then $K^{1-N}$ will be t. dominant factor for large N (say N=10). $\frac{n_i \max}{\overline{n_i}}$ may be large, but

.04 if, e.g., $n_i = i$, then $\overline{n_i} = \frac{m}{2}$, $n_i \max = m$ so $\frac{n_i \max}{\overline{n_i}} = 2$ only.

.05 K could be close to 1 in a sense: If Eval$_1^{(r)}$ has worked in t. past for large enuf r, & ↑ r has always been successful, if this did not work, then t. cpc of Eval$_1$

.07 could get close to 1.

The reasoning of .05 – .07 could give Eval$_1^{(N)}$ a hy pc. — On t. other hand, reasoning about stop rules should give hy pc. It may well be that t. reasoning of .05 – .07 is not so good & should, in general be avoided. On t other hand, searching for a stop rule for a repetitive operator is, in general a good idea for many problems.

1) calculate pc of Eval$_1^{(N)}$

2) Consider these "2 methods of coding" w. different predns. Or just look at them — maybe one is not "obviously better" than t. other.

.17 $\boxed{73081 \mid 8pm}$ Say t. initial apriori of Eval$_1$ is c, w. wt. = 1

$\frac{a}{\underbrace{a+b}_{wt.}} = c$; $a+b=1$, $\frac{a}{c}=1$ so $a=c$.

so for no cases of Eval$_1$, $pc = \frac{c}{c+(1-c)} = c$

.20 for 1 empirical case; $\frac{c+1}{1+1}$ (I guess $r \equiv N$)

" r " cases (no other cases) $\frac{c+r}{1+r}$

$c\cdot\prod_{i=1}^{r}\frac{c+i}{1+i}$ ; say c is small to start ($\approx 0$)

$c\cdot\prod_{i=1}^{r}\frac{i}{1+i} = \frac{1}{1+1}\cdot\frac{1+1}{1+2}\cdots\cdots\frac{1+r-1}{1+r} = \frac{c}{1+r}$

I could make a small correction for $c \ll 1$, using t. factor $\left(\frac{1+c}{1}\right)\left(\frac{2+c}{2}\right)\cdots\left(\frac{r+c}{r}\right)$

$= (1+\frac{c}{1})(1+\frac{c}{2})\cdots\cdots(1+\frac{c}{r}) \approx \exp \sum c\,(\sum_{i=1}^{r}\frac{1}{i}) \approx \exp(c\cdot(\ln r+\gamma))$

$\approx e^{(\ln r+\gamma)c} = \boxed{r^c\cdot e^{\gamma c}}$ ($\gamma \sim .6$; $e^{\gamma} \sim 1.78$ so $\approx (1.8r)^c$; $\gamma = .5772157$ Euler's const.)

Note $r \equiv N \equiv \max_i(n_i)$

so pc of Eval$_1^{(N)}$ $\approx \frac{c}{1+N}\cdot N^c\cdot e^{\gamma c}$ (126.30; Also 127.01 – .11)

$\approx \frac{c}{1+N}$ for $c \ll 1$. (Hvr. see 121.10 – .32 for a different result.)

".04": $\frac{N}{\overline{n_i}} = 2$ in that case

.32

maybe not so impt: see t. example of

Note: $k \equiv c$

Modifying .01 ↑: $\frac{\text{Lcost: no stop rule}}{\text{Lcost: stop rule}} \approx \left(\frac{A}{A+B}\right)\boxed{\frac{N}{\overline{n_i}}}\boxed{l\cdot(1+N)}$

**N.B.**

**Note:** 82TS 25.01 For what looks like a much better analysis of this,

so $\frac{N}{\overline{n_i}}$ & $l\cdot(1+N)$ are t. 2 factors that mite make t. ratio $\gg 1$.

Well, this isn't altogether unreasonable; If N is small (say 5) & l is small, then $\frac{N}{\overline{n_i}} \sim 1$ & $l\cdot(1+N)$ will probably be $< 1$, so t. no stop rule method will have lower Lcost than t. stop rule method.

.01 That $\boxed{\ell \cdot (1+N)}$ should be t. impt. factor, is reasonable. If $\ell$ (t. pc of t. stop rule) is small, Then N must be large before t. stop rule method becomes better Than t. "simpler" "no stop rule method"

T. "no stop rule" method is simpler & shorter if N is small & if $\ell$ is small. Hvr., no matter how small $\ell$ is, for large enuf N, t. stop rule defn. is better.

This may be true in general, for defns That use recursion (like t. "stop rule" type defn).

Note also, That t. $\ell(N+1)$ factor has to do w. pc's. Th. rest of t. factors mainly having to do w. cc, are not nearly as impt. in This case. As a result, we tend to get (in This case) t. operator of max pc — which i. gives better predns. (as well as being of somewhat lower cc).

.17

.18 Comparing t. "No stop rule" w. t. "stop rule" method: t. pc. of t. stop rule, $\ell$, can be rather large if TM has had suitable experience (≡ training). When TM sees a long repetitive seq., it should look for a stop rule. T. Big Q here (so far This part is hype.) is, what is t. pc. of t. 06. {t. result of This string lɲum is a single no}? or t. pc of {This string is a single no.}? This is, in genl., an impt. prob. Obs are certainly impt. & I need to know more about how They are revised & how to get Their pc's. → (120.01)

.23

Look over 117.02 — .10 to find other bottlenecks in t. TS.

On 117.03 This may need more work: As w. t. "stop rule" idea assoc. w. recursion, I need to figure out just how The ideas of "how to apply t. concept" are assoc. w. t. concept of "substitn."

Another, very basic Q; After TM has found that t. operator $\theta$ is an adequate soln. for t. first set of unary & binary op problems, I ~~start w.~~ continue w. 2 level probs. Just how does TM go about searching for a soln. — retaining t. concept $\theta$ which has been successful Thus far? I guess t. idea is That we have a seq. of problem solns: $\theta_1, \theta_2, \theta_3, \theta_4 \ldots \theta_n$ This sequence gives an apriori pd for t. $n+1^{th}$ trial. Clearly $\theta$ will have t. most pc. & will be tried first. When it fails, we continue to look at less likely trials. We could build into TM a stochastic PSG or PSG discoverer to extrapolate This sequence — or, There may be other, better, more "natural" methods to extrapolate. I think I expected to do t. extrapolation "In English", & from This, get some ideas of just what kind of extrapoln. system to use.

$10^{17}$

$\frac{503 \text{ days}}{30\frac{1}{2}}$

$\frac{12}{16} = \frac{3}{4}$ $16$ almost $17$ mo

$16\frac{1}{2}$ mo.

$\sum_{i=1}^{n} \frac{1}{i} \approx \ln n + \gamma$

say $n = 15$

$\Sigma = 3.318229$

$- \ln 15 = .610$

$\sum_{10} = 2.9289$

$- \ln 10 = .6263$

so say $\gamma \sim .6$

$e^{.6} = 1.8$

Actually $\gamma = .5772157\ldots$

$e^{\gamma} = 1.781$

try $25K contract (6 bars) w. Monex! Ask Them to bid & ask ½ way betw. B. & A. for both ends of contract.

I am willing to wait on both contracts.

Another disadvant of This Monex deal: 16 mo. means I'm locked in for $4\frac{1}{2}$ mo more than I'd like.

.01: 119.23 : On t. pc of t. relevant ob: That an ob is needed for t. stop rule & that a stop rule is desirable, are to be regarded as hyp c.
As for t. pc of t. ob itself: There aren't many reasonable pc obs. that successfully predict when ~~[illegible]~~ $Eval_1^{(n)}$ should stop.
.05 T. "correct" one may be t. most likely. Do we use its pc. or t. pc normalized
.06 over all other obs that are consi w. t. observed values of $n$ in $Eval_1^{(n)}$? In this last case, t. pc. will be quite hy! (i.e. close to 1)

My impressn of .05 – .06: That we should not normz. The argument:

Say $\xi$, $\mu$ & $\pi$ are 3 difrnt possl. obs. $\xi$ & $\mu$ are consi. w. t. "proper" value of $n$ & $\pi$ is not. $\xi$ gives proper extrapoln., $\mu$ does not.

So Say $\alpha\cdot\xi$, $\alpha\cdot\mu$ & $\alpha\cdot\pi$ are 3 codes for t. corpus. Their relative pc's are $pc_{\xi,\mu,\pi}$. $pc_\xi / pc_\mu$ is t. rel probty fn. to t. continuations implied by $\xi$ & $\mu$ resp.
Hvr., if $\beta_n$ is t. code using no stop rule, but $\beta_n \equiv Eval_1^{(n)}$, Then t. rel. probtys of t. continn implied by codes $\beta_n$, $\alpha\xi$ & $\alpha\mu$ are
$pc_{\beta_n} : pc_{\alpha\xi} : pc_{\alpha\cdot\mu}$

---

One reason That t. (correct) ob. of interest would have hy pc: In general, if repetition of $\phi$, t. op needing t. stop rule >, leaves t. string invariant, Then clearly, we want to stop.

---

.25 8281 3:20p A point of uncertainty: I had Tho't that after seeing a soln. like $Eval_1^{(10)}$, TM would decide That a loop w. a stop rule should have hy pc, Hvr, I'm not certain That This gives t. "loop" concept a hy pc in t. final code — since in t. final code, TM doesn't know That at t. pt. t. <loop + stop rule> are guessed at, $Eval_1$ will have to be repeated up to 10 times to obtain t. correct result.

$(1-\epsilon)^n\cdot\epsilon$

$\epsilon\sum_{n=0}^{\infty}(1-\epsilon)^n = \epsilon\cdot\frac{1}{1-(1-\epsilon)} = 1.$

apprip distribn. of integers.
$(1-\epsilon) \equiv$ probty of t. operator $n \leftarrow n+1$
$\epsilon \equiv$ aprip of t. operator "stop"

On t. other hand, in linear regression coding, TM is allowed to see t. entire corpus before devising t. code. (see 122.01 for discn of MaxM).
Hvr, note that Maxm. is a PEM (≡ CPM) & is a special kind of coding Method. Z141 & stoch cf Grammar. are also CPM coding methods. Z141, hvr, looks like a sequential coding method, also.

123.01

~~[illegible]~~

8.2.81 (sun) TS

Another point: My analysis of $Eval_1^{(10)}$, say, may have been wrong.

Say $Eval_1^{(1)}$, then $Eval^{(2)}$ were needed for successive sc's. $(Sc_1, Sc_2)$

for $Sc_3$, perhaps $Eval_1^{(1)}$ would be tried first because it has hyer pc & lower Least.

Maybe not! TM is looking for a single operator that has worked for all examples thus far.

.10 ~~[illegible]~~ Actually, t. pc of t. $Eval_1^{(12)}$, say, soln. is quite hy:

TM looks at t. seq. of previously successful solns: say

.12 $Eval_1 (1, 1, 1, 2, 2, 3, 4, 5, 5, 5, 6 \ldots\ldots, 12)$.

The next member of this seq. is most likely $Eval_1^{(12)}$ w. some possy of $Eval_1^{(13)}$, (or $Eval^{(14)}$ if we have ever jumped that much in t. hy seq).

For t. trial $Eval_1^{(n)}$, n is sharply distributed w. a big peak at 12 & a perhaps smaller one at 13 & much smaller at 14 —

.20 & some possy of operators $\notin Eval^{(n)}$

Under these circumstances, t. probty of TM occurring t. loop soln. is very low.

$\boxed{2\cdot26\cdot82}$ in .10 we are contrasting t. conditional pc of $Eval_1^{(n)}$ ~~[illegible]~~ (knowing $Eval_1^{(n-1)}$) w. t. pc (unconditional) of t. loop soln. T. conditional pc. is, indeed very close to 1. — which brings us to t. problem of why look for less al. solns (loop) rather than incremental solns ($Eval_1^{(n)}$) which are more al.

.23 $\boxed{8\,3\,81}$ woops! → t. argt. of .10 – .20 is not so certain! Say we had t. sequence of acceptable solns. of .12; Then t. pc of each soln. would be (t. pc. of t. previous soln. $\times \frac{1}{2}$ or $\times \frac{2}{3}$)

So for, say, ≈ 20 examples total pc would be betw $\frac{1}{2}^{20}$ & $\frac{2}{3}^{20}$

$\left(\frac{1}{2}\right)^{20} \approx 10^{-6}$; $\left(\frac{2}{3}\right)^{20} \approx \frac{1}{3000} = 3 \times 10^{-4}$

3. 456
1. 048

So, ~~[illegible]~~ t. exponential ↓ of pc w. no. of examples would ↓ pc much more ~~[illegible]~~ rapidly than the $\approx \frac{c}{N+1}$ effect ~~[illegible]~~

.32 ~~[illegible]~~ obtained in 118.32.

$\boxed{2\cdot26\cdot82}$: Hvr. t. idea is that we'd be multiplying not

$\frac{2}{3} \times \frac{2}{3} \times \frac{2}{3} \ldots$ but $\overset{\text{like}}{\approx} \frac{2}{3} \times \frac{3}{4} \times \frac{4}{5} \times \frac{5}{6} \ldots \frac{n-1}{n} = \frac{2}{n}$.

Hvr., see 126.30, 127.17 for impt. corrections

.01 A sort of simplifn. Say we have a corpus of n data pts & want to ~~extrapolate~~ get t. ~~[illegible]~~ apsipd for t. n+1th pt.

If we use m coifs, we use the first m data pts as known, & use all m coif linear predn. formulae / plus all values of $\sigma^2$ for predicting t. rest of t. data pts.

We can do this w ll for all values of ~~[illegible]~~ m from 0 to ∞ | (Actually, probly from m = 1 to n−1 maybe adequate). There is no ~~ε~~ r cost for t. coifs & for $\sigma^2$, since all values are tried. } ? (2009)

For a given m, the coding of t. first m data pts can be direct, to whatever ~~[illegible]~~ accuracy is desired. Presumably, t. same accuracy will be used for correcting to linearly predicted values via t. ~~[illegible]~~ gaussn. distribn.

As we ↑ m, we have ~~[illegible]~~ greater r cost for coding t. first m data pts, but we have ~~[illegible]~~ fewer predicted pts & so their r cost

.19 will ↓ by a ≈ similar amount.

Anyway, I think this would give some kind of soln. to t. Q of "how many coifs" to use — { Tho of course one should use

.215 → all nos. of coifs from 1 to n−1, suitably wtd. —

If t. soln. of .01 ~~[illegible]~~ −.19 is o.k., then t. "correct" soln. of .215 will be t. best.

.25 One Q that has always bothered me is: what is t. apripd. of numbers? A possl. "soln" ~~[illegible]~~ for integers.

Say one starts with ϕ given & one can generate n+1 by t. operator, ~~[illegible]~~ S, (≡ successor), so ~~[illegible]~~

$\phi = \phi$ (given)

$1 = $ ~~[illegible]~~ $S(\phi)$ ; $2 = S^{(2)}(\phi)$ ; $n = S^{(n)}(\phi)$.

If t. pc of S is $1-\epsilon$, then n has t. pc $= (1-\epsilon)^n \cdot \epsilon$ ($\epsilon$ is t. pc of "stop"). As we expect, t. Σ aprip of all integers is $\sum_{n=1}^{\infty} \epsilon(1-\epsilon)^n = 1$. A Q of course is: what value of ε to use?

~~[illegible]~~ The number ~~[illegible]~~ $\boxed{\frac{1}{\epsilon}}$ is t. ≈ width of t. distribn.

