Much missing

Dart & 539, 540, 541, 565

"precidence rules"

constraints on parsing

[redacted] work on methods to try to

solve PSG WOQ.

Part of this *may* be obsolete

thru more recent work on

uniq. parsing CFG's.

Much related to ZTB 141

533.40 spac

.01: 538.40. → A trick used in th. coding of th. dagan. (no-... involving ngm. defs (possibly n≥2 ?): Th. precidance r... coding were such that if e.g. $\alpha \equiv abc$ and $\beta = ...$ and $\gamma \equiv cd$, then abcd would always be coded as ~~a~~ αd rather than ~~a~~ $\beta\gamma$. Th. coding of a corpus was se... and at certain points a choice of coding methods was poss... (e.g. abcd coded as either αd or $\beta\gamma$) — Th. first time th... particular choice occured, it would have to be specified in th. ~~co~~ code instructions for choices — but subsequently, if that same choice ever occured, it would be made th. same way that it was made th. first time, i.e. subsequently γ could never follow β, because such a seq. should have been coded as αd.

I think th. coding pgm of Dart B1300 was, to some extent, capable of dealing with this type of coding. In fact, th. making of γ not being able to follow β, is easily implemented in this pgm.

~~Hm~~ I should probly. make sure that it really works o.k. for this simple ngm - non branched case befor trying to ~~exp~~ genlz. this idea to a full WPSG.

A "trouble" with th. above "precidance rule" idea, is that I would much rather not have sharp restrictions, but rather ↑ th. ~~prob~~ pcost of certain subseqs by ↑ th. no. of difrnt ways to code them, rather than by ↓ th. no. of legal possl. other codings. → 540.25 . 541.35

.25

SN Incidently, an apparently impt. idea that I don't think I noticed in trying to solve ~~a~~ th. lang ~~...~~ $0^n 1^n$. ~~...~~

If we are gn. th. corpus 01, 000111, 00001111

th. best defn. appears to be $\alpha \equiv 10$. This makes it impossl. for 0 to follow 1, — so only α can follow 1, and α occures "infreqly".

~~Ne~~ Next, define $\beta = 01$ — this recodes th. corpus as

$\underbrace{10}_{\alpha}, \underbrace{01}_{\beta}; \beta, 00\beta11, 000\beta111$, ~~...~~ — with 1 cannot follow 0,

and β follows 0 rarely, and α follows 0 or 1 even more rarely —

So this gives th. corpus a rather hy ~~...~~ pcost!

Next, we may want to define ~~...~~ $\gamma \equiv \beta 0$, so 0 cannot follow β

also define $\delta \equiv 1\beta$, so β cannot follow 1. — Tho those last 2 may not ↑ pcost enuf to pay for their defns.

At this point in th. coding, we note that β often appears in th. envt. 0, 1 — so th. old defn $\beta \to 01$ can be expanded to include $\beta \to 0\beta 1$, i.e. $\beta \to \beta, 0\beta 1$.

540.01

F Ap 20, 62   TM8

Dart

(540)

.01: 539.40 : There mite be an intermediate defn. $\gamma \to 0\beta$

I suspect that then th. defns.

$\alpha \to 10$
$\beta \to 01$
$\gamma \to 0\beta 1$

would perhaps not give ~~[illegible]~~ out
~~[illegible]~~ so we mite not do it. While

going from $\alpha \to 10$, $\beta \to 01$ directly to $\alpha \to 10$, $\beta \to 01$, $\beta \to 0\beta 1$
~~mite~~ would be more likely to ↑ prost.

When we have $\beta \to 01$, $\beta \to 0\beta 1$, it would seem that

.10 $\alpha \to 10$ would be unnecy. How would we get rid of it?

One could try to do ~~[illegible]~~ PSG bizey with th. folg. set of xforms:

(1) Defining ngms — for ~~[illegible]~~ $n \geq 2$. e.g. $\beta \Rightarrow abc$

(2) Defining and branching. e.g. $\beta \Rightarrow abc$, $a\beta c$.
~~(3)~~ This includes (automatically) looping

.20 (3) Dropping old defs and parts of defs. e.g. dropping $\alpha \to 10$
or if $\beta \to abc$, $a\beta c$, we may want to drop $\beta \Rightarrow a\beta c$
(tho we couldn't drop $\beta \to$ ~~$a\beta c$~~, abc, since then there would be no way to terminate th. loop!).

