SETL Newsletter 91 December 7, 1972

A 'grammarless' parse and a related procedure for retrieval by similarity. J. Schwartz

1. Description of algorithm.

This newsletter will outline a procedure which, given an arbitrary string of characters, will assign it one or more parse trees. The procedure resembles that used in the 'nodal span' type of parse. However, it uses no grammar, but instead is steered by the statistical properties of the string being parsed. It is presented as a contribution to the well known problem of learning the grammar of a language from samples of the language. An experiment to check whether the program to be described is capable of learning the tokens of English from samples of English text would be interesting.

Overall, our scheme is as follows. A text consisting of a string of characters is given. It is first reduced by the elimination of all characters which follow three immediately preceding identical characters. (Principle of 'fatigue').

The process which now follows makes use of a 'maximum number of allowed divisions' parameter md, and of a 'threshold' parameter th.

Scan begins. During scan, a 'known words' dictionary is maintained, as is a 'word pairs encountered' collection. The dictionary is maintained as two differently arranged copies, for a reason which will be made plain below. Items in the dictionary bear unique numbers. They may be thought of as having the form

(1) <wordnumber, string(representing word),

divisions(of word into smaller words)>

Since however some of the 'words' generated in the process to be described will actually be whole sentences, paragraphs, or even pages-long substrings of an initially given string, we will prefer to represent the 'string' component of (1) by two integers, the first a starting character position, the second the string's length. Thus instead of (1), our dictionary entries actually have the form

(2) <wordnumber, starting character, length, divisions> .

In (2), 'divisions' is a set (of not more than md elements); each of the elements of this set is a pair of words; each such pair indicates one way in which the word represented by 2 can be divided into a left and a right-hand part. The form of such a division-pair is specifically

(3) $<\text{wordno}_\ell \ , \ \text{wordno}_r>$

Here wordno_ℓ and wordno_r are the number of the left- and right-hand words into which the word (2) may be divided. The items in the 'word pairs encountered' collection have the following form.

(4) $<\text{wordno}_\ell \ , \ \text{wordno}_r \ , \ \text{count}>$

In (4), wordno_ℓ and wordno_r are wordnumbers representing a pair of overlapping words encountered in the input stream; count represents the number of times this word has been encountered.

Whenever a pair s of successive characters is encountered in the input text, either an item like (4) but of the special form

(5) $<s,0,1>$

is built, or the count of an existing item $<s,0,n>$ is incremented. At the point during scan at which the j-th symbol x of the input is being scanned, the following operation is carried out. The longest word w_ℓ terminating at x_j and the longest word w_r beginning at x_j are formed. The count associated with the pair (w_ℓ,w_r) is incremented (or initialized to 1, if (w_ℓ,w_r) is a pair not previously encountered). If this count exceeds the threshold th, then the pair (w_ℓ,w_r) is placed in the division set of the concatenated $w_\ell+w_r$; this word being issued a wordnumber and added to the known word dictionary if necessary. However, we never add more than md elements to the division list of any word in the dictionary.

The word dictionary is maintained in two copies arranged differently. The first is indexed by the two initial characters of words, individual words with these initial characters then following in a list; the second is similarly indexed by the two final characters of words.

As scan proceeds, longer and longer words will be formed. A given source text is scanned repeatedly until a word equal in length to the whole text develops. The recursive pattern of divisions of this word represents the parse of the input string. The words relevant to this parse (i.e., occurring as such a division) are the words 'learned' by scanning the string.