{ Or, if we let t. integer 1 have pc = ~~[illegible]~~ α, then n will have pc $\alpha(1-\epsilon)^n \epsilon$ & t. total pc. of all ~~[illegible]~~ integers become jus α, instead of 1.

6.5.83 ≈ Rissanen uses ~~[illegible]~~ bcost of n = log n + log log n + log log log n ⋯ ~~[illegible]~~ : The no. of ~~[illegible]~~ terms are as many as possl., yet ~~[illegible]~~ each term must be ≥ 0. t. proof is in Rissanen & may be perhaps easily derived from Cover & Leung-Chan's paper

on MexM. → 149.16R

$\boxed{8381}$ TS

.01: 123.09: **Is** t. folg. reasoning reasonable? **TM** notes t. successive solns of 121.12, & on this basis, decides that a soln. for t. next prob. will be $Eval^{(n)}$ using a stop rule. As data, TM has t. entire seq. of solns upto $Eval_1^{(12)}$. Say, Along w. t. concept of "stop rule for recursion", TM has t. assoc. idea that this is a reasonable thing to try (hy pc) if one has had a seq. of repetitions of an operation of various lengths. Hvr., using t. reasoning mode of 121.23–.32

If $p_0$ is t. pc of $Eval_1^{(12)}$, then if $p_1$ is t. pc of t. stop rule concept & t. correct Ob., then $p_0 \cdot p_1$ would be t. pc of t. final soln. Hvr., this much worse than t. $Eval_1^{(n)}$ method of 121.23–.32 [illegible] using fixed n — which gets ~ $p_0 = \frac{1}{2}$ or $p_0 = \frac{2}{3}$ for t. pc. of t. soln. — presumably $p_1 << \frac{1}{2}$ !

Well: think about it this way: Using fixd n, we are always a bit uncertain about what t. next soln. will be. Hvr. w. TM. ~~stop~~ loop & stop rule, we mite have a sort of AH HA! phenomenon in which we
.21 were rather certain that this new trick would end t. uncertainty in knowing just what t. next soln. was.

Hvr., I don't see any good rationall for t. feeling of .21: The loop method & t. fixd n method both worked exactly correctly for all cases up to now, but t. loop method seems to have much less pc. (tho slightly less (say a factor of 2) cc ) than t. fixd n method.

Actually, th. loop method may look better because it works & yet it gives (often) less cc & it ~~works~~ yields a much more varying n values, than t. "fixd n" method does. The analogy I have in mind is linear correlation! . The correlation betw. x & y is more "signift." if x varys very much & y still follows it.

---

➔ Approach: ① State problem clearly ② try to remove irrelevant parts & put it in as simple abstract, terms as possl.

DEF $\boxed{8481}$ 1030p: O.K. let $\beta$ be t. operator $Eval_1$.

we have been giving T.M problems to which ~~[illegible]~~ t. soln. to t. $i^{th}$ prob. was $\beta^{n}$.

.33 $i = 1, 2, 3, 4, 5 \ldots$ say n was a non ↓ funct of i.
.34 $n_i = 1, 1, 1, 2, 2, 3, 4, 5, 5, 5, 6, \ldots 12$

~~[illegible]~~ T. soln has to fit all probs up to now, so at first approxn it will take much extra time to test $\beta^{n_{max}}$.

[redacted] SN On t. pc of t. $\beta^n$ operators (for fixed n)! t. calcus. of (118.17–.32) may be wrong! ~~ ~~ Those of (12 ~~ ~~.23–.32) (121.23–.32) may ~~be~~ more correct.
The idea is This: In addition to t. essentially rite direction of ~, consider t. augt. at 118.17–.32: ~~After β has occured N times~~, . Actually, we are not trying to solve t. problem" If β has occured ~~ ~~ times in a row, what's t. likelyhood of its occuring again?" Instead, its t. problem: " t. $i$th soln. has been $\beta^{n_i}$ (see 124.33–.34 for table of $n_i$ v.s. $i$.) what is t. ~~ ~~ apsipd for $\beta^{n_{i+1}}$?" Say t. table of $n_i$ v.s. $i$ is ∋ ~~ ~~ $P_{roby}(n_{i+1} = n_i) = \frac{1}{2}$ ; $P_{roby}(n_{i+1} = n_i + 1) = \frac{1}{2}$.

.12 8581 SN Another "soln." to t. problem is $\beta^i$ or $\beta^{2i}$ or $\beta^{k*i}$ or $\beta^{k+2i}$ or $\beta^{2^2}$. $\beta^i$ or $\beta^{2^i}$ is quite simple & would work well, unless t. ~~ ~~ ~~ ~~ depth of t. problems ↑ very fast

Note! If TM just assumed (~ erroneously) that $n_{i+1} = n_i + 1$, ~~ ~~ every time then we get $n_i = k + i$, which is viz & quite ~~ ~~ adequate (usually) as a soln. — except for exessive cost. — Not very excessive, hvr.

If t. successful solns have $n = 1, 2, 3, 4 \ldots$ then $n = i$ is a reasonable extrapoln.

One way to look at it: we have this seq. of codes for t.
.28 soln. $\beta^{(n)}$ $n = 1, 2, 3 \ldots$,
TM ~~ ~~ looks at this seq of codes & tries to find a hyer level code for them. ~~ ~~ t. soln $n = i$ would then be a reasonable output from such a hyer level examination ("coding & Recoding").
In a similar way, perhaps, TM could look at these (.28) solns., & decide that a recursion & stop rule is likely to work — (with their usual ↓ in cc of solns, ≤ (is this ~~ ~~, indeed, true?)) — & so TM looks for a stop rule. Hvr., Just what is t. Gore for this hyer level search? If it is min cost, then "$n = i$" is best. ~~ ~~ If it is pc, "$n = i$" is best. Only for t. cc gore is t. ~~ ~~ lopp a stop rule best! — which is a ~~very~~ not very impt. Gore!
Hvr., t. derb. t. entire sequence of all of t. $\beta^n$'s, t. rule ~~then~~ $n = i$ would not be of such hy pc ~~ ~~ ... & t.

8581 **TS** 126

64 K at 1 mc = 64 ms. at 4 mc = 16 ms

loop rule mite, indeed, be comparable in pc a/o L cost.

.02 Hvr., I don't think that's t. point. T. idea is to get a single operator that works all probs up to now, that is of hy pc. Because of t. sequencial nature of t. TS., t. pc. of t. latest trial depends on what occured

29 k → 39 k

.05 in previous trials up to that point.

---

.06 Another possy is that t. idea of .02 — .05 is basicly wrong. That the particular model of an "operator TM" that I'm using there is a illegitimate mixture of unorderd set extrapoln. & sequencial series extrapoln. We could view t. trng. seq. as:

$I_1, O_1; I_2, O_2; \ldots\ldots I_n, O_n; I_{n+1}$ here , & ; are puncta. symbols & $I_i$ & $O_i$ are strings. We want t. proby distribn for $O_{n+1}$.

We can assume a kind of coding here: That all $I_i$ are coded directly, that we try to code t. $O_i$'s in terms of t. $I_i$.

N.B. (strictly speaking, This is not so — & we do look for regys in $I_i$ sequence — as when we try to characterize t. Domain of t. EVal, operator (as (07.25)))

Viewd in this way, t. operators $\beta'\Delta$, $\beta^2\Delta$, $\beta^3\Delta \ldots\ldots$ ($\Delta$ is t. "stop" symbol or "end of string" symbol) are each successful trials for progressively larger pieces of t. corpus, e.g. $\beta^3\Delta$ is obtaind from $\beta^2\Delta$ by a small amt. of "Back tracking" (94.01). What we want is a final operator that is consi, yet of hy pc. The total pc of this operator would seem to be not based on conditional pc's w.r.t. t. previous, partially

.29 successful operators.

---

.30 Viewd in this way $\beta^n$ will have a pc of ~ $\frac{c}{n}$ {pc of $\beta$ alone = c; or $c\left(\frac{2}{3}\right)^n$. ? 2.26.84; $\frac{c}{n^2}$; may be more correct: see (127.01—.11)

& t. loop will have a pc of $c \cdot l$ ← pc of recursion rule, including stop rule.

So: comparing $c \cdot l$ w. $\frac{c}{n^2}$; ∴ $l$ v.s. $\frac{1}{n^2}$ & $l$ will win for large enuf $n$.

---

∴ It may well be that t. analysis of .06 — .29 gives ≈ t. way we have to think about pc's

→ 127.17 spec

See 130.01 for explan'n: $\Delta$ is necy!

.01 : A re-analysis of 118.17 – .32 : I think the string we want t. pc of is $\beta^n\Delta$ —

.02 where $\Delta$ is a "stop" symbol. T. initial ~~pc~~ pc of $\beta$ is $c$. ($c \ll 1$).

pc $\Delta$ has some value ($\sim c$ maybe) & there are other symbols w. various pc's.

So pc of $\beta\Delta$ is $c \cdot c$, say; (No: $c \cdot (1 - \frac{1+c}{2}) = c \cdot \frac{1-c}{2}$)

" " $\beta^2\Delta$ is $c \cdot \frac{1+c}{2+c}$ ← pc of $\Delta$ — is this correct? No ← should be $\frac{1-c}{3}$ ← $\phi$

which looks right, so pc of $\Delta$ maybe $\frac{1-c}{n+1}$ — Most likely

Actually, it is $(1 - pc_{next\,\beta}) = 1 - \frac{n+c}{n+c+1} = \frac{1}{n+c+1}$

on 118.20 I had a $\phi$ here!

.08 Clearly, as t. pc of t. next $\beta$ is $\frac{n+c}{n+1+c}$ & $n$ becomes large, $c$ must ↓.

T. pc of $c$ is probably $\sim \frac{1}{n}$, since it occurs just once in a string of $n$ symbols.

So t. pc of $\beta^n$ is $c \cdot \frac{1}{n+1+c} \cdot (\frac{1}{n}) \approx \frac{c}{n^2}$ ← approx'ly

$\frac{c}{(n+c+1)^2}$ is more exact — quite likely, because of 130.01

I think $n = N$

.11 pc couldn't be $\frac{k}{n}$, because $\sum_{n=1}^{\infty} \frac{c}{n} = \infty$

SEE 82 TS 24.01 for a better analysis of

We must then, in accord w. 126.30 compare $\frac{c}{n^2}$ w. $c \cdot l$ ← pc of loop & stop rule.

So $l$ v.s. $\frac{1}{n^2}$ : It begins to look like I will win for not too large $n$!

.17 : 126.40 $\boxed{8681}$ $\boxed{\beta 45P}$ Well, if this is correct, then certainly for large enuf $n$, t. loop will have more pc (& less cc) & certainly less Least than $\beta^{(fixd\,n)}$ or $\beta^{(i)}$.

A Big trouble is how TM would ever learn t. loop method! The successive solns. are $\beta_\Delta$, $\beta^1_\Delta$, $\beta^2_\Delta$, $\beta^3_\Delta$ etc. Given each soln., t. search for t. next one is very fast, since it involves minimum modifn. of t. previous soln.

Reconc. diffuse prob. Kicked off welfare Mama Baby — No easy street for ME! Back to Academe to get my PHD.

.22 One way TM could find t. loop soln. : Say t. pc of t. loop soln. was $\alpha$.

.23 Then if TM's search always backtracked to a "depth" $> \alpha$, it would find the loop soln. It could give t. incremental $\beta^n\Delta$ as t. ~~[illegible]~~ "quick & dirty" soln. for ~~an~~ immediate need — but would find t. loop soln. after "meditating" in less hurried mode.

.27 See 94.01 for discn. of "Backtracking".

One diffy: Discovering t. operator Eval, ($\equiv \beta$) involves solving 2 level ($\sigma > 1$) subsns. If we backtrackd all t. way back to 1 level subsns., — we would not have defined $\beta$.

At a certain pt., say TM has this $\beta^{10}$ soln. T. earlier TS solns. obtaind, leading to $\beta^{10}$: first $\Theta$, then $\beta$, then $\beta^2$, then $\beta^3$, ..., $\beta^{10}$.

.32 One way to implement this: TM takes t. soln. to say, t. 3rd problem. For the $3 + r^{th}$ problem, he "adds onto" t. soln. of t. 3rd problem, & does an "exhaustive" L search for & solns. The larger I pc available

.35 to TM, t. larger $r$ he can afford to use.

What this (.32 – .35) effectively does is remove many (most) examples

.01: from t. TS!

What we want it to do is both ① take advantage of all of t. info given in t. structure of t. T.S. ② Do a broader search than would be suggested by ~~the~~ t. ~~narrow~~ small cjs's of t. T.S. Seq.

.05 An analogous problem: Consider pure sequential extrapoln. of a sequence.
We move along t. sequence, retaining t. best code we can find (for gn. CB), s.o.p. to each symbol, & then we continue searching for a continn. of ~~that~~ code. We end up w. a code for t. (short) corpus. (say of length, 10) corpus
.09 This corresps to, say, t. $\beta^{(n)}$ code
.10 On t. other hand, we can directly Lsrch for t. best code for t. entire 10 symbols, not ~~using~~ utilizing t. info in its sequential distribn.
.12 This would corresp. to t. loop code.

Now, lets modify .05–.09 a bit! In t. presentation given, we allowed no backtracking — in fact, we didn't need any. We could regard .05–.09 as part of t. search of .10–.12 —

In .05–.09, hvr., using Lsrch, we use an Lsrch for each new sub-corpus — which is a finite subsequence of t. corpus — so we get t. continuation codes for that s.c. in t. Lcost order.

Well, if we select t. lowest Lcost sub. code for each S.C., Then t. resultant seq. of sub code, is not necly a minimally Lcost code... since for sequential sc's, cc's add but pc's multiply.

→ An impt. desideratum is to retain t. good concepts that have been discovrd in various sc's & short seqs of sc's.

.26 Say I used t. T.S. in which $\beta^n$ & t. loop were solns! $\theta$ & $\beta\theta$ were solns to t. first 2 sc's. → Then, for t. first 2 & t. next 8 sc's, i.e. t. entire corpus of 10 sc's, I usd straight Lsrch. Would this discover $\theta$ & $\beta$ & t. : loop soln (which, say, has less Lcost than $\beta^{10}$)? If so, then this kind of search may, indeed, still be able to use t. info in t. ordering of t. sc's.

8781 Say I just used straight Lsrch for t. entire <corpus of .26>. — I think it mite work O.k.

One advantage of a T.S. broken into sc's: that we can discard a trial soln. early in t. T.S.; ~~[illegible]~~ Early sc's may be ~~[illegible]~~ quicker (less cc) to test than later sc's.

If this is t. only advantage of a T.S. over simply having TM try t. "most diflt" (in t. sense of most likely to be unsolved by a trial soln) sc first, then we may have to ~~[illegible]~~ rethink our present (≡ classical) conceptual basis of T.S. design → 130.17

Note that I should think of TM as normally testing a new trial op. on t. entire corpus. — unless there are certain heurs. to economize on this.

There are impt. variations on this! ① T. trial op. is in "parts" so that each part is for certain parts of t. corpus (i.e. a certain subset of sc's) (w. suitable ob(s) for identifying a sc's "subset name"). If a new op differs only in one of its parts, then only t. new part need be tested on its assoc. subset of sc's.

Of course this breaking a Op into sub-ops can be costly, pc-wise.

② Random/ (or perhaps non-random) samples of all or part of corpus — say random samples of various sc's. At first glance this seems rediculous — since one would have to do all of t. sc's eventually anyway to test a trial op. — i.e. we would throw away all of t. sequenciality info, & gain nothing! Not so!

