D 5 79

Number Bases: Non-Integral:

Say we want to use t. base, π. T. representation will be a string of integers $< \pi$, ≥ 0; i.e. 1, 2, 3.

"3 2 1 0 3" means $3 + 0\pi^1 + 1\cdot\pi^2 + 2\cdot\pi^3 + 3\cdot\pi^4$.

"3.2103" means. $3 + 2\cdot\pi^{-1} + 1\cdot\pi^{-2} + 0\cdot\pi^{-3} + 3\cdot\pi^{-3}$

To convert a no.s N;

.08 Find largest power (n) of $\pi \ni \pi^n \leq N$. If it is =, $N = 1\cdot\pi^n$.
If < find t. largest a_{n+1} integer $\ni$ $N > a_{n+1}\pi^{n-1}$, a_{n+1} is t. first "digit" (msd)
Take t. remainder $N - a_{n+1}\pi^{n-1}$ and get t. next msd via .08. Loop until remainder
.11 is zero or is small enuf to satisfy t. approxn. needed.

~~I vaguely feel that this sy~~

Using t. algm. .08, t. conversion is, I think, always unique, if th. base is not an integer. I'm not sure under what conds. (if any), one is assured of a finite conversion time. e.g. it may take an infinite amt. of time to decide if, say π^3 is > or < or = to N, if N is of arby precision. This will happen infreqly, hvr.

The system is inefficient, hvr. To represent an integer N, ~~always~~ usually needs more digits to base π than to base 4 (tho in both cases, only t. digits 0, 1, 2, 3 are usd.

Integers are ~~always~~ usually represented as non-terminating expansions — but not always. ~~e.g. consider $3x^3+2$~~

e.g. $3x^2$ = integer has a soln. for x that is a non-integer > 3.
$3\cdot 3^2 = 27$, $3\cdot 4^2 = 48$ so $3\cdot x^2 = N$ has a soln betw. 3 & 4 if N is betw 27 & 48. or even $3x$ = integer can have non int solns for $x > 3$.

One thing I was thinking of using these non-integral bases for was CBI. By using the shortest dcrn. length (N) as proby index for t. string., we get 2^{-N} for probty if base 2 is usd.

Using other bases is O.K. for number radixes, but how do we use non-integral bases for dcrns of strings?

E.g. using 1.1_{10} as a base, gives us proby values 10% apart.

Negative bases are of some interest. To use base -3;

$2\cdot(-3)^0 + 1\cdot(-3)^1 + 0\cdot(-3)^2 + 2\cdot(-3)^3 =$
$2 - 3 + 0 - 54 = -55$

I don't know if its more or less efficient than other bases. Using a positive base with a ~~negative~~ sign symbol looks like a Mixd base. Arith may be simpler w. a neg. base - (say -2). I think there were some recent papers in Computer, CACM or IE3 comp. trans on neg bases.

D579

Number Bases: Negative.

One apparent advantage of neg bases: ~~For addition~~ when adding 2 nos., the carrys do not propage. This is because a carry is always a borrow from the next higher digit! Woops! if one borrows from zero, this can propgate!

Anyway: This type of arith may have advantages.

~~Another Woops!~~

2.19.80 CODE

coding w. binary defns! This is a more rigorous, exact method of doing wht was dcrbd in I & c II :

~~[illegible]~~ I will dcrb. a method of assigning probys to sequences.

T. codings details, I will not go into, but refer to t. inverse kraft thrm.

~~[illegible]~~ Apriori info. gn! ① No. of symbols in t original (raw) corpus. ≡ **N**

.10 ② Aprip, ~~[illegible]~~ δ, of a ~~[illegible]~~ punctuation symbol indicating a defn, is to be made. (— This is to be chosen by t. "coder": I think it's not a critical choice.)

⟨14 vr., see .16⟩

~~Seq t. orignl alphabet is~~ N .

~~[illegible]~~

.16 → Actually: .10 is probly unnecy: as we shall see: First, to code t. Bern seq. of **N** symbol alphabet: (N known apri).

~~[illegible]~~ Dcrb. t. **CPM** first in prefix code: This is simply a list of the probys of t. N symbols, to adequate accuracy, (actually, ~~any~~ all accuracy levels ~~[illegible]~~ should be usd, and all conceivable sets of pc's for t alphabet symbols ~~[illegible]~~ — ~~[illegible]~~ These give various CPM dcrns — & t. final pc. of t. corpus is t. sum of ⟨pc's of each CPM times pc of corpus wrt that CPM⟩.