Th. Q, then, is whether we can work $0^n 1^n$ and th. ~~[illegible]~~ palindrom. lang with these 3 ~~of~~ h.o.c. xfms a/o modifications of them.

.25 A natural extension of th. "restrictions" in th. non-branched PSG coding to Branched PSG, is th. folg: Say we have a corpus we're coding, and ~~[illegible]~~ we already have a set of defns. ~~[illegible]~~ (including Branched ones). Th. Q is — which parsing shall we use? Well — in parsing with nbPSG, we have a bunch of ~~[illegible]~~ (usually nested) definitions that we use. We look at th. corpus and make a ~~[illegible]~~ subtitution, then in th. resulting new corpus, we make another substitution, etc., untill we can make no more. ~~Then~~ ~~[illegible]~~ At certain pts. in th. substitution process, there will be mutually exclusive subsns. that can be made. The first time a p.i. situation choice occures, and that choice is made, determines ever after that how that choice must be made whenever that situation arrises. Everytime such a choice is first made, ~~[illegible]~~ we obtain a new "precidence rule" saying that a certain symbol cannot follow a certain other symbol. See 565.01 for objection

541.01

FAp 20,62

~~Dart~~

TMS

541

.01: 540.40: Now, in th. case of th. PSG with branches, [parsing using] ... exactly same thing — except that some of th. subst... are singular — e.g. if $\beta \rightarrow ab$, $a\beta$ is a "rule", then either ab or $a\beta$ can be rewritten as β. This is th. only difference betw. th. branched and non-branch PSG. It results in certain constraints that later ↑ prost by making certain substitutions illegal.

I don't immediately see how this constraint type should cause more diffy in th. branched, than in th. non-branched PSG.

Th. trouble is, I haven't yet done ~~any~~ much work with th. nbPSG with n-gm subsns for n > 2. I think there is a lot writ around Dart β1300# (perhaps following) about th. diffys involved in "precidence rules" etc.

If things get too bad, I can revert to PSG's (both branched and unbranched) in which only binary substitutions are used — i.e. $\alpha \rightarrow ab, ac$

Whether triple branches can be allowed (like $\alpha \rightarrow ab, cd$, ...) w.o. causing trouble — I don't know. Certainly ~~tripl~~ any triple branch can be replaced by ~~a~~ / 2 ~~binary~~ double branches if neccy. — but this will probably cost more

---

In discovering PSG's one can reparse / each time a new defn. of xfrm is tried, and use difrnt precidence rules on each reparsing if neccy. Also one can try for optimum parsing with a fixed set of rules.

Look at th. stuff on precidence rule diffys around Dart β 1300, to see if they apply to th. present "soln." to this problem.

.35 : $39.25 : I still would like a more satisfactory way to encode th. fact that b hardly ever follows a. As it is now, th. way I do it is arrange so that ab can be coded only as α, and not as ab. — then we find that α has a low freq.

→ 565.01 spac.

M May 2■, 62 TMJ

Dart 565

541.40 spec

.01: : → Re: These "constraints on parsing": even in PSG's, I don't see how th. constraints of 540.25 –.40 w[illegible] one to define $\alpha \equiv AB$, and then, (tho AB never occur[illegible] th. corpus) get th. constraint that B can't follow A (except thru symbol α, which has hy~~[illegible]~~ rcost).

Also, in th. non-branched PSG, once one has made correct parsing choice, th. future choices of that type should always be made that way, since there is a ~~maximal se~~ set of parsing rules that gives maximal pcost. for branched PSG's, ~~th. parsing~~ tho there may be a set of maximal ~~pc~~ pcost parsing rules, I don't think th. rules would take as simple a form as just precidence rules — (tho I'm not so sure of this, either).