Say t. corpus consisted of 80 sc's.; 10 sets of sc's of 8 sc's each. The sc's in each set of 8 are similar, so if a op works w. l in a set, it is likely to work w. t. rest of t. 8.

So a good test procedure is to test t. op ~~[illegible]~~ by picked a random sc from t. first set of sc's, then " " " " " 2nd " " " " , etc. — untill 10 sc's are picked. If they all work, then test all 70 of t. rest of t. SC's. → 130.17

127.02

.01: 127.02 (SN) ∴ T. "Δ" (end) symbol is = necy! This is because we are derving an operator to be usd. by TM. To use Lsrch., t. Σ pc of all such operators must be < 1 (or of known Bound). One way to get Σ pc < 1 is to have all operator dscns be a prefix set. Putting in an "End" symbol makes all such dscns. a prefix set.

[ Tho of course it is not t. only way to make a prefix set! — [redacted] — [redacted] Hvr., its an easy way to do it w. operators: Say S is t. operator dscrn. & M is a Umc. Then S operating on t. string X is M(S⌢X). Since M can figure out where S ends, it knows where to start work on X ]

.17: 129.40 / 129.11 → If we start out our TS. like 128.26, [redacted] with first a one & then a 2 subsn. problem, Then trial operator strings that begin w. $\theta(\cdot)$, Then $\beta(\theta(\cdot))$ will not be discarded [redacted] until after t. first 2 probs; & t. proper contnuns of

.21 $\beta(\theta(\cdot))$ will not be [redacted] ever discarded (except for excess [redacted] cc).

Several ways to "do" a trg. seq: Say for an Operator TM.

.25 1) Consider t. entire set $[I_i, O_i]$. Try all possl. operators {in Lcost order} on this entire set. Do t. tests [redacted] on I,O pairs in i order (i = 1 first, 2 second, etc). Try to find ops. of Lowest Lcost That work all probs.

.30 2) SCs (subcorpus) groupings of t. I/O pairs: each SC is a sequencial sub set of I/O pairs.

Say [redacted] $SC_1 = I_i, O_i, i = 1$ to 5
$SC_2 = i = 6$ to 14
$SC_3 = i = 15$ to 22
etc.

We first do an Lsrch on all ops in order of Lcost to find a soln. for $SC_1$. Say this ~~Next~~ Soln. is $Op_1$ (we may try finding

.35 Several solns., to facilitate backtracking later)

We Then try minimal modifns of $Op_1$ That will solve both $SC_1$ & $SC_2$. These are lookd at in Lcost order. One meaning of "minimal modifns of $Op_1$": The p.c. of an "$Op_2$ w.r.t. $Op_1$" is something like, t. pc of $Op_2$ given $Op_1$. This may be similar or identical to Chaitin's "Conditional probability" or "conditional Entropy".

8 78 ITS



.01: Specifically, t. pc. of $Op_2$ wrt $Op_1$ is t. pc of t. dern. of $Op_2$ given all of t. legal codes for $Op_1$. This defn. is a modifn. of Chaitin's defn., also → 133.01

.03 in which he used t. pc of $Op_2$ given t. shortest dern. of $Op_1$. → 132.01

.04: My defn. more exactly, for strings: say $x$ & $y$ are strings:

.05 say $s_{y,i}$ is t. $i^{th}$ pgm. for $y$; i.e. $M(\Lambda, s_{y,i}) = y$.

($M$ is a 2 input umc).

.10 Unnormzd $P(x/y)$ (prob of $x$ given $y$) $\equiv \sum_i \sum_j 2^{-\ell(\overset{x}{r}_{j,i}) - \ell(s_{y,i})}$

$r_{j,i}$: $M(\overset{x}{r}_{j,i}, s_{y,i}) = x$

$\equiv$ ~~[illegible]~~

$\sum_j \left( 2^{-\ell(\overset{x}{r}_{j,i})} \sum 2^{-\ell(s_{y,i})} \right)$ — I think wrong

Note: $x$ & $y$ are finite objects, so $\sum$ pc (all finite strings) $= 1$ (normzd)

.19 normzd also $\sum Pc(x/y) = 1$
fixed $y$, over all poss $x$

Is this defn. any better (or ≈ equivt) to Chaitin's defn.

Woops! actually, it looks much difnt. from Chaitin's defn!

It looks like in .19 I'd get $\sum_x pc\,(x/y) = pc\; y$.

So maybe a better defn.: modify .10 to

"semi normzg factor!" or partial Normz factor

un normzd. $p(x/y) \equiv \sum_i \sum_j 2^{-\ell(\overset{x}{r}_{j,i}) - \ell(s_{y,i})} \Big/ \sum_i 2^{-\ell(s_{y,i})}$

$\equiv \sum_j 2^{-\ell(\overset{x}{r}_{j,i})} \sum_i 2^{-\ell(s_{y,i})} \Big/ \sum_i 2^{-\ell(s_{y,i})}$

I think this has to be ommited. This is o.k., hvr.

Actually, t. defn. of .10 isn't so bad. Since it was stipulated to be unnormzd, the $\sum_i 2^{-\ell(s_{y,i})}$ ($\equiv pc(y)$) can be regarded as part of t. normzn. factor!

unnormzd $p(x/y) \equiv \sum_i \left( 2^{-\ell(s_{y,i})} \underbrace{\left( \sum_j 2^{-\ell(\overset{x}{r}_{i,j})} \right)}_{\text{This is a fnc of } i \text{ only}} \right)$ } This looks o.k.

→ 132.12

→ 132.27 (spec)

.01: 131.03: Note that t. defn. of conditional pc. of ~~op~~ strings (or operators) suggested by 131.04–.40 is practical, ~~because~~ w.r.t. applicn. to t. problem of 130.30 ff — since, when we obtain $Op_1$, we have ~~some~~ (usually), some kind of t. short code for it; — & we want $Op_2$ trials that have a short code w.r.t. t. (known) short code of $Op_1$. If we have > 1 short code for $Op_1$, we can get closer to t. defn. of condl. PC given in 131.04–.40 —

.10 Also we will be in a better position for (backtracking 94.01) if necy.

→ 139.01

.12: 131.40! A possl. way to list strings w. large pc's w.r.t. t. string y: Devise a Grammar (say Z14( or ol stoch CF Gramm) ∋ y is one of its a s's. Then hy pc members of t. resultant stoch lang. could be used as trials. → Hvr., I don't really see this as being strings of hy pc. wrt y. — Not v.G.!

→ A more reasonable way to get t. desired "strings close to y": Make small modifns in t. codes for y. These can be by t. addition (concaten) of symbols onto t. code, &/or by various hype "Modifn operators" applied to t. dern(s) of y. TM should have some such good modifn. operators wired into it ab initio. Also, of course, $TM_2$ should ~~be~~ ~~be~~ devise better ops of that sort as it matures.

.26 $\boxed{133.01 - 138.62.40}$ Has some impt. ideas on this, too. gn. ~~some~~ some short derns of y, — to make x's that are "close to y", is easy. → 139.01

.27: 131.40! → One of t. things Chaitin was trying to get, was ~~the~~ something corresponding to the Info theory equ: $H(x,y) = H(x) + H_x(y)$ $= H(y) + H_y(x)$

I guess ~~H~~ $H(x,y)$ is t. entropy of t. joint object, $(x,y)$.

$H_x(y)$ is t. entropy of y, given x. [In my CBIS paper, I misunderstood the point of Chaitin's discussion]

In t. terms of 131.04–.40:

.30 ~~H~~ ~~P~~ we want $P(x,y) = P(y) \cdot P(x/y)$.

I guess we have to put in t. normzn. constants ← Tho not necly all of them!

I'm begining to see it as working! $P(x,y)$ is t. $\Sigma$ pc of all inputs to M that are able to get the pair x, y as outputs.

E.g. t. string pair $s_{y,i}$ and $r^x_{j,i}$ are able to produce t. 2 strings y & x ~~resp~~ by §131.05 & .10 resp: (should be comma, probly)

i.e. $M(\Lambda, s_{y,i}) = y$; $M(r^x_{j,i}, s_{y,i}) = x$. — These are a bit reversed from Chaitin's Notation Q.V. CBIS p 427 col 2.

Chaitin has $M(s_{y,i}, \Lambda) = y$

.01 so unnormzd $P(x,y)$ would $= \sum_i \sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}$

unnormzd $P(x/y)$ would $= \left(\sum_i \sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}\right) \Big/ \sum_i 2^{-\ell(s_{y,i})}$

unnormzd $P(y) = \sum_i 2^{-\ell(s_{y,i})}$.

.04 so t. unnormzd eqns. satisfy 132.30: $P(x,y) = P(y)\cdot P(x/y)$.

---

To normalize: $P(x,y) =$ t. normza. const.'s selected so that $\sum\sum_{\text{all finite } x,y \text{ pairs}} \overset{\text{normzd}}{P}(x,y) = 1$

$\sum_{\text{over all } x}$ Normzd $P(x/y) = 1$ ~~[illegible]~~ (I'm not sure of this, hvr).

$\sum_{\text{all } y}$ normzd $P(y) = 1$.

Do we want $P(x/x) = 1$?

so ~~$\sum\sum P(x,y)$~~

.20 ~~$\sum_i \sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}$~~

$\sum\sum_{\text{all } x,y} 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}$ $\overset{?}{=}$ ~~$\sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}$~~

$\sum_{\text{all } x}\left(\sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}\right)$ ~~$\sum_i 2^{-\ell(s_{y,i})}$~~

this is a funct of y.

order of $\sum$'s can be usefully chngd.

$\cdot \dfrac{\cancel{\sum 2^{-\ell(s_{y,i})}}}{\sum_{\text{all } y}\sum_i 2^{-\ell(s_{y,i})}}$

woops! leftout a factor here

.25 see 134.03

so we ask if

$\sum\sum_{\text{all } x,y}\sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})} = \left(\sum_{\text{all } y}\sum_i 2^{-\ell(s_{y,i})}\right)\left(\sum_{\text{all } x}\sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}\right)$

It Looks like it couldn't! — since this is a funct of y — but none of t. other factors in t. eqn. are! (No! see 134.03)

---

Also, try this same genl. idea using Kolmogs defn. of $H(x,y)$ (see CBIS p 4 29 col 2)

---

Another Q: using defn of .01; would $P(x,y) = P(y,x)$?

also

Perhaps use Chaitin's defn. directly, only sum over all codes & use suitable normzn. or $\overset{partial}{A}$ normzn.

.03 T. normzn. eqns of 133.20 – .25 are in error; it should be

$$\sum_x\sum_y \sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})} \neq \sum_{\text{all } y}\sum_i 2^{-\ell(s_{y,i})} \cdot \underbrace{\left(\frac{\sum_x\sum_i\sum_j 2^{-\ell(s_{y,i})-\ell(\overset{x}{r}_{i,j})}}{\sum_i 2^{-\ell(s_{y,i})}}\right)}_{\text{factor } \alpha\,!}$$

Q: is factor $\alpha$ indep of x & y?

$$\alpha = \frac{\sum_i\left(2^{-\ell(s_{y,i})}\cdot\sum_x\sum_j 2^{-\ell(\overset{x}{r}_{i,j})}\right)}{\sum_i 2^{-\ell(s_{y,i})}}$$

.17 conjecture: a nec. & suff. cond. for $\alpha$ to be indep of x & y, is that

$$\sum_x\sum_j 2^{-\ell(\overset{x}{r}_{i,j})} \text{ is a constant (indip of } x \text{ \& of } i\text{)}.$$

Well, it is / perhaps a constant. This is because if we sum over both j & x, we sum over all possl. values of $r_{j,i}$ that gives any output at all from $M(r_j, i, s_{y,i})$ [see 131.10]

Actually, both the $r_{j,i}$ and $s_{y,i}$ strings are each from different prefix sets.

.22 Anyway, it would seem that t. set of strings $[r_{j,i}]$ that would
.24 give convergent outputs to $M(r_{j,j}, s_{y,i})$ would depend on $s_{y,i}$ — fixed i

Also, for any universal 2 input mac., it's not clear that $P(x,y)=P(y,x)$ for t. defn. of 133.01

.29 Look at Chaitin's original defn. of condl. probly (CBIS P427 col.2)!

$P^c(s/t) \equiv \sum 2^{-|r|}$ : $(U(r,t^*)=s)$ | Note: s & t are finite strings.

Acceptable first args (i.e. args for which output is defined) form a prefix set,
.32 for each value of t. 2nd arg. → See notes on 138.6.29

$t^*$ = shortest string $\ni$ $U(t^*,\Lambda)=t$ — $\Lambda$ = Null string

Hvr., if t. 2nd arg can be a null string, doesn't it have to have some kind of end symbol or equivt. to tell its a null string? — suggesting that t. 2nd args. as well must form a prefix set.!

.38
.39 To insure $P(x,y)=P(y,x)$, perhaps have both inputs of $M(\cdot,\cdot)$ be prefix sets, & define $M \ni \forall x,y;\ M(x,y)=M(y,x)$.

Hvr., at t. present time, I don't really ~~undst~~ remember how Chaitin's prefix set inputs worked. Anyway, it well be, That m systems of t. type usd in 134.29 → .38 That t. diffty of 134.23 → 24 would not exist.

---

For t. eq. of 132.30, It would seem That There would *have* to be a normzd form:

say $\overset{U}{P}$ are t. ~~nn~~ unnormzd probys in 132.30 &

$\overset{N}{P}$ " " normz " " " .

we know $\overset{U}{P}(x,y) = \overset{U}{P}(y) \cdot \overset{U}{P}(x/y)$ is true, from 133.04.

say $A_{xy}$ & $A_y$ are t. normzn consts for $P(x,y)$ & $P(y)$;

So $\overset{N}{P}(x,y) = A_{xy} \cdot \overset{U}{P}(x,y)$ ; $\overset{N}{P}(y) = A_y \cdot \overset{U}{P}(y)$

we know $\sum_x \sum_y \overset{N}{P}(x,y) = 1$ ; But is $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}(y)$?

since $A_{xy}$ was defind so This ∑ would be true.

---

Also, we want $\sum_y \overset{N}{P}(x,y) = \overset{N}{P}(x)$ ; & $\overset{N}{P}(x,y) = \overset{N}{P}(y,x)$

---

If $P(x,y) = P(y,x)$ (see 134.39 for away) then $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}y$ implies.

---

Also, perhaps we want $P(x,y)$ to be a max ~~[illegible]~~ (fixd $x$, varying $y$) when $y = x$.

---

→ I Think we can *define* $\overset{N}{P}(x,y)$ so That $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}(y)$.

Then automatically, since $\sum_y \overset{N}{P}(y) = 1$ , $\sum_y \sum_x \overset{N}{P}(x,y) = 1$.

.29 $$\sum_x \left( \overbrace{\overset{U}{P}(x,y) \cdot B_y}^{= \overset{N}{P}(x,y)} \right) = A_y \overset{U}{P}(y) = \overset{N}{P}(y).$$ $A_y$ is known = $\left(\sum_y \overset{U}{P}(y)\right)^{-1}$.

.30 $$B_y = \frac{A_y \cdot \overset{U}{P}(y)}{\sum_x \overset{U}{P}(x,y)} = \frac{\cancel{A_y} \cdot \cancel{?} \sum_i 2^{-l(s_{y,i})}}{\underbrace{\left(\sum_y \sum_i 2^{-l(s_{y,i})}\right)}_{= A_y} \sum_i \sum_j \sum_x 2^{-l(r^x_{i,j}) - l(s_{y,i})}}$$

well, .30 seems to do it.