[SN] t. coding of t. pure bern. seq. is unclear! what pc. is to be assigned to t. various probtys?

→ To Code t. Bern Seq. in which N, t. ~~[illegible]~~ alphabet size, is known.

N − 1 ~~[illegible]~~ probty params, p_i are needed. $(i = 1/N-1)$ $\left(p_N = 1 - \sum_{i=1}^{N-1} p_i\right)$!.

To express these, we use **R** ≡ 1000 bit accuracy (T. result is indep of R for $R \gg 1$).

If t. sequence in **K** bits long, t. accuracy needed in t. p_i's is $\sim \frac{1}{\sqrt{K}}$.

As a result, of the ~~[illegible]~~ 2^{1000} parallel codes for each possl p_i value.

.25 $\sim \frac{1}{\sqrt{K}}$ of them are about rite — So t. pc of each p_i is $\sim \frac{1}{\sqrt{K}}$

The pc of t. N − 1 params is $\left(\frac{1}{\sqrt{K}}\right)^{N-1}$, and the pc of t. seq. itself w.r.t. this CPM dcrn is $\prod_{i=1}^{N} p_i^{K p_i} = \left(\prod_{i=1}^{N} p_i^{p_i}\right)^K$.

.29 **T.** entire p.c. is $\approx \left(\frac{1}{\sqrt{K}}\right)^{N-1} \cdot \left(\prod_{i=1}^{N} p_i^{p_i}\right)^K$ → 3.30 for exact expressn.

.30 The "≈" is because t. p.c. of each p_i gn. is probly not exactly $\frac{1}{\sqrt{K}}$) — but has to be figured out .... it is t. equivt p.c., obtaind by using all ~~[illegible]~~ 2^R codes for each p_i: so $(2^R)^{N-1}$ difrnt codes.

The pc. is obtaind by some kind of integration (rather than a summation) of codes (assuming uniform aprip density in $\vec{P}$ space ... the "Laplace's Rule" assumption).

which its equivt. to.

.36 The way the coding is done: N and R are given apriori.

The first ~~[illegible]~~ N − 1 sequences of **R** bits each, give t. N − 1 p_i values.

The following seq. is t. code of t. corpus wrt t. ~~[illegible]~~ CPM just dcrbd.

We sum over all $2^{(N-1)R}$ CPMS, and we sum over all dcrns of t.

2.19.80
CODE

corpus w.r.t. each CPM (There will be an ∞ of codes for t. corpus wrt each cpm,
.02 ordinarily .... unless the p_i's are ~~a~~ terminating binary fractions ~~E~~., (I think !))

For ~~a~~ coding using Binary definitions : : If we are initially given N, t. no. of Alphabet symbols $a_1 \ldots a_N$ i D, t. no. of Binary defns usd,

we code t. corpus by writing e.g. (D = 3 in this example)

.10 $\overbrace{a_3 a_2}^{a_{N+1}}\overbrace{a_5}^{a_{N+2}} a_{N+1}\ \overbrace{a_{N+2}\, a_1}^{a_{N+3}}$ T. recoded corpus, w. suitable restrictions.

$\overset{\uparrow}{1}\ \overset{\uparrow}{2}\ 3\ 4\quad 5\ 6$

In positions ~~[illegible]~~ 1 i 2, only a_1 thru a_N are legal.

In positions 3, 4, a_1 thru a_{N+1} are legal

" " 5, 6 ; a_1 " ~~a_{N+2}~~ a_{N+2} " " .

~~In all positions after~~ In Most positions after 6, a_1 thru a_{N+3} are legal.

— except for restrictions: after a_{N+1} has been defnd, a_2 can't follow a_3
" a_{N+2} " " " a_{N+1} " " a_5
" a_{N+3} " " " a_1 " " a_{N+2}.

~~We code t. corpus we then~~ we then rewrite t. corpus using t. defns. i write this ~~at~~ prelimy coded corpus here
. Then we code t. string at .10 as a Bern seq. w. N+D sized alphabet with the restrictions mentiond.