2. Text of algorithm

We now use SETL to represent the procedure just described. (The following code is complicated somewhat by the desire to have a code which will transpose readily to an efficient BALM program.)

```
/* eliminate repetitive characters */
revinput = input(1:3) +
     [+: 3<n≤#input] if #{input(n-k),0≤k≤3} eq 1
                              then nulc else input(n);
/* plausible values for controlling parameters */
     md = 3;  th = 4;
/* initialize dictionaries, etc. */
ℓwdict = nℓ; rwdict = nℓ; wordsenc = nℓ;  wnct = 1;
/* loop repetitively over input, forming longer and
     longer words */
noparseyet = t; (while noparseyet)
n = 1; (while n ℓt # revinput doing n = n+1;)
nowc =  revinput(n:2);
if wordsenc(nowc,0) is count eq Ω then
     wordsenc(nowc,0) = 1;
else if count ℓe th-1 then
     wordsenc(nowc,0) =  count+1;  end if;
```

```
if count eq th-1 then
/* enter character pair into word dictionary */
ℓwdict(nowc) = ℓwdict(nowc) orm nℓ
     with < wnct,n,2,nℓ> is newwd ;
rwdict(nowc) = rwdict(nowc) orm nℓ with newwd;
wnct = wnct + 1; end if;
/* find longest word terminating here
     and longest word starting here */
mℓ = 1; (while if mℓ ge n then f else ∃ tℓ ∈ (ℓwdict(nowc)orm nℓ)
     |revinput(tℓ(2):tℓ(3)) eq revinput(n-mℓ; mℓ+1))
                                         mℓ=mℓ+1; tlf = tℓ; end while if;
mr = 1;  nowcr = if n+mr gt #revinput then Ω
    else revinput(n: 2);
 (while if n+mr gt #revinput then f else
   ∃tr ∈ (rwdict(nowcr) orm nℓ) |(revinput(tr(2):tr(3))
     eq revinput(n: mr+1))  mr = mr+1; trf = tr; end while if;
/* pair operation only if left and right words
     contain more than one character */
if mℓ ℓe 1 or mr ℓe 1 then continue while n;;
/* otherwise do pair-count operation */
wℓ = tℓf(1); wr = trf(1);  /* wl and wr are word numbers */
if wordsenc(wℓ,wr) is count eq Ω then
     wordsenc(wℓ,wr) = 1;
else if count ℓe th-1 then wordsenc(wℓ,wr) = count + 1;
     end if;
if count eq th-1 then
/* enter concatenated word into word dictionary */
/* first determine its start and end */
start=n-mℓ+1; wend=n+mr-1; length = mr + mℓ - 1;
     star2c = revinput (start:2); end2c = revinput(wend-1:2);
/* now put in both dictionaries */
ℓwdict(end2c) = putinw(ℓwdict(end2c) orm nℓ, start,length,wℓ,wr);
rwdict(star2c) =putinv(rwdict(star2c)orm nℓ, start,length,wℓ,wr);
/* check to see if have complete parse */
if length eq #revinput then /* parse is complete */
     noparseyet = f; quit while n; end if length;
end while n;
```

```
end while noparseyet;
/* now remove from the word dictionary all 'irrelevant'
     words, i.e., those which do not arise from the
     longest word by a process of division */
relwords = nℓ; topword = <revinput(1:2), wnct-1>;
divide (topword,t);
stop;

/* here follows the putinw procedure */
definef putinv(dictlist, start, length, wℓ, wr);
if not ∃x ∈ dictlist|revinput(x(2):x(3)) eq revinput(start:length)
     then /* insert into dictionary */
     newelt = <wnct, start, length,{ <wℓ,wr>}>;
     wnct = wnct+1;
     return newelt; end if;
/* otherwise item is present. make addition to division set */
return <x(1),x(2),x(3),
     if (#x(4)) eq md then x(4) else x(4) with <wℓ,wr>>;
end putinw;

/* here follows the recursive 'divide' procedure */
define divide(word, havefirst);
word in relwords; <chars,wno> = word;
<start,len,divset> = if havefirst then rwdict(chars)(wno)
     else lwdict(chars)(wno);
if len eq 2 then return;;
/* otherwise must divide recursively */
schars = revinput(start:2);
ℓchars = revinput(start + len-2:2);
(∀pair ∈ divset) divide(<schars,pair(1)>,t);
     divide(<lchars,pair(2)>,f);  end ∀pair;
return; end;
```

3. A similarity primitive for an artificial intelligence oriented extension of SETL.

A defect in the procedure presented above is that it insists on identity rather than similarity in the learning process. A wider net is cast if this restriction is relaxed: this will allow the program to learn even in the presence of input containing a degree of 'noise'.