It normalizes $\overset{U}{P}(x,y)$ so $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}(y)$. & so $\sum_x \sum_y \overset{N}{P}(x,y) = 1$.

T. normzn of $\overset{U}{P}(y)$ is simple.

we Then define $\overset{N}{P}_y(x)$ as $\overset{N}{P}(x,y) / \overset{N}{P}(y)$.

cancel!

.01 $\overset{N}{P}_y(x) \equiv \frac{\overset{N}{P}^{M}(x,y)}{\overset{N}{P}(y)} = \frac{\overset{U}{P}(x,y)}{\overset{U}{P}(y)} \cdot \underbrace{\frac{\sum_i 2^{-\ell(s_{y,i})}}{\sum_i \sum_j \sum_x 2^{-\ell(r^x_{i,j})-\ell(s_{y,i})}}}$

(the factors $A_y$ cancel out from ~~num~~ numerator & denominator)

for discussion of why this factor ~~[illegible]~~ is almost indep of y, but is not exactly indep of y: see 134.17–.24

.05 When $\overset{N}{P}_y(x)$, $\overset{N}{P}(x,y)$ & $\overset{N}{P}(y)$ are defined in this way, $\overset{N}{P}(x,y) = \overset{N}{P}(y)\cdot \overset{N}{P}_y(x)$ ; $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}(y)$ & $\sum_y \overset{N}{P}(y)=1$ ; $\sum_x \sum_y \overset{N}{P}(x,y) = 1$ .

.08 It would be nice if $P(x,y) = P(y,x)$.
134.29 mite be one way to get this.

[I'm not sure how I can get $M(\cdot,\cdot)$ to be universal on both args, symmetrical on both args, & have both args be prefix sets. Chaitin has written about part of this problem — perhaps all of it.]

The properties of .05–.08 ~~[illegible]~~ define $\overset{N}{P}(x,y)$ (& $\overset{N}{P}_x(x)$) in terms of $\overset{N}{P}(y)$: which ~~[illegible]~~ is probably defined adequately. (y being a finite string). Whether $\overset{N}{P}(x,y)$ is uniquely defined (in terms of $\overset{N}{P}(y)$ by these properties, is unclear)

I suspect it is not: e.g. $\overset{N}{P}(x,y) \equiv \overset{N}{P}(x)\cdot\overset{N}{P}(y)$ would satisfy all of the conditions.

$\boxed{8981}$ An adequate set of ~~of~~ postulates for a defn:

.27 1) $\overset{N}{P}(x,y) > 2^{-k} P_0(g(x,y))$

$g(x,y)$ is any recursive non-sing (≡ info preserving) mapping from pairs of finite strings to single finite strings. } Shades Chaitin! CBIS p430 col 1.

$P_0$ is any computable proby measure CPM; k is a finite constant, indep. of x & y, but a funct. of the functional forms of $\overset{U}{P}$ & $P_0$.

3) $\overset{N}{P}(x,y) = \overset{N}{P}(y)\cdot \overset{N}{P}_y(x)$

2) $\overset{N}{P}(x) = \overset{N}{P}(x,\Lambda)$

4) $\overset{N}{P}(x,y) = \overset{N}{P}(y,x)$

Normalizn. { 5) $\sum_x \overset{N}{P}(x,y) = \overset{N}{P}(y)$

6) $\sum_y \overset{N}{P}(y) = 1$

From 5 & 6 we obtain $\sum_x \sum_y \overset{N}{P}(x,y) = 1$

from 1 & 3 we obtain the eq. like 1) corresponding to a single var: $\overset{N}{P}(x) > 2^{-k} P_0(x)$.

I guess that my defns of $\overset{U}{P}$ & $\overset{N}{P}$ satisfy all t. postulates except #4 (symmetry). I'm not sure that this postulate is of practical import: But we ~~may~~ can assure it if M is symm in its args: I don't know if this is possl., hvr.

it may be unnecy for most ~~work~~ Applications.

To Review.

.60 M is a Mac., symmetrical on both args.,
, [redacted] each arg. is a prefix set. (CBIS p 427 col 2.)

.13 $y = M(\Lambda, S_{y,i})$ ; $x = M(\overset{x}{r}_{i,j}, S_{y,i})$

Chaitin has $y = M(S_{y,i}, \Lambda)$ CBIS p 427 col 2
Is this impt? It does not change t. eqns & defns of .14 – .27

.14 $\overset{U}{P}(x,y) = \sum_{i,j}\sum 2^{-\ell(S_{y,i}) - \ell(\overset{x}{r}_{i,j})}$

$\overset{U}{P}(y) = \sum_i 2^{-\ell(S_{y,i})}$ [ This is derivable from .14 & .13 ]

.17 $\overset{U}{P}_y(x) = \left(\sum_i \sum_j 2^{-\ell(S_{y,i}) - \ell(\overset{x}{r}_{i,j})}\right) \Big/ \sum_i 2^{-\ell(S_{y,i})} = \overset{U}{P}(x,y)/\overset{U}{P}(y)$.

The "normzn. constants" are:

$\overset{N}{P}(x,y) = B_y \overset{U}{P}(x,y)$ $B_y$ is to some extent a funct of $y$.

$$B_y = \frac{\sum_i 2^{-\ell(S_{y,i})}}{\left(\sum_y \sum_i 2^{-\ell(S_{y,i})}\right)\left(\sum_i \sum_j \sum_x 2^{-\ell(\overset{x}{r}_{i,j}) - \ell(S_{y,i})}\right)}$$ See 135.29–.30

$$\left[\ \overset{N}{P}_y(x) = \overset{U}{P}_y(x) \cdot \frac{\overbrace{\sum 2^{-\ell(S_{y,i})}}^{\equiv \overset{U}{P}(y)}}{\sum_i \sum_j \sum_x 2^{-\ell(\overset{x}{r}_{i,j}) - \ell(S_{y,i})}} = \frac{\overset{U}{P}(x,y)}{\sum_i \sum_j \sum_x 2^{-\ell(\overset{x}{r}_{i,j}) - \ell(S_{y,i})}}\right.$$

See 136.01.

.27 $$\overset{N}{P}(y) = \frac{\overset{U}{P}(y)}{\sum_x \sum_i 2^{-\ell(S_{y,i})}}$$

.28 $\overset{U}{P}_{(x,y)}$ satisfys eq. 136.27 (i.e. $\overset{U}{P}(x,y) > 2^{-k} P_0(g(x,y))$)
— But since $B_y$ ($\overset{U}{P}(x,y)$'s normzn factor) is a funct of $y$, it is not clear that $k$ is indep of $y$ .... However, t. $y$-dependent
.31 factor in $B_y$ is $$\frac{\sum_i 2^{-\ell(S_{y,i})}}{\sum_i \left(\sum_j \sum_x 2^{-\ell(\overset{x}{r}_{i,j})}\right) \cdot 2^{-\ell(S_{y,i})}}$$

.32 & I may be able to show that $\sum_j \sum_x 2^{-\ell(\overset{x}{r}_{i,j})}$ has an upper & lower bound (over various possl $i$) (& this upper & lower bound are not distant from one another) ← The this latter is not necy to show)
Actually, all I have to do is show that this has an upper bound (that is indep of $i$). Well, since $\overset{x}{r}_{i,j}$ is a prefix set (for each value of fixed $i$, t. $\overset{x}{r}_{i,j}$ is a prefix set over various values of $x, j$)

.01 it is clear by Kraft's ineq. that 137.32 must be true. so, if 137.28 is true, then t. post. 1 (136.27) is true.

That $\sum_j \sum_x 2^{-l(\overset{x}{r}_{i,j})} \neq 0$ stems from t. Universality of M — i.e. there must be some valid codes of t. first argt. for every value of t. first argt. (I think!) — or at least once

.10 valid codes for each value of y dcd by t. 2nd argt.

---

So: T. unnormzd & normzd probys defined on 137.10–138.10 seem O.K. All the Postulates of 136.27–.40 are satisfied, except 4) (symmetry) — & this can be obtnd if $M(x,y)$ is symm. in both argts: (I don't know if this is possl.). Whether it is or not, this 4th post. is not needed for most applicns.

T. forgg. could be t. subject of a paper/letter — sort of a kind of addenda to t. CBIS paper — for IT Xactions

---

8.14.81 Alternate Defns: If I use Chaitin's defn. on 137.13: $(y = M(s_{y,i}, \Lambda))$

Then (137.14) $\overset{u}{P}(x,y) = \sum_i \sum_j 2^{-l(s_{y,i}) - l(\overset{x}{r}_{i,j})}$ as before

& (137.17) $\overset{u}{P}_y(x)$ is as before. Also t. normzd forms have same equs.

On t. Normzn. of $P_y(x)$: It may be possl. to use t. unnormzd form to obtain probys of alternative outputs in QA (or any Operator) induction. We then use the relative probys. of the alternate outputs to get normzd probys. Just how this result would compare w. t. use of t. normzd $P_y^u(x)$ of 137.17 is unclear. In Q.A. induction ~~we usually need t.~~ t. resultant (unnormzd) proby is t. product of many $P_y(x)$'s. I'm not sure ~~that~~ t. resultant proby ratios are so directly normzd by $\frac{a}{a+b}$; A more complex normzn. may be needed: say like CBIS: P 423 eq (6) (Bott. of col II).

For my own use & perhaps as a public report or paper, I should write a discussion of why I'm interested in Normalized proby measures. Partly as a rebuttal to Levin .... but also, t. reasoning is impt. One of t. impt. things is that Normzn. permits better comparison of various proby measures [e.g. Cover's measure v.s. $P'_M$. Also, t. idea that if probys are to be used to make decisions (which is their main use) that relative probys (≡ normzd probys) are needed, & these are not semi computable.

→ 138.5.01

.01: 138.40 : An alternate formulation of Chaitin's Entropys, closer to his defns., That satisfys $H(s,y) = H(y) + H_y(s)$ exactly.

~~$H(t) \equiv \min|t^*|$ $U(t^*,\Lambda) = t$~~

~~$H(s,t) \equiv |t^*| + |s^*|$ $s^* = \min|s^*| \ni U(r^*, t^*) =$~~

.09 1) $H(t) \equiv |t^*|$ $t^*$ is t. shortest string $\ni U(t^*,\Lambda) = t$

.10 2) $H(s,t) \equiv |t^*| + |s^*|$ $s^*$ is t. shortest string $\ni U(s^*, t^*) = s$.

.11 3) $H_t(s) \equiv |s^*|$ with

1) & 3) are t. same as Chaitin's defns: 2) hvr., is difrnt.

.16 Chaitin uses (CB13 P430 col 1) $H^c(x,y) \equiv H'(g(x,y))$ where $g$ is any total recursive non-sing. mapping from a pair of finite strings to single finite string.

.18 It would be well to show that $H^c(x,y)$ is within an additive const. of $H(X,y)$ of .10

As it is, it looks like .10 is a rather A.H. defn. of $H(s,t)$. (.18) would show its not a.H. Hvr., in .10, it is clear that $t^*$ & $s^*$ have enuf info to create $s$ & $t$. Th. Q is: could $H^c(s,t)$ ever be "signifly" < t. $H(s,t)$ of .10 ?

Perhaps a more exact Q! Are $|t^*| + |s^*|$ bits all that are needed to specify $t$ & $s$, or do we need extra punctuation info that takes a no. of bits that is an ↑ funct of $|t^*|$ or $|s^*|$ ? (.28 – .31 says No punct. needed) (Maybe ln or < ln.)

.28 Well! remember that both $s^*$ & $t^*$ are members of prefix sets, so if we are given the string $t^* \frown s^*$ we can always break it down uniquely into $t^*$ & $s^*$. No "extension" of $t^*$ is a member

.31 of its prefix set. Prefix sets form a "commaless code". Even if only $t^*$ was a member of a prefix set (or only $s^*$) we could do this unique decomposition. [Hvr. Both sets do form prefix sets: see 138.6.29]

So t. string $t^* \frown s^*$ is enuf to specify $s$ & $t$ uniquely — so

.35 t. defn .10 looks quite reasonable.

As for .10 & .18 differing by a constant — this must be true, since

.39 $\overset{.10}{H}(s,t) = H(t) + H_t(s)$ exactly ← was proved by Chaitin.

.40 & $H^c(s,t) = H(t) + H_t(s) + \text{konst.} = H(s,t) + \text{konst.}$

.01: 138½.40 : Hvr., viewed in this way, t. defn. 138½.10 is still a bit A.H., but t. defn. $H^c(x,y) = H^c(g(x,y))$ of 138½.16 is only just a little bit more "intuitive". T. discn. of 138½.28 - .35 make ~~this~~ this defn. very intuitive.

It may well be that Chaitin's proof of 138½.40 is based on t. ~~t~~ invertability of both $t^* \frown s^*$ & $g(s,t)$. Perhaps look at his proof again.

One thrm of Chaitin's that I may want to review t. proof of (now that I understand that in Chaitins' ~~t~~ $P^c(x)$, $x$ was a finite string & in $U(p,\Lambda)$, t. pgms, p, formed a prefix set.) is that $lg_2(P^c(x))$ is within ~~constant~~ an additive constant of $H^c(x)$

$H^c(x)$ being t. shortest $|r| \ni U(r,\Lambda)=x$.

This thrm. suggests that adding all possl. explains (for a finite string) does not ~~...~~ give much better results than t. ~~...~~ single "Best" explainn.

— Also that my Entropy defns involving summations are not much better than Chaitin's simpler defns. involving minimal length dervns. They may be better from a practical standpoint in that multiple dervns are a good way to try to do inductn ~~Also~~

.25 Also if we actually need probability values, we can't use t. "shortest dervn" defn.

T. involvement of ~~Prob~~ Probys in .25 is perhaps a very impt. argt. In fact t. nessessarily integral nature of Chaitins $H^c$ 's is a strong argt. against them. This integral argt. v.s. 138½.10 is t. only thing against it. 138½.10 is uniformly better than Chaitins defn.. My defn. involving summations is uniformly better than 138½.10, since it does reduce to ≈ exact probabilities when it should.

.29: 134.32! On the prefix property of t. argts of $U(\cdot,\cdot)$. We consider $U(r,t^*)$. The legal second argts are a prefix set.

~~&~~ say $x$ is a legal value of t. 2nd argt.

Then for each legal value of x, there is a prefix set that constitutes t. legal ~~1st~~ first argts for that particular 2nd argt.

 Mut.ᵢ Cond. Entropy of finite Strings: 

.01: 138.6.40 : One possl./ set of defn. was ≈ Chaitins: [redacted] $138\frac{1}{2}$.09 – .11.

Hvr., there are 2 ways to do this: one is ↗ another;

1) $H(t) = |t^*|$ $t^*$ is shortest string ∋ $U(t^*,\Lambda) = t$ as befor.

2) $H(s,t) = |t^*| + |s^*|$ but $s^*$ is such that $|t^*| + |s^*|$ is minimal, w.r.t. constraint $U(s^*, t^*) = s$

.07 3) $H_t(s) = |s^*|$ w. conds

This is closer to t. defn. Chaitin used (CBIS P 430 col. 1) than $138\frac{1}{2}$.10 is. Hvr. have 3) (.07) is difent from $138\frac{1}{2}$.11 ∧ is difent from Chaitin's $H_t(s)$.

[redacted] I [redacted] think, intuitively, we want t. defns:

.12 1) $H(t) = |t^*|$ w. $t^*$ shortest string ∋ $U(t^*, \Lambda) = t$.