To ~~[illegible]~~ write t. rest of t. CPM, we first write N+D-1 strings of R bits each, (R >> 1) representing the R+D-1 probys involvd. The probty of t. corpus wrt each such assignment of probys in N+D-1 space is computed as t. product of t. probys of t. symbols, taking t. restrictions into account. e.g. if, at a particular position, a_3 occurd, ~~to~~ and a_7 is the only symbol illegal in that position, t. resultant probty usd is

$P_{a_3}\frac{1}{(1-P_{a_7})}$: If a_7 i a_4 were illegal there, $P_{a_3}\frac{1}{(1-P_{a_7}-P_{a_4})}$ would be usd.

I don't know how to go about computing t. accuracies of each of t. probys (i ~~[illegible]~~ t. pc of ~~[illegible]~~ deriving each is ∝ its accuracy)
~~I don't know exac~~ I'm not sure about how to get t. peak values of t. probtys themselves. ~~[illegible]~~

Hvr., t. discn. of pure Bern coding of 1.30 - 2.02 is to be applied to obtain the ~~[illegible]~~ accuracy equivt i t. ~~[illegible]~~ equivt peak values of the P_i's. ⟶ see 3.30 - .40 i 4.01 ff. for further discn, hvr.

There remains t. p.c. of the ~~param~~ coding parameter, D. This has to be coded in a prefix code that is inserted befor t. rest of t. code first derbd.

One interesting way to assign aprip to t. integers $i = 1, 2 \ldots \infty$. $(i = 1/\infty)$ Use ~~Σ~~ $p_i = \frac{1}{i}$ (!). Consider the fact of non-convergence!

Say R is the upper limit of nos. we will consider; Then t. normzn factor is $\approx \frac{1}{\ln R + \gamma}$

Euler's const

Since we are interested in ratios of aprip, this/ exact value of this normzn factor is irrelevant!

— T. ratio of/ aprip D values D_1 & D_2 is simply $\frac{D_1}{D_2}$ (!)

Actually, I can use any aprip I like for D — usually w. t. restriction of convergence: But this $\frac{1}{D}$ thing really looks neat!

In general, it is not of much importance, hvr. We are interested because we want to know how much t. pc of t. corpus is ↑ by/ a particular defn. The main thing that tends to ↓ t. pc. is t. factor equiv to $\frac{1}{\sqrt{K}}$ (1.25). The factor $\frac{D}{D+1}$ is so close to 1 (even if D is only 2), that it is usually of ~~no importance~~ negligible importance wrt. .

Hvr., if $D = 2$ $\frac{D}{D+1} = \frac{2}{3}$: which does mean that one method of coding will be gn. $1\frac{1}{2}$ times as much wt. as another — — this could be of some import. Also, ~~&~~ when $D = 1$ (an impt. case) $\frac{D}{D+1} = \frac{1}{2}$, which is a fair sized factor!

I think that much of t. foregg. stuff is relevant to t. problem of coding w. linear (& non-linear) regressn; determining how many (& which) regn. coefs. to use.

.30 : 1.29 : Instead of $\left(\frac{1}{\sqrt{K}}\right)^{N-1}$, all we want is t. total wt. in $N-1$ dim space of th. p_i coefs.

This is exactly $\underbrace{\int\int\int \cdot \int}_{N\text{ fold}} \prod_{i=1}^{N} \left(p_i^{m_i} dp_i\right)$

Here m_i is t. no. of times a_i occurs; $p_N \equiv 1 - \sum_{i=1}^{N-1} p_i$

I'm not sure this is an N fold $\int$ — it may be only N-1 fold.

anyway the integral ~~[illegible]~~ involve is well known and rather simple in structure. The complicated part involve a ~~£~~ Gamma function — which may be where the approxn $\frac{1}{\sqrt{K}}$ comes in.

Maybe not so easy ... see 4.17!

2·19·80

Code

For t. corresponding factor in "coding w. binary datus",

we have to integrate <u>not</u>, e.g. $p_1^{17} p_2^{5} \cdots p_9^{21} (1-p_1-p_2 \cdots -p_9)^4$

but $p_1^{17} p_2^{5} \cdots p_9^{21} (1-p_1-p_2 \cdots -p_9)^4 \cdot \boxed{\left[(1-p_2)^4 (1-p_5)^3 (1-p_5-p_3)^5\right]^{-1}}$

The new factors occur because of the restrictions in coding at certain positions

<u>Woops!</u> There <u>is</u> a difty! This factor causes divergence! $\int_0^1 \cdots \frac{1}{(1-p_2)} \, dp_2$ diverges.