We will now describe a primitive of 'pattern similarity' type which a hunch and a small amount of experimentation indicate might be useful. In heuristic terms, the primitive compares each one of a prestored 'dictionary' of SETL objects $x_1,\ldots,x_n$ to another such object y, and forms what is essentially the set of all x_j similar to some subpart of y. Similarity is here taken in the following heuristic sense: a set of 0-1 valued 'features' is formed for each of the items x_j and for the item y. These features are formed by applying some collection $f_1 \ldots f_k$ of numerical-valued functions to the x_j and to y, where all the functions f_j have (i.e., are forced to have) values in some prespecified range $1 \le n \le nmax$. The set of features associated with each x is then $\{f_i(x), 1 \le i \le k\}$. Clearly this set may also be represented by the 'feature-bit' string obtained as follows:

```
bstring = nmax * 0b;  (1 ≤ ∀i ≤ k) bstring(f_i(x)) = 1b;;
```

We say that y selects x_j as similar to a subpart if more than half the features of x_j are present in y. The primitive we propose is that which, given $x = \{x_1,\ldots,x_n\}$ and y, forms the set $\{x_{i_1},\ldots,x_{i_m}\}$ of all elements of x which, in this sense, are similar to a subpart of y.

This operation is of course easily expressed in SETL. We suggest it as a primitive since, if it is found useful and extensively used, its efficiency would be important.

An experiment testing the utility of this primitive will be reported on below. We shall see also that the proposed similarity primitive has interesting relationships with the grammarless parse described in the preceding section.

A detailed SETL specification of the proposed similarity primitive is as follows.

```
definef symbyfeats(a,b,setoffeats);
/* a and b are objects, setoffeats is a set of integer-
   valued mappings used to form 'features' in the sense
   explained above */
/* the boolean value returned by this function will have
   the value t if a is found to be similar to a substring
   of b */
afeats = {f(a), f ∈ setoffeats};
bfeats = {f(b), f ∈ setoffeats};
return (#(afeats * bfeats)) gt (#afeats)/2;
end symbyfeats;
```

In situations in which the collection *setoffeats* of maps is understood, we may write this same primitive, with suppression of its third argument, as *simtosub(a,b)*. Other closely related notions of similarity, easily defined in terms of the primitives, will also be found useful in what follows.

If primitives of the type envisaged turn out to be widely useful, their optimization may be an important problem. Note that a trivial algorithm allows all those items in a collection of N items which are individually similar to the members of an input stream of K items to be found in NK operations. The question as to how much this trivial upper bound can be improved deserves to be studied.

Note also that in dealing with serial input (input strings or tokens) it is reasonable to form features in the following way. Take all pairs of successive characters present in a string. Each such pair may be regarded as a pair of bytes determining a (12 or 16 bit) integer. Reduce this modulo some constant equal to a small multiple of machine word length. The resulting residue is a feature.

This procedure, tested experimentally against a collection of about 2000 tokens, seems to perform well, i.e., to retrieve a reasonable number of tokens similar in an intuitive sense to a given input, rather than an over-large number of tokens most of which have no intuitive similarity to the input. The following computer output will allow the reader to make his own judgement of this heuristic issue.