.13 2) $H(s,t) =$ $|t'| + |s'|$ w. $|t'| + |s'|$ minimal

$= |t'| + |s'|$ ∋ $U(s^*, t^*) = s$ ∧ $U(t^*, \Lambda) = t$ } ≈ Chaitin's $H^c(s,t)$.

.14 3) [redacted] $H_t(s) = |s^*|$ w. $|s^*|$ minimal ∋ $U(s^*, t^*) = s$.

[redacted] 1) ∧ 3) are chaitin's directly. 2) is ≈ chaitin's.

Probably Chaitin's proof of $H(s,t) = H_t(s) + H(t) + $ const can be ~~[illegible]~~ modifyd to show this is true for .12 – .19 → It can: .35 – .40 is essentialy t. proof.

So Essentially .12 – .19 are ≈ Chaitin's defns. ∧ so

[redacted] ~~[illegible]~~ that my $H(s,t)$ (of [redacted] $138\frac{1}{2}$.10) $= H^c(s,t) + $ const would

.23 be provd by $138\frac{1}{2}$.39 – .40 if it were first shown that .13 is within a constant

.24 (or identical) to Ch's defn. $H^c(s,t) = H^c(g(s,t))$ ($g$ is any non-sing. funct from pairs of strings to single " )

(which should be easy to show.) (.35 – .40 is t. proof)

No! $138\frac{1}{2}$.39 – .40 is an ~~[illegible]~~ to prove it as it is!

~~Well, one non-sing function from pairs of strings s,t to single strings, is~~

~~$s, t \to s^* \frown t^*$, where $s^*$ ∧ $t^*$ are defined by .12, .19, w.r.t. t. specific umc, U. ~~Q. does there exist a umc ∋ $s^* \frown t^*$ is t. shortest~~~~

~~[illegible] w.r.t. U, $s^* \frown t^*$ is not t. shortest code for itself. In fact $s^* \frown t^*$ (or $t^* \frown s^*$) is not a legal input to $U(\cdot, \Lambda)$, because if $s^*$ is a member of a prefix set, $s^* \frown t^*$ cannot be (unless $t^* = \Lambda$).~~ [redacted]

.35 No! For t. purpose of .23 – .24, we want to show that if .13 is used to define $s'$ ∧ $t'$, (w. $|s'| + |t'|$ minimal), then [redacted] $H^c(s' \frown t') = |s'| + |t'| =$ $H(s,t)$ of .13

(since $s,t \leftrightarrow s' \frown t'$ is a non-sing. mapping)

Actually, all we need to show is $H^c(s' \frown t') = |s'| + |t'| +$ a constant, ∧ this is easy to show: t. instructions for $H^c$ to take $s' \frown t'$ ∧ xfm it into $s^{\prime\prime} \frown t'$ are only a constant long, ∧ no punct. is needed, since both $s'$ ∧ $t'$ are from prefix sets.

138.61.40

.01: [redacted] So:

$H^c(s,t) = H(s,t) \leftarrow$ (of 138.61.13) + constant : Is shown by 138.61.35–40.

$\equiv |t'|+|s'| \leftarrow (|t'|+|s'|$ min).

$H^c(s,t) = H(s,t) \leftarrow$ (of $138\tfrac{1}{2}$.10) + const : Is shown by $138\tfrac{1}{2}$.39–.40.

$\equiv |t^*|+|s^*| \quad \leftarrow \quad [\,|t^*|$ min $]^{\text{then}}(\,|s^*|$ min $]$

So: $H(s,t)$ [redacted] (w. $|s'|+|t'|$ min)

$= H(s,t)$ ~~(w.~~ [redacted] $|t^*|$ min then $|s^*|$ min (sequential minza))

+ <u>Constant</u> indep of s & t.

---

8.19.81 → One of t. reasons (I think) Chaitin based his $H^c_x(y)$ on $y^*$, t. shortest code for y, was that ordinarily, $y^*$ is not computationally available from y.

Hwr. <u>all</u> of t. codes for y are available — in fact they are enumerable — but we never know which is t. shortest.

For my defn of $P(x,y)$ (137.14) & $P_y(x)$ (137.17) we don't have to know t. shortest code for x or y — we just sum over all of t. codes & this is mechanically poss.

.25 → We obtain t. codes of y, say, by ordering them in Lcost. Every code is eventually counted. Given any integer, n, I can find t. $n^{th}$ code (t. code of $n^{th}$ highest Lcost). Given any code, I can find its Lcost order number. In cases of codes of t. same Lcost, use lexical order, or numerical order.

.29 For this reason, I <u>may want to</u> <u>define</u> $P(x,y)$ & $P_x(y)$ differently than I have — sort of take advantage of t. fact that a one could find all of t. codes for y, if it were given y.

Note: tho I can <u>list</u> all t. codes of y countably & <u>cannot find t. shortest</u> code of y: I can also make successive approxns to $P'_M$ — but I <u>never know</u> when I've finally gotten very close to t. limit. — So maybe .25 isn't really so impt. That .29 would be true!

.01: 131.03 132.10 132.26 → 130.25 —.40 seems to be t. current problem: 130.30 —.40 in particular

One way to look at this: that there are various ways to divide up t. corpus, to group parts of it for Lsrch.

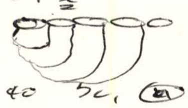
One way is: t. whole corpus is one group & t. Lsrch is done on it directly. A Second way is to form sequential sub sets of sc's:

① ————————

whole corpus is t. object size.

Sc₁ ≦

②



③ ⓐ Lsrch on $Sc_1$; ⓑ Lsrch on increment of $Sc_2$ to $Sc_1$, ⓝ Lsrch on increment of $Sc_{n+1}$ to results of $n^{th}$ coding.

④ ⓐ partial coding of all or many sc's or subseqs of sc's. This yields a first order code that is recoded using ~~any all of~~ this or any other method of coding.

---

.17 So, one goes thru t. corpus, coding sections or parts that seem simple: where t. "correct" apparently best code(s) is very likely. Then one recodes t. resultant string using any

.20 available method's — including this one.

It is possl. to do this ↑ (.17 —.20) coding each section indiply, or w. varying amounts of "conditionality" of pcosts relating to previous stuff coded.

---

.23 So: at a gn. point in t. coding of a corpus (which may itself be t. code for another corpus): One has to decide how much

8|11|81 of t. corpus to code (subsequence size) & how "seriously" to code it {i.e. one may want to just code it using very likely abss}, & whether (and how much) t. pc's usd in t. code should

.24 be dependent upon t. codes for previously coded parts of t. corpus.

---

Some examples of foregg.: Preprocessing of [redacted] observed data (calld "perception"). This could involve in visual processing edge detection, posing various hypotheses on what t. objects in a scene were.

In acoustic processing: tentative assignment of allophones (≈ phonemes); Tentative assignments of words, tentative parsing of sentences

So, one makes a prelimy run of t. corpus, coding "chunks" of it, thereby forming a new corpus, which is again recoded in this a/o other ways.

N.B. ON kind of impt. initial "Precoding" is ≈ A → D conversion. This decides how much accuracy is needed (or available), cuts out what is that to be noise, & may perform various prelimy abss.

---

T 300'

[redacted]

2 ips

[redacted]

800 bits/in

800 × 12 × 300

3000000

= 3M bits ~ .4M bytes 400K bytes

150' 40 ips

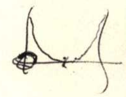
150 × 3/2 / 40

= 45'

for 1/2 second.

= 30' for 2/3 × 1/2 second

= 1/3 second.



There are many possl. ways one mite divide up a corpus for 139.23-.29 to do t. prelimy processing. (Since this process is to be "repeated until done", it amounts to a total dcm. of t. coding method). Anyway, one can have a "PLAN" which [redacted] looks (obs) at t. corpus & assigns pc's to every possl. way of dividing it up. — Then one just does an Lsrch.

.10 $\boxed{\text{SN}}$ When ~~one~~ coding a **corpus**, one should periodically do various low cc obs. These obs see if there's info present that is in any of several impt. classes. If there is, then more obs are usd to narrow things down for them. The Goal of these obs is to see if t. coding plan that one has embarked on should be changed.

T. intuitive signific. of .10 ff. is: while ~~one is codin~~ a human is coding a corpus, he may "notice" certain things about it that will change his coding plan either slightly or grossly. If t. change is small, it & t. needed obs could well be part ~~of~~ t. explicite PLAN of that time. If the change is Gross, it mite be because several obs of import have been made ~~[illegible]~~ & t. signific. of them

.26 has been computed by t. "subconcious mind" (91.01-.40) 2.092 .216 2.3 6.44 1.2k 4.9k

.27 NOTE: The present problem of how to divide up t. corpus for coding, came about as an outgrowth of t. problem of 130.17-.21; 129.01-.40: The idea is that the $\beta^{\text{lo}}$, say, soln. will be obtaind. if we divide t. corpus into sequential sc's — each of which is a separate ~~evaln.~~ problem: Then we solve each sequencially, starting w. t. first, ~~then solving t. n+1th using t. pc's obtaind from t. soln. of~~ The $n^{th}$ problem, is to find a common soln. for sc$_i$'s = 1 thru n. The ~~soln.~~ Lsearch to t. $n+1^{th}$ problem ~~[illegible]~~ is based on a ~~[illegible]~~ proby distribn. which is the cond'l. pc ~~[illegible]~~ w.r.t. the $n^{th}$ soln. This is the $P^{N}_{x}(y)$ of 137.17 (≈ Chaitin's $P_x(y)$.) 3.5k 14.5k on t. other hand, t. ~~solar~~ loop soln. will be obtained if we take t. entire corpus for which, say $\beta^{10}$ ~~is~~ is a legal soln., & we do a L search ~~on it, using t. entire~~ on it as a single object, using a apripd having no ~~previous cor~~ conditional dependence on ~~p~~ anything. It just lists t. possl. operators in ~~Lcc~~ ≈ Lcost order & tries them on t. entire corpus, discarding ones as soon as a discrepy occures.

Now here we have 2 difent. solns: that clearly depend on how t. corpus was divided up. T. 2nd soln. obtains ~~w~~ all over ~~[illegible]~~ better pc & cc. & i. Lcost, but t. entire search takes much more cc, I think.

.18 In general, an Lsrch over t. entire corpus w. minimal apri info gives v.g. final pc., but it can be done only if ~~[illegible]~~ enuff cc is available for an Lsrch of that magnitude.

.20

.22 So, we can have various ways to divide up t. corpus & have difent. amts. of ~~pcost~~ pcost/inter dependancy in 139.23.–.29 we unite be able to order ~~these~~ in terms of expected pc of soln. — ~~[illegible]~~ But it will turn out that t. methods of best expected pc (i.e. those like .18–.21) ~~[illegible]~~ may have an Lcost beyond what one can afford.

.27

.28 In .22–.27 I'm not clear on just what t. "pcost is of" in these various ways of breaking up t. corpus. It sounds like

.30 t. pcosts of difent "PLAN"'s. Hvr., I think I did ~~[illegible]~~ get an adequate soln ~~to this problem.~~ — ~~but I forgot~~ what it was! 142.17

AH! .28–.30 is one aspect of a very old, difclt problem: i.e. How to use pc's obtaind w. one c.B. (or L~~[illegible]~~cost threshold) & use them with searches of ~~a~~ difent (& perhaps unknown) Lcost threshold. Perhaps a new way to look at it: when one is doing a part of a srch — befor one does that part, say one doesn't exactly know t. params of that srch (e.g. say t. Lcost threshold is not exactly known). — Then that adds more uncertainty to ~~t. result~~ ~~of~~ one's apri knowl. ~~of~~ what t. result ~~of~~ of that srch will be. 142.01

see

.01: 141.40 → One / impt. Idea in Willis' paper is that each machine, C.B. pair defines a (usually computable) proby measure. For ~~u~~ ums's w. CB=∅, t. proby measure is not computable, but it has simpler properties, in many ways, than [redacted] other proby measures.

.05 One big/diff problem is to predict what one proby measure will give, by using a difernt proby measure! Hvr., since any proby measure is capable of making a prediction about anything, it can make a predn about what a difernt proby measure would give! Also, if t. params of a particular proby measure are / partly unknown, a difernt [redacted] (known) proby measure can still estimate t. result of t. uncertainly identified proby measure. 38k 200k 4 20 20%, income/yr!

[redacted]

.17 very likely ↓ **Re: 141.28 — .30:** One part of t. [redacted] soln. to this prob. might have been that {a "PLAN" is part of a (i.e. a dern) code for t. corpus.}

8.12.81 T. cpc of a plan itself, as well as t. cpc's of t. various operations (or other activities) within it are obtaind from previous experience on t. corpus — this is like in [redacted] Z141

I think what "t. Pcost of a given PLAN" is; its a pc that multiplied by t. other pc's that generate t. corpus, gives t. pc. of that particular code dern. for t. corpus. This ~~ }~~ pc for that particular code is ultimately expressable as one or more binary strings that could generate t. corpus.

T. pc. of a plan (again) is t. proby that t. use of that plan (at that point in coding) will ultimately give a code for t. corpus. ~~+~~ Well, no; some plans will ultimately code any corpus: we have to also consider t. expected pc of t. entire corpus if that plan is usd. T. relative pc's of ~~...~~ one plan relative to several difrnt. plans (at a given pt. in t. coding) is t. relative expected pc. of t. ~~...~~ continn. of t. coding of t. corpus, if that plan is usd.

T. idea is that in t. code for t. corpus, t. symbol for t. particular plan usd, occures just like any other symbol, & its cpc depends on t. rel. freq. of its use under those circumstances, weighted by t. pc of t. entire corpus coded ~~...~~ using ~~...~~ those instances of that PLAN.

[ Th. discovery of .17 ff was made in 1980 — apparently not! It looks like Feb 80, summer ~~of~~ if I remember correctly: perhaps try to track it down via various "Review" articles that I've written → see 73.01 (Calld H GPS) ]

2 examples of "Plans": ① GPS ② see if problem is in category of probs already solvd. ③ if not, try to xfm it into such a category. → see 73.20 for Refs, Calld H 7320 } 73.10-.40 derbs bolst these.

.34

N.B.: The sequence 140.27 — 142.34 is an impt. main line direction. Work on that stuff & get ~~...~~ it in good, clear form.

Refs to how "Plans" are simply part of t. code! 1) 80 TS 112.30—.40: 2) 80TS 76.01—77.39 ← Feb 80!
Ithis is on t. meaning of pc — I this is on PLANS.

SN An impt Q that I want to get close to in this TS work: What does it mean for TM "To have learnd an Abs(traction)"? How (quantitatively) does this effect futur TM behavior? i.e. how is t. resultant cpc hence forth usd? Is t. cc of that abs of interest in t. future? In what kinds of "Division Plans" (141.22 – .27) does "Abs learning"

.05 occure? → 146.11

---

SN In "Block coding": (Also in Coding ~~a~~ A trial soln. for an Operator in QA induction:)

The possi. codes for t. block are a prefix set, so one doesn't have to use a UIO (≡ "sequencial property) machine to get kraft's ineq.

As a result, I think t. search for t. code may be appreciably different from t. search for a sequencial induction code.

---

An interesting & perhaps impt. kind of "Division plan" for a corpus:

.18 Say t. corpus = $Sc_1, Sc_2 \ldots S_n$.

we divide up t. corpus into as large blocks as we can accomodate w. t. ~~arem~~ available cc (≡ C.B.).