It is clear that I'll have to go into this in some detail befor I have any good relyable ideas on this. — But anyway, (1) th. nacty of "constraints" in PSG coding seems clear, and (2) I must be sure that ~~B~~ th. constraints in a branched PSG will do at least as well as th. constraints I used in binary defn, non-branched PSG coding. (i.e. ZTB (41)).

γ 550, 551

Some good ideas on how to make a Mt. Carlo TM to solve probs. in a suitable lang.

Ap 14, 62

TMJ

Dart

550

533.29 spec

.01 : 549.40 ; Some time ago, I had th. idea that if, in some sense, I ha[d] (I had a "complete")? lang. for derbg. ops., that I would need only very simple fixed heurs. [if] I used a suitable and long enuf tng. seq.

As an example, say I had th. Lisp system for derbg. ops. so that by using random trial combs. of ops., I could eventually solve any problem. Th. soln. time would be much ↓, if I used some simple heur, like Barn seq. coding, to make trial strings.

However, using th. idea of 536.30, of having "erasure" symbols, and using a suitable tng seq., I could do far better.

Another trick, is one that uses an essentially time-varying operator. This means that if th. ~~oper~~ string α (which derbs an op.) solves everything up to now, and one is gn. a new problem that α doesn't solve, — then first find an op. β, that solves just th. new problem. Get trial β's using th. statistics of symbols in α (this minimizes th. pcost of concating β onto α). Then, betw. α and β insert a symbol seq. telling when (i.e. th. no. of th. problem) for which β must be used, rather than α. Next try to find a recognition op. that can tell when β should be used — ∋ this op pcosts < this symbol seq.

This (after 77 isn't a new idea, and was considered, I think, among other places, as part of th. "ob-op" technology.

It is my present impression that I could implement this, to some extent, rite now. Th. idea, is that I write this tng. seq., and at each pt. in th. seq., I write a soln. that satisfies it up to that point. Now ... my sequencial set of solutions can be written historically as a string of ops, followed by additions, revisions, deletions, etc. — i.e. every point in th. soln. seq. can be represented as an historical listing of th. sequence of additions and revisions that were required to arrive at that point.

We will also have to give reasons ~~for~~ why ~~each~~ each of th. operations used, was a "reasonable one" (i.e. by pcost). This involves

551.01

Dart (551)

-01 :55040: heurs — but they, too, can be coded as part of th. seq.

e.g. Normally, we will use th. freq. of symbols as a ~~cl~~ clue [illegible] Mt. C. trials — However, if any other regularities are noted in th. soln. seq., a dcrn. of these regularities should be written down, and made part of th. dcrn. of th. soln.. Any such bona-fide regularity will ↑ pcost of th. existing coded. Also, any such regularity should be used in making future Mt. C. trials — since any new trial will have hyar ~~[illegible]~~ incrimental pcost if it conforms to th. statistics (i.e. regularities) of th. past.

A perhaps related idea: We have this code of th. soln. seq. (This code is th. history of revisions, etc.), and at first we have a ~~[illegible]~~ Mt. C. Stoch Gramm that tries to create it — ~~[illegible]~~ using at first, only ~~[illegible]~~ symbol freqs., then, perhaps, ngm freqs. Later, it tries other revisions of its stoch gramm, in order to try to ↑ th. pcost of th. ~~[illegible]~~ soln. seq. It could use WPSG, or any other kinds of constraints.

What I should do, is to list various sorts of regularities that I actually notice, and see how easy they are to incorporate into th. constraints of a creative Mt. C. gramm.

Another good old idea (of uncertain validity) is that every heur can be viewed as a reordering of trials, and ∴ corresps. to a better assignment of proby to th. trials — or a ~~[illegible]~~ ↑ in post of th. code of all past data that led to this heur. Anyway, if each heur is suitabley used in coding, then ~~[illegible]~~ selecting all trials in order of ~~[illegible]~~ pcost will be ~~a~~ th. absolutely best ~~[illegible]~~ (in only th. sense of min. no. of trials) method of search. Also, it will always yeild th. "most probable" ~~[illegible]~~ soln. — i.e. th. one that is most likely to extrapolate properly.