Test 1. <u>Retrieval by similarity from a group of 2000 tokens used in a computer program</u>.

```
   17 TOKNS SIMILAR TO MINHTSIZE  SIMILAR TOKENS ARE
EBLKSIZE   BLKSIZE    MINUSSIGN  HTSIZE     MIN        ESETSIZE
MINHTSZLOG MSG        SIZE       DIMINIS    MINUS      UWBITSIZE
    2 TOKNS SIMILAR TO RCALL      SIMILAR TOKENS ARE
CALL       REAL
   20 TOKNS SIMILAR TO 00047      SIMILAR TOKENS ARE
0000000000 00004      00007      00017      0000000000 0000000000
00041      00042      00043      00044      00045      00046
   24 TOKNS SIMILAR TO 00048      SIMILAR TOKENS ARE
0000000000 00004      00008      00014      00018      0000000000
00038      0004000000 00040      00041      00042      00043
   19 TOKNS SIMILAR TO 00049      SIMILAR TOKENS ARE
0000000000 00004      00009      00019      0000000000 0000000000
00041      00042      00043      00044      00045      00046
    3 TOKNS SIMILAR TO MASKN2     SIMILAR TOKENS ARE
A2         N2         MASK
    5 TOKNS SIMILAR TO ERRIMPL    SIMILAR TOKENS ARE
ERRIMP     ≠ERRIMPL C ERRVAL     ERRMIX     ERRCRI
   15 TOKNS SIMILAR TO ENDIFLABEL SIMILAR TOKENS ARE
ENDIF1LABL END        THEN5LABEL ENDIF      ENDIF5LABL LABEL
ENDIF2     ENDIF1     ENDIF3LABL ELSELABEL  ENDIF2LABL
   19 TOKNS SIMILAR TO CASESCHAR  SIMILAR TOKENS ARE
CHAC       MAXLSCHAR  CHAR       CS         CASETYP    SCSCHAR
CHARSUB    CASESLABEL SCHAR      CASEREAL   CASECHAR   LCHAR
    1 TOKNS SIMILAR TO ALXOF      SIMILAR TOKENS ARE
OF
    6 TOKNS SIMILAR TO ≠CNEBITSS≠ SIMILAR TOKENS ARE
BITS       NBITS      EUWBITS    ≠ NOT YET  ENUMBITS   ONEBITS
    9 TOKNS SIMILAR TO WHILE1     SIMILAR TOKENS ARE
WHILE1LOOP WHILE2     WHILE3     WHILE4     ENDWHILE1  WHILE3STRT
   12 TOKNS SIMILAR TO WHILE2     SIMILAR TOKENS ARE
WHILE1     WHILE3     WHILE4     WHILE2STRT ENDWHILE2  WHILE3STRT
WHILE2LOOP CONTWHILE2
   10 TOKNS SIMILAR TO WHILE3     SIMILAR TOKENS ARE
3          WHILE1     WHILE2     WHILE4     ENDWHILE3  FILE
```

[continued]

```
   14 TOKNS SIMILAR TO WHILE4     SIMILAR TOKENS ARE
WHILE1LOOP WHILE1     WHILE2     WHILE3     ENDWHILE4  WHILE3STRT
WHILE      WHILE3LOOP WHILE2LOOP WHILE1STRT
   13 TOKNS SIMILAR TO ≠REAL CONV SIMILAR TOKENS ARE
≠A≠        REALVAL    ≠TUPL CONV ≠BOOL CONV ≠TOTAL CP  ≠TOTAL CP
≠REAL1.0$≠ ≠LABL CONV REAL
    9 TOKNS SIMILAR TO ARGOK      SIMILAR TOKENS ARE
ARG1       ARG2       ARG3       AOK        ARG1P      ARG2P
    8 TOKNS SIMILAR TO ERRIMP     SIMILAR TOKENS ARE
ERRIMPL    ERRTYP     TEMP       MTEMP      DIMSIMP    ERRMIX
    4 TOKNS SIMILAR TO ARBTUP     SIMILAR TOKENS ARE
ARG1P      ARB        ARG2P      TUP
   10 TOKNS SIMILAR TO SETSETS    SIMILAR TOKENS ARE
S          SSET       SET        TEST       SETA       SETB
   17 TOKNS SIMILAR TO 00050      SIMILAR TOKENS ARE
0000000000 0          00005      0000000000 0000000000 00035
00055      00056      00057      00058      5000       500
   13 TOKNS SIMILAR TO WHILE2STRT SIMILAR TOKENS ARE
WHILE1     WHILE2     WHILE3     WHILE4     WHILE3STRT QUITWHILE?
WHILE2ENDL WHILE2LOOP WHILE1STRT
    4 TOKNS SIMILAR TO CHARBLK    SIMILAR TOKENS ARE
CHAC       CHAR       BACK       CHARBST
   19 TOKNS SIMILAR TO 00051      SIMILAR TOKENS ARE
0010000000 0000000000 00001      00005      00011      00015
00041      00050      00052      00053      00054      00055
```