Say $K$ is t. total cc we have available: Then we ~~tentatively~~ code $Sc_1$ as a unit.

then code $Sc_1, Sc_2$ as a unit: ... etc to code $Sc_1 \ldots Sc_r$ as a unit.

we then find $r \ni$ (t. Lcost of coding $Sc_1 \ldots Sc_r$ as as unit) $\times \frac{n}{r} \approx K$.

.27

So we tentatively divide corpus into $[Sc_1 \cdots Sc_r]; [Sc_{r+1} \cdots Sc_{2r}];$ $\ldots ; [Sc_{ir+1} \ldots Sc_{(i+1)r}] \cdots [\ \ Sc_n]$.

— These units having all Lcost $\approx \frac{K}{n} \cdot r$.

.33 A perhaps better way: find $r$ as .18 – .27.

.34 Then start to code t. part of t. corpus from $Sc_{r+1}$ to $Sc_n$, using a new $r_1$ as found by a process like .18 – .27, but with $K \to K -$ (Lcost of coding $Sc_1 \cdots Sc_r$ as a unit).

Then loop back to .34 until t. entire corpus is coded.

.33 is usd. & a further v.g. improvement is 153.06 – .40. → 153.06 → 144.01

This ~ codes t. corpus using as large blocks as t. available cc. will accomodate. → 153.06

143.18 —.40 is not a bad method if one isn't able to divide up t. corpus in a more reasonable way. If it is possl. to ~~do so, it should be done~~, find good reasons to divide up t. corpus in a certain way, then it should be done. How to assign an L cost max to each part is a serious problem, hvr.

A possl. way: tentatively divide up corpus into meaningful chunks: Do Lsrch on each chunk untill a soln. is found. If one of t. chuncks seems to be taking too much cc for t. Lsrch, try to divide it up into smaller pieces — or try redividing up t. entire corpus in a better, more meaningful way.

(153.06 has a good modifn on 143.18 –.40) → 153.06

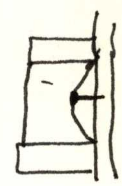
Working from t opposite direction: Say one has coded t. corpus by coding $sc_1$ as a unit, then $sc_2$, then $sc_3$ etc... t. $sc_n$. One still has lots of cc left, so one joins $sc_{n-1}$ & $sc_n$ together & codes them as a unit. Similarly, various other $sc_i$'s are joined together & recoded as larger units. If there is still cc left one continues w. even larger units, untill all t. cc is used up.

T. forgg. stuff touches on t. very general problem of ~~dun~~ "how shall I divide up a corpus for coding?": "Elementalization" is one aspect of this problem. Hvr, "dividing up t. corpus" isn't t. whole story, since one can also do coding in a heirarchical way by coding "lightly", then recoding t. resultant code, then recoding that, etc. Normally, one mixes these methods together — e.g. say t. input problem was in acoustic English — One first does heirarchical coding to get t. problem into a logically meaningful form (to TM): Then t. resultant problem can be treated further either heirarchically a/o by dividing it up into parts (e'zn.)

Each tentative method of dividing up for coding a/o heirarchical coding can be regarded as a difrnt. "PLAN" (Q.V. 142.17 —.34).

---

.19 A somewhat New approach to t. problem of $\beta^{10}$ v.s. t. loop method: For large enuf values of n $\beta^n$ has more Lcost than t. loop method .... So all we have to do is list trial solns. in Lcost order. O.k. so say we have just tried $\beta^9$ & it works o.k.: Say n=10 is t. crossover point to t. loop method. $\beta^9$ works ok. w. t. first 20, say scs's. In looking for solns to this 20 problem set, TM considers $\beta^9$, but not t. loop method / since its Lcost is > that of $\beta^9$. When t. next problem is added to t. corpus,



$\beta^9$ no longer works & we start searching. We do consider t. loop method before $\beta^{10}$ since $\beta^{10}$ has more Lcost. [actually, since t. search is n't exactly in Lcost order, we may need more complex problems before t. loop necy .... i.e. say comes before $\beta^n$ in t. search. to compare $\beta^{11}$ w. t. loop —].

conditional pc? T. impt. thing here, is that while $\beta^{10}$ has ~~smaller~~ large cpc wrt $\beta^9$, we use the entire value of $\beta^{10}$'s pc & i.e. to determine t. order of trials. to calculate t. Lcost — $\beta^{10}$'s pc. is t. product of conditional probys. — So its maybe $\propto \frac{pc}{10 \times 10}$ which can get << t. pc of t. loop for large values of "10".

I'm not sure t. "soln." of 145.19-.40 is adequate. It looks like a soln. type using t. entire set of 21 (say) problems as a "Block" to Lsrch for solns. of [redacted] This soln. simply dscrbs t. way TM would search if it had decided on t. "Plan" of doing a Block srch on t. [redacted] entire-corpus-as-a-unit.

As t. corpus continued to grow, this sort of search would become less & less practical, since t. Lcost of soln. would become [redacted] too large.

.11: 143.05! I'd like some device so that after t. loop soln. had been found, & found useful for many problems, it would be given a hype or cpc [conditional?] for some reason or other. This is one of [redacted] t. ideas of 143.01-.05!

.19 Say t. loop soln. is discovered at $Sc_{21}$. If we have cc available we continue t. search at even hyer Lcost levels, hoping for a hyer pc. soln. After we have, say, up to $Sc_{35}$, & t. loop still works, ~~we only use up to $Sc_{21}$ to test new tests. If we find ones only then do we test past $Sc_{21}$~~ (Actually in testing, we start w. $Sc_1$ & continue as far as we can go.... usually we get failure well before $Sc_{21}$.).

.25

.26 O.K. Then we want to characterize t. problems for which [redacted] this particular loop works. ... we want to define its Domain. (A standard heur device, usd before in this T.S.).

If a new problem is outside of this domain, & we know there are no other solns up to $Sc_{35}$ (other than this loop) [redacted] for a certain [redacted] Lcost threshold (since we have looked for such solns (.19-.25)), so we ~~[redacted]~~ Devise a new kind of [redacted] OB: One that recognizes t. domain of t. old loop & [redacted] invokes that loop ~~when~~ appropriate — & if not appropriate, [redacted] invokes a new operator (This OB) that we have ~~to~~ yet to discover.

It (This OB) recognizes that we have a "new kind of problem".

Say $\mathcal{L}$ is t. loop operator (essentially, $\mathcal{L} \equiv$ "Eval"). Then TM sort of "knows" that $\mathcal{L}$ works w. a certain part of t. corpus ($\equiv$ t. Domain of $\mathcal{L}$) which has a dern. of reasonable pc ([redacted] & reasonable $\mathcal{L}$cost).

Recognizing this domain of $\mathcal{L}$ may not be so easy! — The dscr of t. loop & its "Stop rule" may define t. domain well enuf

The search for solns., then, [redacted] is only over t. part of
t. corpus that is outside ℒ's domain — , which makes new solns. of
.03 reasonable cost.

The activity of 146.19 [redacted] to 147.03, which occurs after ℒ has been found, is all very reasonable, but I need to [redacted] make up rules for TM that would get it to behave in that way, & have these rules general enuf so they are really a good way for TM to behave.

.11

.12 One apparent [redacted] diffy! After ℒ is devd, we want TM to "change Mode" in response to new probs: so t. new ~~sol~~ trial solns are [redacted] "modifns" of ℒ: Any reasoning that would tell us to do this would also tell us to modify $\beta^q$ in attempts to find solns to t. larger corpus .... & This [redacted] latter is undesirable. → see 148.20 -.21.

.20

.21 $\boxed{8 \cdot 16 \cdot 81}$ 12:30P T. discussion of how to divide up t. corpus for coding, & just how completely to code each part, & what conditional pc's to use, & how hierarchical to do it starts at 130.25 — ~~144.40~~ & goes to 144.40. This is an impt. idea. I don't know if I will have to work on it now, hvr. — whether t. TS. leading to [redacted] & past "Eval" will need it, or whether t. approach of 145.19 – 147.20 is adequate.

An [redacted] impt. thing about t. "subdivision" problem of 130.25 – 144.40, is that it can (& is probly best) be thot of as a kind of "PLAN" ← (142.17ff)

.28

.29 $\boxed{SN}$ What looks like an IMPT IDEA: In general, when one has an operator that works on all or part of t. corpus, one wants to also find out t. domain of that operator. This is very impt., because it makes it possl. for TM to tell whether it [redacted] is able to solve a prob. using one of its old operators, or whether it needs to try to devise a new one for [redacted] a particular problem. While it is possl. for TM to make an estimate of t. Domain of an operator w.o. having any negative instances, it is usually a lot easier & more accurate if some negative cases are available.

This idea enables TM to "divide up a corpus" in 2 senses!

.01 ① Sequencially (considering a sequence of probs. given to TM.) it can decide that certain probs are solvable by certain Ops., & certain other probs. by other ops. — & it will have very naturally constructed obs. to tell which Op to use when.

② A single problem itself can be divided into parts, & suitable operators applied to those parts.

---

T. forgo. (i.e. ① (.01) certainly) seems related to t. problem of dividing a corpus into parts that was refered to in 147.21 – .28 [w. body of work mainly on ~130.25 – 144.40]

64 chars/line
~32 lines/p.
2k bytes/p.

3M bytes = 1.5k pp ~ 5 yrs!

4 discs ~ 20 yrs!

800 bpi
12 × 300 × 100 By/in
360,000 Bytes.

180 pp. on one 300' tape

3 ft/sec
36"/sec
≈ 100 sec for 300 ft.

$360. for 360k bytes.

.16 Re: t. "Soln" of 145.19 — 147.20 (in particular, 146.26 – 147.11):
This mite be regarded as a particular kind of "PLAN"; anyway, we can compute t. pc of using this ob. to divide up t. corpus into parts solvable by difnt ops. — & we can contrast this pc. w. that obtained thru difent. coding methods.
.20 — so if method is not A.H. On t. other hand, if its not A.H. — it should
.21 be able to deal w. t. difty of 147.12 – .20 in a natural way!

---

O.k.: Re: 146.26 → 147.11 again

.22 Say we have $\mathcal{L}$ as a soln. up to Sc20. Then we get Sc21. We try to continue Lsrch past $\mathcal{L}$, for a while, but we find nothing new. Then, t. Lcost threshold becomes hi enuf to use an Ob that recognizes t. domain of $\mathcal{L}$ & decides that Sc21 is probly not in that domain — so a new operator search is tried for Sc21 alone. Note that the Lcost for t. search for this operator is relatively small, because we only have to test trials on Sc21 alone — We may be able to do this Lsrch using t. pc of this new operator only, (not multiplied by t. pc of $\mathcal{L}$).

.30 [ T. possy of using this short Lsrch is perhaps t. essential point of "settling" on a certain coding of a part of t. corpus & leaving it that way. — Hvr., t. conds. under which this sort of thing is to be done, must be clearly understood — also t. conds. under which Backtracking is to be done . . . i.e. t. "undoing" of "leaving it that way".

.35

.36 8.17.81 T. forgoing seems also closely related to t. defining of ngms (or, more
.37 generally, t. defining of arby abss) in Σ141. Actually Σ141 may be regarded
.38 as a particular "PLAN" for coding a corpus. It can be used as either a sequencial coding method or a Block coding method. In t. Block coding method,

$\int_{-\infty}^{+\infty}$

AB

$\int_{-\infty}^{+\infty} e^{-x^2}dx$

8.17.81 TS 

We go thru t. corpus looking for unusual (char. / zgm.) freqs & make appropriate pc assignments.

.02 Then we go thru t. corpus again looking for unusual zgm. freqs & make appropriate pc assignments & datas & re code t. corpus. Then loop to .02 untill nothing more can be done.

---

To use it as a sequencial method, we Block code t. corpus up to a certain pt, α. Then, past α, we use (t. datas & pc's obtaind in t. Block code up to α) to code t. rest of t. corpus. ......(I'm not so sure its v.g. viewed as a sequencial coding method, hvr.).

---

On second thot. Z141 is more of a CPM-method of coding: (like

.16 linear regressn.) — & is usd to Block code large corpi.

---

from 122.40

For MaxM; I don't think it did it strictly in accord w. CBI: I did not use t. corpus augmented w. 1 hypothetical data pt. & obtain t. ... Aside from this, would it make much difference?

**Def** A "large corpus": its $\frac{cc}{pc}$ (& L cost) is > cc available.

As far as I know, t. only way to code a "large corpus" is to chose a CPM & code it wrt that CPM. T. method of chosing t. CPM may itself be fairly elaborate. E.g. we can view t. determination of t. coifs & $\sigma^2$ in linear regsn. coding as t. "Selection of t. CPM" part. — or, if we use t. entire MaxM, we can regard t. decision to use MaxM as t. entire CPM selection part.

~~800.100~~ ~~64000~~
1600 pp
64000 lines
40
400 lines/p →
160 pp
200 lines/p
→ 360 pp.

**Def** A "small corpus": anything that is not a "large corpus": i.e. something that can be directly coded w. Lsrch & give an "acceptable code" w. t. available cc. i.e. t. $\frac{cc}{pc}$ of a usable code is ≤ available cc.

---

Refs: 147.12–.20
148.22 ff
148.30–.35

.28 T. Main Difty at Present is 147.12–.20: One way to look at it: At what point does one say: "I have an adequate code for this part of t. corpus: I will just leave it that way & work on other parts of t. corpus"?

.31 More generally, say one has found part of t. corpus that one feels that one has solved. One mite have strong feelings about how to divide up t. rest of t. corpus into problems → each should be assigned a relatively indip. Lsrch. This could be true of several sections of

.35 t. corpus even befor one has "solvd" any of them.

In 147.12 → .20, we don't try to break up t. corpus untill we have found t. "Satisfactory soln, 'Eval'" — there was no apri idea of divisiveness .31–.35.

A plans output will be a coding of t. corpus. It need not be a u.g. coding, hvr. Good "plans" while have typical names of hy pc & they will tend to ~~have~~ give codes of hy pc's for t. (parts of) t. corpus they are applied to.

— If a particular Plan's inversion, tends, on t. average, to yield a hy pc code for t. part of t. corpus it is applied to, then that plan ends up w. a hy pc. This is due to t. way pc's are assignd to PLANS. PLANS

.14 Anyway! CPM's & PEMS like Maxm or Z141 are all "plans", & as such, their names acquire cpc's after they've been usd on t. corps.

Remember Note on how to estimate pc of abs. that has been applied using various difrnt. CB's. — usd idea of one CPM estimating t. output of a difrnt CPM — See 142.05

One "plan" could be a method of dividing up t. corpus for lsrch... either apri or apsi or mixd, to solve difftys like 147.12—.20 & 149.28. Each such "plan" would be assignd a pc, depending on it (empirical) past success in getting hy pc for things that it helpd code. These pc's would (probably; perhaps) be assignd using a Z141-like reasoning.

N.B. Actually, any method of assigning pc's to sc's is o.k. T. only condition is that: t. pc. must be normz or normzble. — (& preferably, shouldn't assign zero to any sc). — So summd over all possl. sc's; $\Sigma = 1$.

In this sense, any plan of this sort corresponds to a CPM (or PEM). Methods of dividing up t. corpus and of assigning pc's to sc's & so they correspond to CPM's. — Hvr., t. possibility of normzn. isn't so clear — so it may be diffc. to compare (& give relative wts) to difrnt "plans" of this sort.