It <u>may</u> be canceld out by the $\approx (1-p_1-p_2 \cdots -p_9)^4$ term

Also, since we do have t. constraint $\sum_{i=1}^{N} p_i = 1$, if p_2 here got close to 1, then all of the other p_i's would have to be close to zero!

So: This integral ~~is not so normal to do,~~ perhaps is <u>not</u> so simple to do!

.17 Note also that this integral is more diflt than I thot, since ~~we may~~ $\sum_{i=1}^{N} p_i = 1$ must hold over t. region of integration! so we can't just integrate each variable indply. — This modifys t. sugt. of 3.30 —.40 as well!

2·19·80 Code:

Linear (& Non-linear) Regn:

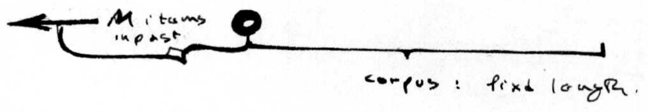
One ~~[illegible]~~ part that was rather messy, was coding t. first ~~#~~ M ~~[illegible]~~ items of t. Corpus (M is t. no. of coeffs to be usd).

.03 I had previously considerd coding each of them optimumly in terms of "known" previous/ items — "known" being "coded thus far".

A much easier way, would be to consider t. first M items as "given free": Then code t. rest of t. seq. w.r.t. that info. We use something like

complexity (x/y) ≡ complexity of x given y.

~~Then~~ For larger M, we have to consider items further into t. past — if we keep t. corpus itself of constant length.



As we ↑ M, we keep corpus constant, but M extends further into t. past.

8583! See more recent Notes on this; e.g. Notes for Harvard talk

→ w 3/83! Trouble is each of these M ~~[illegible]~~ datapts represents an enormous amt. of "info". Giving M+1 rather than M coeffs is prost ratio of prist of 1 data pt. ~~[illegible]~~! which ~~[illegible]~~ very much.

Actually .03 isn't bad. We start w. 1 (or 2 or 3) ~~[illegible]~~ data pts known, whether its 1, 2, or 3 ~~[illegible]~~ will only affect whether we decide to terminate t. search w. only 1, 2 or 3 coeffs (I think) — so it prob'y does not affect t. $\boxed{\text{final apsipd}}$ — but Check on this

→ Ref to [illegible] is!

On discovery PSG discy of ~~[illegible]~~ stochastic Models using H.C. i Entropy estimates ~~is~~ as hill ht.

— Mainly on how large a SSZ is needed to "identify" PSG's

— or what ~~[illegible]~~ predn. accuracy is obtainable w. what SSZ.

Also on t. Efficacy of using simpler grammar models ~~for~~ SSZ, (or even SSZ), & then modifying t. simpler grammars into PSG's (i.e. introducing loops or rules giving potentially infinite langs.

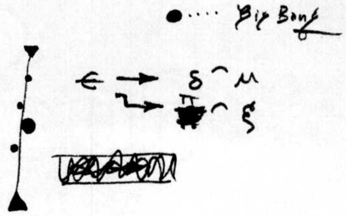
I'm thinking of 3 levels of Grammars: egs.

1) Grammar for a finite lang:

i.e. no. loops, but otherwise a PSG: stochastic choices.

$S \to \epsilon \frown \beta$
$\hookrightarrow \gamma \frown \epsilon$

$\epsilon \to \delta \frown \mu$
$\hookrightarrow \pi \frown \xi$



$\alpha \to a$, $\gamma \to c$
$\beta \to b$, $\delta \to d$; $\mu = e$, $\pi \to f$, $\xi = g$.

2) Finite State langs., w. stochastic choices........

3) stochastic PSGs w. loops being legal.

~~all SSZ are of t. form~~

In ① if one had a large enuf SSZ, one could get t. prby of each s in t. lang. & then try various Grammars to reduce t. no. of continuous params. descbg. t. lang.

Hvr. perhaps t. usual case is that t. lang. is ~~to~~ has too many members for this to be practical, so the loopless PSG ~~to~~ gives a very large ↓ in t. amt. of info in t. grammar.

Actually, I usually think of finding a type 1) model ~~going~~ then going to a type 3) model. — ~~using~~ I haven't really considered that a type 2) model ... being able to model an infinite lang., mite be closer to 3) than 1) is! (Except that the idea of t. "a new method .... 1960" paper was that PSG's are very ≈ to FSG's. & this fact mite be used to implement PSG discy).