T. set of all possl. plans has $\Sigma$ pc = 1, & t. dcrns of these plans form a prefix set.

.33 One sort of soln. to t. PW (= Pam weighting) Prob. is that t. relative wts of various PEMS (= plans) are $\propto$ t. resultant pc's of t. sc's that they help code: These wts (= pc's of t. "Names" of t. PEMS) are assignd via Z141.

.37 One problem here, was that when a gn. plan was used on previous sc's, t. CB usd was difrnt from that of t. present sc. For what looks like a new approach to this problem, see 150.14 R & also 142.05. T. Basic idea is that most PEMS (no matter how good or bad they are) can be used for making probilistic estimates of any thing using any available info. So they

can also be used to estimate pc of a corpus wrt a particular plan, & w. CB = A when & has been usd in t. past w. difrnt t. sc's w. different CB's.

151.01
152.01

.01:150.40 Actually, t. main $\boxed{\text{PW prob.}}$ was more involved: I think in one part of it, one had several PEMs that had various pc's, & t. sub corpus had various pc's wrt each of t. PEMs. T. Q. was, how much wt. to give t. future predns of each PEM. In particular, say one had an enormous no. of PEMs. One could pick out a sub set of PEMs that predicted what one wanted. Any wts assigned to the members of t. subset alone would not free us from t. A.H. effects of this choice.

In general, picking t. single PEM that did best prediction in t. past, is not t. best way to do prediction (e.g. regular linear regn. v.s. Maxent).

I think a big Question was how to estimate t. limits of accuracy of an arbly presented — (possibly A.H.) PEM.

I did a lot of work on this at MIT (~ 1973) — I don't think I got much good results then — I may have gotten some, hvr. — but I may have gotten some better results since then. — say within last yr. or 2.

In particular, t. assignment of pc to t. name of t. PEM was very difft — if t. PEM was obtnd, say, from a person, or other source from which a pc would be difft to obtain. E.G. say one was given a difnt PEM by each of several "adversaries" advocating difnt. courses of action, to be based on these PEMs.

Σ

.01: 150.40: One way to deal w. t. dif'ty of 150.37: Each plan should contains specifications of exactly how t. work is to be carried out. This includes specification of CB's, if such are needed. — In many cases, a plan need not include CB specifn. — since t. thing t. plan does is just a simpler algm. to be done & no cc. is measured.

If there are ≃ same version of a plan, but w. diffnt. CB's, then a certain amt. of Data pooling can occur... but only if t. variation of resultant pc of cc with CB is considered — a functional Relation. We mite look upon this "pooling" of info on difnt versions of a gn. plan in this way, as being a "Hyperorder plan".

.15 As I see it, t. Big Problem now: I am coding this part of t. corpus, α. I work on it up to a certain cc level, then I am tentatively

.17 satisfied w. my code for α. I have t. soln. operator, $S_{\alpha,1}$. I find the Domain of $S_{\alpha,1}$, which is $D_{\alpha,1}$. (& this, too, I decide on after a certain cc expenditure).

$D_{\alpha,1}$ enables me to recognize parts of α & t. appropriateness of t. Soln. $S_{\alpha,1}$

Then, I turn to a new part of t. corpus & try to code that using cpc's dependent on t. code of t. operator $S_{\alpha,1}$. I terminate this Lsrch when I'm satisfied, — then loop to .17, etc.

T. Q is: at just what point am I satisfied w. t. code for α, say? & Then, what chunk of t. corpus do I next chose for coding?

this would solve t. prob. of

.30 .15 ff may not be t. problem: e.g. I could just quit further work on α after I found t. first Lsrch Soln.. T. problem of how to break up t. corpus into reasonabley sequencially worked on chunks is of import, hvr.

4% of 35k 1900,00 4 = 1.4K

.33 Suppose I try various operators on a certain set of sci's, R. I find an op. that works on a subset of t. sci's in R, & I'm able to characterz. t. Domain of that op. I can then try to find a new operator that works for all of or part of t. rest of t. sc's in R. — If I use this technique, R can be a "Large corpus", that would be inaccessable to being coded "as a whole" by Lsrch.

T. approach of 152.30 may be O.K.! It is a kind of "PLAN": 152.33–.40 is a more general method (slightly). "t. initial subset of R where we have a soln." ← The stop rule for Lsrch., depends on t. pc of t. operator & t. pc of its apparent Domain. ... I don't know just how this dependence on t. 2 pc's goes, exactly.

.06 → (143.40, 144.40) One idea for a rule on when to decide to break up t. corpus! Say one has a CB of A.

.07 One has Lsrch'd & used up ≈ $\frac{A}{10}$, say & one has an operator that works

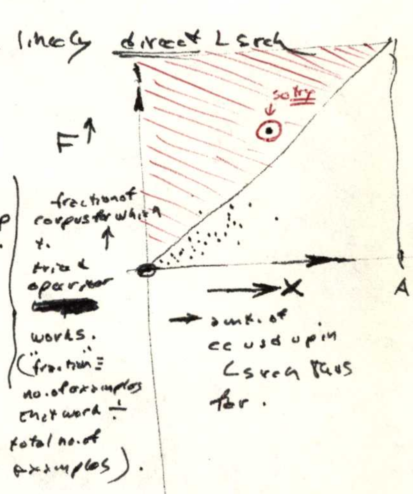
.08 for ≈ $\frac{1}{10}$ of t. problems. At such a time one should try to see if this operator & its domain (or a subclass of its domain) can be defined at hy pc — & then do Lsrch over t. rest of t. corpus. Now, $A \to A - \frac{A}{10}$ & t. remaining corpus size → ×.9 ⟶ loop back to .06 & Lsrch out t. rest of t. corpus.

To generalize, t. "10" of .07 & .08 can be n — any not so large number.

T. rationale of .06 ff! If one used up $\frac{A}{10}$, then using up .9A would, using direct Lsrch, yield an operator w. a pc of only $\frac{1}{9}$ × t. pc of t. previous operator — This isn't much additional complexity & one has .9 of t. entire corpus more to do — so its unlikely direct Lsrch will succeed, so one tries breaking up t. corpus.

So as one does trials, X increases toward A. When, for a particular trial, $F \geq \frac{X}{A}$ ← fraction of available cc that's been used up, then this trial op. is a good candidate for trying to define t. domain & breaking t. corpus up at that point.

$F \geq \frac{X}{A}$ is the region




So, try to get this worked out in more detail; then try to apply it to t. problem of 147.12; Also do $\boxed{145.19 - 147.20}$ → it may be O.K. for 145.19 – 147.11, superficially; 147.12 may be a serious defect.

Is .06 ff similar to 143.18 – 144.40? → yes: its ≈ identical,

.36 $\boxed{8.19.81}$ but .06 has an impt. improvement: i.e. we start out by attempting to code t. corpus-as-a-whole — then we look at t. fraction of problems that are solved. This is a more natural way to divide up t. corpus than t. ~~methods~~ simple sequential method of 143.18 ff.

Also .06 considers t. domain of t. partial solution.

T. main Rationalle of 153.06 – .40 (& 143.18 – .40) is That it trys [redacted] to code t. entire corpus in ≈ t. cc That one has available (as a "Cβ").

---

.03 $\boxed{\begin{matrix}820\\81\end{matrix}}$ **FN** This idea of having Various **PLAN**s on "How to divide up t. corpus" (sequencially
TM &/o heirarchically) seems to be opposed to Levin's idea That Lsrch may, indeed, be in some sense ≈ optimum from a practical stand pt. → Hvr., → I'm not ready yet to give up t. possy that L. may be rite.

If L. is rite, this means That in some sense, less apri info about t. world is needed by **T.M.** (& for **OOL**, perhaps).

**If L. is wrong, this means** That I do have to devise a fair amt. of apri info. for TM to start w.. **Methodologically**, hvr., I will probly end up w. t. same course of action **indip. of** whether **L. is** rite or wrong on This point.

—— **Except that if** he's rite, then it will be easier for me to decide upon an ≈ optimum course of Behavior for **TM**...... If he's wrong, This "opt. behavior" (or even "adequate behavior") is more tied up w. t. kind of world we expect TM to work in.

Viewed in This way, t. Q. of just how A.H. These "how to divide up t. corpus" Algms are, **is an impt.** theoretical Q. as well as **practical Q**.

---

What looks like an impt example v.s. Levin's hypoth:

.26 We code a certain section of t. corpus, $\alpha$, w. code $S_\alpha$, at cc = $C_{S\alpha}$ & pc = $P_{S\alpha}$.
We find ~~a way to reco~~ an ob. to recognize $\alpha$, t. domain of $S_\alpha$; This ob is $O_\alpha$, its pc is $P_{O\alpha}$; its cc = $C_{O\alpha}$. We then try to code t. rest of t. corpus, $\beta$ — by using $S_\alpha$ & $O_\alpha$ to recognize & work on $\alpha$, & leave $\beta$ for new ~~trash~~ ~~open~~ operator trials. When we find a soln. for $\beta$, it is $S_\beta$, w. pc = $P_{S\beta}$ & cc = $C_{S\beta}$. The Lcost for t. search for $O_\alpha$ is $\frac{C_{O\alpha}}{P_{O\alpha}}$; Lcost of srch for $S_\alpha$ is $\frac{C_{S\alpha}}{P_{S\alpha}}$. | Lcost of srch for $S_\beta$ is $\frac{C_{S\beta}}{P_{S\beta}}$

.34 Total Lcost of this soln. = $\frac{C_{O\alpha}}{P_{O\alpha}} + \frac{C_{S\alpha}}{P_{S\alpha}} + \frac{C_{S\beta}}{P_{S\beta}}$.

.35 However ~~would~~ t. Lcost of directly finding this soln. would be $\frac{C_{O\alpha} + C_{S\alpha} + C_{S\beta}}{P_{O\alpha} \cdot P_{S\alpha} \cdot P_{S\beta}}$

which ~~was~~ is >>> $\boxed{.34}$.

Furthermore, There mite very well be other solns of Lcost $\lesssim$ .35

8.20.81 TS 

.01 That we have missed in using this particular "heuristic" or "Plan".

.02 A possl. justifn. of t. soln. method usd to obtain t. soln. $\alpha_\alpha, S_\alpha, S_\beta$,

is that this soln. method was, indeed, "most likely", in view of t. limited C.B. available.

T. way this mite work: Say we have a stable of 3 plans:

1) (A) Direct Lsrch on t. entire corpus taken as a block.

2) (B) ≈ 153.06, or some other means of dividing up t. corpus that gives of t. soln. method of 154.26 – .40

3) (C) some other method of dividing up t. corpus for soln.

Now t. 3 plans have pcosts $P_A$, $P_B$, $P_C$ resp.

$P_A$ is very small $= \epsilon$ for corpi of t. size $\alpha \cap \beta$, since in t. past, this has rarely, if ever, obtained a soln. for t. available CB.

$P_B$ & $P_C$ are ≈ .5 each. (or $.5 - \frac{1}{2}\epsilon$)

Using t. standard Lsrch, A, B or C must be t. first symbol in t. trial soln.

However $P_A = \epsilon$ is so small, that it doesn't get tried until the trials invoked by B & C become very small in pc.

I think t. symbol A is invoked when t. Lcost index has gotten to shortest cc possl. for a trial of any kind $\epsilon$ ($\equiv P_A$).

So B mite well find a soln. befor much cc is spent on A, t. direct Lsrch on t. entire corpus-as-a-whole.

.25

.02 – .25 mite be a (temporary) (partial) justifn of L's hypoth. (q.v. 154.03 – 155.01)

8.21.81 → Note that A, B, & C are, in general not sequential coding methods: they are allowed to look at t. entire sc. befor devising a code. The code can be constructed in any way. The only constraint on t. code is that it be possl. to xfm. t. code into t. sc. w. finite cc (i.e. — that it is a legitimate "code").

8.23.81 A kind of Genzn. of t. ideas of .02 – .25:

Say TM is askd to code a sc. (which may or may not contain > 1 problem.) after having had much experience coding other sc's of "similar ilk". He is also Gn. a C.B. for this coding problem.

On t. basis of this known C.B., & his experience of t. past, he is able to assign apriori pc's to various coding methods for that sc.

These coding methods involve, first, a bunch of standard obs. on t. new sc. ... followed by t. appitation of various coding techniques w. pc's based on these obs.

8.23.81 TS 156

.01 One common method of coding a SC: (this is a SC consisting of many QA's): Try various ops in Lcost order. Certain of them will work on certain QA's — certain ops will work on others. Then (& perhaps simultly) try to find ways to correlate OBs w. the appropriate Ops. The correln. need not be exact to get a (partially) workable code.

Note that the obs may be very much involved w. the ops —
E.g.: we can identify an expressn. for which "Eval" works, by applying Eval & seeing if a pure number eventually results. — Obs of this sort are of
.17 relatively hy cpc w.r.t. their appropriate Op.

For .01 ff, we may want to try randomly selected QA's to try each new trial Op on. .... or try each op on all of the QA's in the corpus.

8 24 81 Mon: My present impressn. is that pure sequential coding is rarely, if ever, used by humans. A common method of coding is to divide the corpus into blocks & code each block (by perhaps Lsrch), one after the other, using cpc's obtained from previously coded blocks. (& using obs to recognize what ops fit what problems)

We may save the first 10 (lowest Lcost) codes that appear for each block — but if this is done, the meaning of ≤pc of ~~subsequent~~ obs in ~~subsequent~~ codes of subsequent blocks, is rather complicated — since they depend on which code was used for the previous blocks.

.30 .01 – .17 is a common method — The idea of trying to solve whatever parts of a corpus one can solve, using various ops — then trying to recognize just what it is (i.e. an OB) about a problem (or a part of the corpus) that makes a certain op (or a certain coding method) work well for it

other than giving more accurate probys, Those multiple codes enable one to backtrack if reasonably typical codes for future SC's are not found.

Consider t. problem of learning "Eval" using arith examples (unary & binary functs). Say we used t. method of 156.01–.17 & .30–.40. [It would probly work fine for simpler (1 layer) unary & binary functs. It would find ops that worked for certain probs. & then it would discover obs to tell which ops to use on which problems. T. "soln." would be a set of ops & corresp'g. obs. This soln. is a bit ~~A.H.~~ A.H.

.10 By continuing Lsrch, one mite find a "better" soln. — with
.11 hyer pc & conceivably lower CC. Hvr. T. reason such a soln. mite be found after t. initial (≈ A.H.) soln., is that t. initial soln. has ~~to~~ a hour that gives it low CC: i.e. after one has found a op & a corresponding ob to tell when to use it, one need only try new/trial ops over t. remaining problems not solvd by t. first op, ob combination. However, if one multiplies t. pc's of all t. separate ops & obs (that have worked) rite together, one could get a rather low pc — {say, compared to a non-A.H. method, found by a Global Lsrch.}

.20 Other than t. direct Lsrch of .10–.11, one could obtain a better code by looking at t. final initial code & recoding it/at a "hyer level".] This hyer level coding would note the similarities in the op-ob pairs for +, −, ×, ÷ etc. & devise a simpler, hy pc form for them all — ~~[illegible]~~ — possibly w.o. use of an Ob at all. This is probably t. way one would actually find a "better code" of this kind.
.27

**8.25.81** In t. presently contemplated T.S. (arith learning), there are 2 sections of t. seq. that seem very similar:

Section ② ⓐ learning $\beta$ then $\beta^2$ then $\beta^3$ .... $\beta^{12}$ say!
.30 ⓑ learning t. "loop" method of Eval. — which ~~has better~~ has
.31 better (cost, ~~if~~ than $\beta^{12}$ if one uses a Global criterion.

109 v.s. .11

Section ~~②~~ ① ⓐ learning ~~2~~ 1, 3, + i.e. ~~a~~ ob to recognize this problem type.
" 3, 7 − " " " " " "
8, 2 × " " "
2, 1, ÷ " " " " " "

ⓑ looking at t. solns. of ⓐ & obtaining a more genl. soln. of lower cost than t. 4 ob, op pairs. — this mite be done as in .20–.27 or it mite be done in a manner closer to t. way in which .30–.31 is done.

So 1a : 1b ≈ 2a : 2b.

82581 TS

158

↑ may be 82681 ?

I want to characterize just where I am in this TS problem,

so I can state t. problem(s) needing soln most clearly,

\# The TS being workd on goes up to Learning t. function "Eval"

from examples — then continuing past this.

Various parts of the T.S. have been workd on, so ~~[illegible]~~

I have a lot of pieces that "fit together" to some extent.

I guess what the problem is, is the method of searching for the

partial solns. How to divide up t. corpus (what epist. justifn)

& what ~~[illegible]~~ dependancys to include in t. cpc's, & whether

& how to do hierarchical coding ("= coding & recoding).

.17 My present impressn. of one common method of coding a corpus:

.18 Go thru t. corpus looking for regys. Any that one finds, one uses

to code part of t. corpus. This partially coded corpus

.20 becomes a new corpus; loop to .18 untill nothing can be done.

T. technique of .17 – .20 includes both ~~[illegible]~~

coding by breaking corpus into parts, & ~~[illegible]~~

.23 hierarchical "coding & recoding at ~~[illegible]~~ higer levels".

For an infant T.M. there will not be many coding methods available.

.25 Given a seq. of examples for "Eval", one natural way is to try

to solve t. problems individually: each w. its own Lsrch; then try

.27 to recognize which probs have t. same soln(s).

Say one is gn. a large set of "eval" problem examples

(Not necly in order of difty!!): ← NB: an interesting outgrowth of present Method.
One trys to solve ~~[illegible]~~

whatever one can, using .25 – .27 — this will solve t. first level unary & binary

function problems. — This ↓ t. effective corpus size. Next,

using ~~t.~~ cpc's thus obtaind, do Lsrch on various "easiest looking"

probs of t. rest of t. corpus. This way ~~[illegible]~~ we obtain t. solns

$\beta, \beta^2 \ldots \beta^{12}$, say.

.35 ~~[illegible]~~ A Big Q is "How to we find t. Loop soln."?

pts/word
150 ~~bits~~
× 32 =
4800 ~~bits~~ pts

Re: How t. loop soln. is found (after $\beta^{12}$ is found): $\beta^{12}$ was found using ~~essentially~~ ≈ 158 [redacted] .17 —.23 $\beta^{12}$ was more or less satisfactory — it took small cc because each successive modifn. of $\beta^{n}$ required only a little cc (ie Lcost).

If one was very short on cc, ≈ $\beta^{12}$ would remain as optimum soln. Only if one had extra cc "to contemplate w," or, for some reason, one suspected that there existed better solns, would one try a more Global Lsrch, of t. kind that would eventually yeald t. loop soln.

4×10 = 40,
50 5 bits.

.10 Well, that may be it. If one can afford more cc, it's clear that t. soln. of 158.17 —.23 was not very Global. The "grain size," (so to speak), was a single Eval example. — Actually, it's t. size of increment of "t. corpus up to now" that's relevant: this size is "a single Eval example" in 158.17 —.23.

Just how one should best ↑ t. "Lsrch chunck" size, [redacted] (for a gn. available cc) is unclear. 4098 6

.20 One way would be to do Lsrch on (t.-first-i-problems-as-a-block,) 8%

.21 first for i = 1, then 2 . . . . untill t. available cc was exceeded. 200k; 40k 16k

$\boxed{82881}$ Fri 9:40P: 10:15P: 10:36P: while it's clear (ie .10) that t. 158.—.07—.23 soln is not very global (ie. its very "el."), its not clear just what a slightly less

.24 el. soln. would be. One can always do a global Lsrch of t. entire —

.25 — corpus taken-as-a-whole: If one can afford t. cc! Usually t. steps towards non-elism are smaller!

.27 One way is to bunch problems together that seem similar in some sense, & do an Lsrch for [redacted] an integrated soln. for that set of probs. In t. case of "Eval" probs — this set would be t. entire corpus.

.30 $\boxed{83081}$ Sun 11PM: [Some reviewing of recent writings]: That t. General problem [of which t. Loop soln. v.s. $\beta^{12}$ soln. is an example) is that of using 158.17—.23 — which is a good el. way of coding — v.s. obtaining a less el., more global soln. (better Lcost soln.) using a ~~more global~~ search over more global chunks of t. corpus. So t. idea is .10—.21: If one has t. extra cc. available, what's a good way to do a more global (less el.) Lsrch?

→ .20 & (.24, .25) / & .27 are 3 suggestions — but I don't see a way to treat t. general Q.

→ Another way to continue after one has coded t. entire corpus rather el-ly: (Say it is coded in small parts w. recogn. Obs for each part): We then try to ~~oven~~ recode & recode at hyer levels t. mitial code.

.01 2 other genl. ways: (1) Start w. t. most el. way of coding & try to find a somewhat less el. way.

.03 (2) Start w. t. most Global method & try to find "el.zns" (≡ more el. ways).

.01–.03 are rather genl. ideas; perhaps try to find examples of each & try to genz.

[SN] Note: T. loop soln. is not t. most non-el. soln!; trivial ~~[illegible]~~ It's t. result of starting w. t. Operator Θ & Θ's Domain, then doing an Lsrch.

9.4.81 TS :

This page numbering is [redacted] arby jump; I forgot to bring [redacted] recent TS. Notes to NIP

→ 161 [redacted] ~~201~~

.01 : 160.40 [redacted] On the "$\beta^{12}$" v.s. "Loop" Problem: One genl. way to look at it:

That I want to be able to try Lsearchd solns. at difrnt. levels of non-elism.

$\beta^{12}$ is fairly el. , [redacted] t. "loop" soln. is much less so.

As was previously noted, each method of elzn. — of "dividing up t. corpus" can be viewed as a "plan", having a known cpc.

I think such "plans" amount to PEMS (or CPMS) — (tho perhaps "Plan" is [redacted] more general term — so "A pem is a Plan" but not necy visa versa.)

One has all those difrnt. possl. plans for coding t. corpus. — They can have difrnt. amts. of elzm. in them.

One way to look at t. output of a plan, is t. cc of using that

.20 plan, v.s. t. pc it assigns to corpus. In general, we have one trivial coding of t. corpus (t. ~~identity coding~~) which has low cc & very low pc. ~~On...~~ It has an acceptable cc, but t. Ccost of finding this code is ~~...~~ its cc rather than its Lcost ($\equiv \frac{cc}{pc}$.) This is because there is no "real search"

.27 involved.

Another low cc, but hyer pc code is t. Bernoulli code — which measures freqs of each of t. symbols used. ~~...~~

Z141 is another simple coding method of much less cc

.30 than its $\frac{cc}{pc}$.

---

Lsearch gives us a way to order trials in $\frac{cc}{pc}$ order.

We would like a method of ordering ~~...~~ actual solution codes (trials that actually fit) by $\approx$ pc order; perhaps $\frac{cc}{pc}$ order. Lsrch does enable us to order solns. in $\approx \frac{cc}{pc}$ order, but it takes ~~much more cc~~ $\approx \frac{cc_i}{pc_i}$ to find t. $i^{th}$ soln., rather than $\approx cc_i$.

T. 3 coding methods of .20 — .30 have various pc's but their cc's are always rather low — $\approx$ t. cc of a "trial" itself, rather than $\frac{cc_i}{pc_i}$

T. idea of breaking a corpus into "Physical Parts" is certainly gonzable in useful ways. ~~Exit~~ Most trivial, is ~~non-localization of t~~ that t. parts not nec'ly be ~~phy~~ physically continuous : e.g.
Rather broad — i perhaps very genl.


so (B) is in 2 connected sections.

.04 One ^genzn. of "Parts" : That we are able to divide t. coding problem into 2 parts A i B, ∋ by first working problem A (which need not be a coding problem) then working problem B (in view of t. soln. of A) [B need not be a coding prob. either] Somehow one has solved t. prob. of coding t. corpus.

This is an example of an "AND" decomposition in SP nets.

"OR" decompositions are simply alternative soln. plans of alternate codings of t. corpus.

.18 An impt. example of "AND" decomposition is ~~to~~ posing 1 or more SUBGOALS.

In t. case of t. $\beta^{12}$ v.s. Loop problem: Say one has already coded t. initial (or an initial) section of t. corpus, using t. operator ~~$\Theta$~~ (which can deal w. 1 level of binary ~~and~~ a/o unary functs). If, at this pt., one decides to do an Lsrch for t. entire "rest of t. corpus" using cpc's based on $\Theta$, then we would get t. Loop soln. — but what is t. justifn. of this particular way of trying to "divide up" t. corpus?>>

.22 → Hierarchical coding is a way to "Divide up t. corpus" in t. genl. sense of .04 : I.e. Code on first level, Recode on 2nd level, ~~R~~ .... recode

.25 on nth level. Z141 is a simple example of this kind of coding.

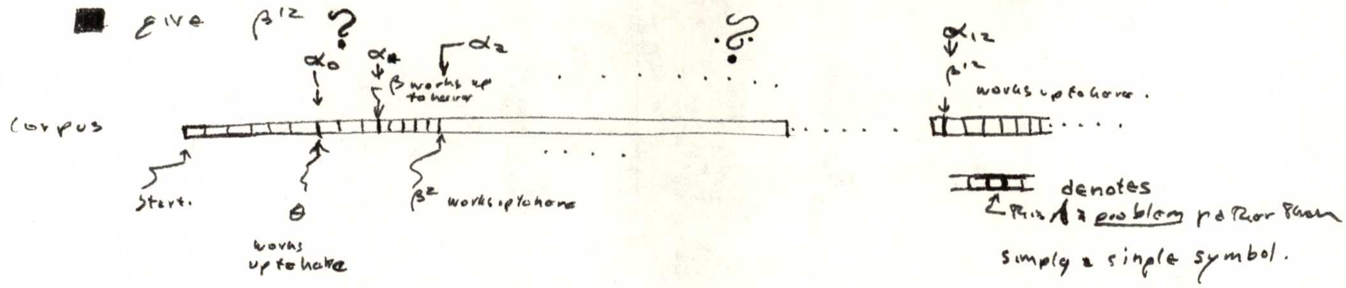
Well, after one has coded part of t. corpus using $\Theta$, i one notes that $\Theta$ has a simple domain dern., it is perhaps reasonable to regard ~~the~~ t. ~~[illegible]~~ thusly coded section of t. corpus as a "chunk" i perhaps regard t. complement of this chunk as another part of t. corpus to try coding as a chunk.

T. idea is pretty much, that in the $\Theta$ code, that one now has ~~some~~ a (perhaps) useful trick to code t. rest of t. corpus. It may be that having $\Theta$ i a hype method of recognizing t. Domain of $\Theta$ (i.e. when $\Theta$ is applicable), makes it natural to want to divide up t. corpus i try to find a separate code for "T. rest of t. corpus".

Hvr., it would seem a bit heuristic to decide that this "rest of t. corpus" was now amenable to a/ completely non-el. Lsrch (within t. CB available)!

9.4.81 TS

[redacted] Just how does this/ relatively ~~non~~-el. [redacted] Lsrch. differ from t. very el. srchs that [redacted] give $\beta^{12}$ ?




This non-el. srch (that eventually yields t. loop) can be regarded as backtracking from $\alpha_{12}$, all t. way back to $\alpha_0$. [redacted] How can we justify TM's backtracking back that far?

One way: he tries backtracking to $\alpha_{11}$ w. no useful results — then back to $\alpha_{10}$, w. still no success ........ Then all t. way back to $\alpha_0$ before a new useful soln. appears.

---

A note on C.B.: For t. $\beta^{12}$ soln. TM knows its cc & its total pc — so he knows its Lcost. This Lcost could then be a reasonable CB to use in any Big, [redacted] less el., (less el. than ($\beta^{12}$)) Lsrch.

---

$\boxed{9.5.81}$ sat [redacted] (12:10PM): T. way Korsybski thot of elzn., was that it was usually possl. to divide up a problem into sub-problems, & that one of them mite seem t. only way to solve t. problem: This ~~would~~ elzn. would, presumably be t. "[redacted] obvious" one.

Anyway: say we divided t. prob. into 4 parts, A, B, C, D. We could then be less el. by considering [redacted] various subsets of these 4 objects.

[redacted] [redacted]

4 [redacted] : 1, 3 subsets

3 : 2, 2 ''

1 : 4 subset ← completely non-el.

1 : 1, 1, 1, 1 subset.

$4 \times 3 = 12$ : 1, 1, 2 subsets (3 ways to break up each set of 3).

} ~~so~~ → possl. $\boxed{21}$ [redacted] subsets!

| | |
|---|---|
| 1 | 1, 1, 1 [redacted] |
| 2 | 1,1 ; 2 |
| 3 | 1 |
| 4 | · 0 |

$\boxed{9.6.81}$ Also, since one can try these in any order, it amounts to 21! difrnt possys. $5.11 \times 10^{19}$ $\approx 5.11 \times 10^{19}$

~~Order is, of course, impt. because of SPA's.~~ No, it doesn't!!

In each division of subsets there are at most 4 subsets, so 4! orders possl. but most subsets are 1, 1, 2 subsets so 3! orderings: so usually only $21 \times \sim 6 \approx \boxed{126}$ ways to try. ($6 = 3!$)

T. idea of 202.04 –.18 ; .22 –.25 is good: It regards coding as [redacted] kind of "Problem" or "Task" [redacted] & thus amenable to t. general methods of SP, for task not soln. T. idea of 202.04 – .18 is of dividing a prob. into sub-probs — of which <u>subgoal</u> construction is one particular [redacted] common method.

Hvr, t. problem of how to divide up a prob. into sub-probs is a "<u>pre-SP</u>" problem.

So we now have t. follg. similar (& sometimes identical) sets of objects:

1) Plans: [redacted]

2) CPMS ≈ PEMS (a PEM, perhaps, need not be a <u>computable</u> proby measure (?))

3) Method(s) of breaking a problem into subprobs { AND &/or OR ) ( serial &/or ∥ probs )

4) SP : which is an optimum approach after 3) has been done.

---

1) a Plan for coding may or may not yield a CPM : It may not converge at all — it may yield neither a semi convergent or a Normed semi convergent proby measure — Tho certain classes of plans for coding <u>always</u> yield CPM's or other well [redacted] behaved measures.

---

Lsrch is a particular plan for solving <u>any</u> coding problem. It <u>always</u> works, but may take too much cc.

Lsrch can also be used for another (very broad) class of non-coding problems

---

In t. general Coding problem, we are ≈ interested in dividing up t. problem so that we can get t. ≈best possl. code [redacted] available within t. CB.

{ In general, this is only ≈ true : we will settle for occasional CB overruns, (But not too much) & often for not such optimal solns.

Anyway, this means that when we divide up t. problem, Lsrch <u>need</u> not be applied to each (or any) of t. parts.

---

Wrt t. $\beta^{12}$ v.s. Loop solns. problem: Perhaps we can confine ourselves to considering Lsrch as being t. <u>only</u> way to solve a coding prob. (rather than dividing up t. subcorpus again).