Any type 3) Grammar can be written as a (potentially infinite) type 1) grammar!

.37

e.g. type 3)

$S \to \alpha$
$\alpha \to \alpha a$
$\hookrightarrow b$

so all s's are: ba^n.

O1480

PSG discy

2

An infinite type 1 PSG for this:

.03

$S \to \alpha$ | $\alpha_1 \to \alpha_2 a$ | $\alpha_2 \to \alpha_3 a$ . . . . . .

$\alpha_0 \to \alpha_1 a$ | $\to b$ | $\to b$

$\to b$

Its not clear whether this grammar would have less info than t. original corpus. — for any finite corpus, only a finite no. of Grammar rules would be needed. Also, if we allow repetition of a's in t. given corpus, t. corpus would ordinarily, have much more info than t. grammar.

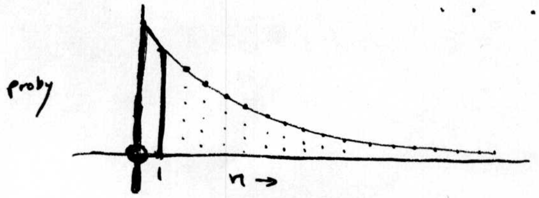
e.g. say t. probty of $\alpha_i \to \alpha_{i+1} a$ was p; $\alpha_i \to b$ was $1-p \equiv q$.

So, if p is close to 1, we will have a mean string length of

$\sim \frac{1}{q}$ (a string of length 1 has probty $q \cdot p^0$

" " " " 2 " " $q\,p^1 = q \cdot p^{2-1}$

. . . . n " " $q\,p^{n-1} = \frac{q}{p} \cdot p^n$



If t. ssz is such that we have 1 string of length $n = n_0$, then we have ~ (~~$\frac{1}{p^{n-1}}$~~)

$\frac{1}{p^1}$ strings of length $n-1$

$\frac{1}{p^2}$ " " " $n-2$, etc.

so, total string length $= \frac{n}{p^0} + \frac{n-1}{p^1} + \frac{n-2}{p^2} \dots\dots \frac{1}{p^{n-1}} + \overset{=0}{\frac{n-n}{p^n}}$.

$= p^{-n_0}\left(p + 2p^2 + 3p^3 \dots\dots n_0 p^{n_0}\right)$

$\Sigma \equiv p^{-n} \sum_{i=1}^{n} i\,p^i = p^{1-n}\left(\sum_{i=1}^{n} i\,p^{i-1}\right) = \frac{d}{dp}\sum_{i=1}^{n} p^i$ $i \equiv n$

$\frac{d}{}\frac{\alpha}{\beta} = \frac{\beta\,d\alpha - \alpha\,d\beta}{\beta^2} = d\left(\alpha \cdot \frac{1}{\beta}\right) = \frac{1}{\beta}\,d\alpha + \alpha\,d\left(\frac{1}{\beta}\right) - \frac{\alpha\,d\beta}{\beta^2}$ | $\frac{d}{dp}$ " $\frac{1-p^{i+1}}{1-p}$

$d\,\frac{1-p^{i+1}}{1-p} = \frac{(1-p)\times(-(i+1))\,p^i - (1-p^{i+1})\cdot(-1)}{(1-p)^2}$ $i \to n$ $i \equiv n$

$= \boxed{\frac{1 - p^n\left(1 + (1-p)n\right)}{(1-p)^2}}$

~~$\frac{-(n+1)p^n}{1-p} + \frac{1+p^{n+1}}{(1-p)^2}$~~

~~$\frac{1}{1-p}\left(-np^n - p^n + \frac{1}{1-p} + \frac{p^{n+1}}{1-p}\right)$~~

~~so $\sim \frac{1}{(1-p)^2}\left(1 + p^{n+1} - p^n\right)$~~

~~say $n \sim \frac{1}{1-p}$~~

proby of string of length n is $q\,p^{n-1}$

.40

so $\Sigma = \frac{1 - p^n\left(1 + (1-p)n\right)}{p^{n-1}\,(1-p)^2}$

PSG discy

The idea here, is that if the longest example string is of length n, $(n \gg 1)$, Then the no. of ~~the~~ choices in t. raw corpus would be very large: for $p = .9$ & $n = 20$, t. no. of choices made is ~ 470 : (from 2.40).

On t. other hand, t. no. of choices ~~m~~ to create a non-loop PSG would be $\approx$ 3n (see 2.03) which would be $\ll$ t. no. of choices in t. raw corpus.

Hvr. this is not t. point! Even using a good grammar, t. no. of choices needed to create t. corpus, would be about t. same as t. no. of symbols in it: i.e. (2.40) So the argt. would have to be very detaild to tell if a loopless PSG ~~[illegible]~~ did, indeed, achieve a compression > say a simple Bernoulli code!

Actually for t. ~~&~~ lang of 1.37 w. $S \to \alpha$; $\alpha \xrightarrow{.9} \alpha a$ $\xrightarrow{.1} b$

t. Bern seq. is not so bad: for a large corpus,

$P_\alpha \sim .9$, $P_\beta \sim .05$, $P_{\text{end symbol}} \sim .05$

Grammar-Grammar

Hvr., t. main idea of t. present discu. is: If one is gn. a looped PSG corpus, is it useful to first find a good (loopless PSG) (i.e. type 1 gramm.) for the corpus .... Then try closing t. loop(s) .... using H.C. & apparent Entropy (≡ Pcost) ~~&~~ as a Gore.

Whether a Bern seq. would give a better Gore is, hvr., Not relevant... because [redacted] → A Bern seq. is not a type 1 Gramm, since it is for an infi. lang. (Here we use a stop symbol as one of t. Bern ~~[illegible]~~ alphabet, so we get > 1 sentence in t. corpus).

(SN) From a large loopless Gramm of a loopd PSG corpus, one can devise a grammar for this (potentially infinite) Grammar, & perhaps from this "Grammar Grammar" find an easy way to close t. loop(s).

Hvr., what has always seemd like a v.g. idea, is to find out how to find t. Gramm of a ~~s~~ stochastic finite state lang. From this use t. correspondence betw. FSG & PSG to devise a method for PSG discy.

There may be known ways to do this for FSG: See Taylor Booth: papers, reviews, book(s).

N22 82 TM: PSG ~~[illegible]~~cy:

.00 First make as good as possl. TREE grammar. This is a loop free CF Grammar. For a given corpus size, there will be a bunch of "Best" Grammars. From one (or more) of these tree grammars, one tries to xfm them into looped grammars. Here, each Tree grammar is regarded as an object that one wants to xfm into a more "compact" (more pc) object — i.e. t. loopd grammar. Once we have t. tree grammar, we don't go back to t. original corpus.

T. only (sort of) backtracking method usd, is to retain several Tree grammars & try to "loopify" each of them, to see which results in t. best "pc" object.

T. reason I think this is a good approach: "Working backwards" Say one had a good loopd grammar for t. corpus: Then one could break t. loops in various ways & get Tree grammars of probably lower pc. Looking at it this way makes it reasonable that .01 ff Would work.

How many diferent ways can on convert a loopd grammar into a tree Grammar? Does one always ↓ pc by doing so? Do t. various alternative Tree grammars all have ~~about~~ t. same pcs?

consider t. loop language $(a)^n$:

Grammr:

$S \to a$

$\quad\to S a$

Some Tree grammars

$S \to S_1$

$\quad\to S_1 S_2$

$\quad\to S_1$

$S_1 \to a$

$S_2 \to S_1 S_1$

$S_3 \to S_2 S_2$

$S_4 \to S_3 S_3$

.35

or

$S \to S_3 S_3$

$S_3 \to S_2 S_2$

$S_2 \to S_1 S_1$

$S_1 \to a$

$\quad\to \Lambda$

How to go from this to this?

or better Tree grammar

$S \to S_3$ [redacted]

$S_1 \to a$

$\quad\to \Lambda$

$S_2 \to S_1 S_1$

$S_3 = S_2 S_2$

$S_1 = a^{0,1}$

$S_2 = a^{0,1,2,3}$

$S_3 = a^{0,1,2,3,4,5,6,7}$

N2392 | TM PSG-Discy

Actually, t. grammars of .35 R (& .35 ~~a~~ — .40 in general) are ~~[illegible]~~ not exactly what one gets from $\underset{TB}{Z}\overset{?}{\widetilde{141}}$ (coding u. defns.). Z141 gives a ~~[illegible]~~ lang. That is a Bernoulli seq.

Hvr., t. methods of Z141 can be applied to ~~[illegible]~~ ordinary langs & t. devising of free Gramars for Rex.