Test 2. <u>Retrieval by similarity from a group of 1000 words in English (Pharmacological text)</u>.

```
    6 TOKNS SIMILAR TO FIBRES     SIMILAR TOKENS ARE
RES        FI         FIGURE     MESS       FILMS      FIBRE
   17 TOKNS SIMILAR TO INHIBITING SIMILAR TOKENS ARE
INHIBITOR  INHIBINOT  THINGS     INHIBITORY IN         IT
INITIA     INHIBITS   ACTING     INHIBITEDI INHIBITED  INHIBITION
    1 TOKNS SIMILAR TO IMPLY      SIMILAR TOKENS ARE
SIMPLY
    9 TOKNS SIMILAR TO SIMPLE     SIMILAR TOKENS ARE
SICE       SIDE       TIME       LE         SITE       SE
   16 TOKNS SIMILAR TO SUBMAXILLA SIMILAR TOKENS ARE
SAS        SALIVARY   WILL       WALL       SAME       GLA
SU4        GRAUS      SUPRAMAXIM USUALLY    CAUSE      SUBR
   11 TOKNS SIMILAR TO WILBRANDT  SIMILAR TOKENS ARE
HAD        AD         WHAT       WILBRHELAT HAND       THEAND
TOAD
   21 TOKNS SIMILAR TO CORTISOL   SIMILAR TOKENS ARE
C          I          CI         CL         CO         CONTROLLED
CORTISONE  CLYCOSIDES CARCCSIDES COMPOSITIO ISOMETRIC  CRITICAL
   18 TOKNS SIMILAR TO OCCURRING  SIMILAR TOKENS ARE
ACCORDING  GIVING     RISING     SINING     RINC       OC
OCCURRED   MEASURING  COOLING    MORCING    RINCS      OCCURS
```

[continued]

```
    16 TOKNS SIMILAR TO POTASMS    SIMILAR TOKENS ARE
WAS        AS         SAS        POTASSI    POTASSIUM  MA
GAMMA      POTASS     PLASMA     POTASSIUMM POTASSIUMT POTASSIUOR
     3 TOKNS SIMILAR TO KNOWN      SIMILAR TOKENS ARE
ON         NON        WN
    11 TOKNS SIMILAR TO BEING      SIMILAR TOKENS ARE
BE         GIVING     SINING     BEATING    BIG        RING
BELIEVING
    19 TOKNS SIMILAR TO SECONDS    SIMILAR TOKENS ARE
S          SECR       ENDS       SECTION    MESS       SE
SECONDARY  SLICES     SECOND     KINDS      SOURCES    SECTIONS
    26 TOKNS SIMILAR TO ESTABLISHE SIMILAR TOKENS ARE
THES       EHRLIES    IS         CIS        ANALYSE    LIST
OESTRADIOL DIMINISHED SLICES     SITES      SIDES      DIGITALIS
     2 TOKNS SIMILAR TO BLOOD      SIMILAR TOKENS ARE
DO         GOOD
    33 TOKNS SIMILAR TO INJECTION  SIMILAR TOKENS ARE
WITHIN     INACTIVATE INACTIVATI ATION      SECTION    ADDITION
IONS       IBITION    ILATION    DION       ION        ON
    14 TOKNS SIMILAR TO AREENT     SIMILAR TOKENS ARE
AT         ARED       AHER       APART      BENT       ARE
DIFFERENT  PREVENT    PRESENT    ABSENT
     4 TOKNS SIMILAR TO .RIMENTS   SIMILAR TOKENS ARE
MESS       SS         INTRIGUINE SYSTEM
     8 TOKNS SIMILAR TO SHOWE      SIMILAR TOKENS ARE
SHOWED     SHOW       WERE       SE         WE         SHOWEDABAI
    21 TOKNS SIMILAR TO CHARLES    SIMILAR TOKENS ARE
CAR        THES       CA         CHARGE     CHANGE     CHE
CARE       CASE       GRAUS      REHOSTS    CLEARLY    CLEAR
     8 TOKNS SIMILAR TO WITGHT     SIMILAR TOKENS ARE
THT        HT         IT         WITHOUT    WITH       WITM
     6 TOKNS SIMILAR TO SHOWN      SIMILAR TOKENS ARE
SHOWED     SHIN       SHOW       WN         SHOWE      SHOWS
    38 TOKNS SIMILAR TO OBTAINED   SIMILAR TOKENS ARE
AD         DETAILED   TAENIA     DETERMINED WED        NEED
ARED       TED        ED         INITIATED  SAID       SATURATED
    10 TOKNS SIMILAR TO SHOWS      SIMILAR TOKENS ARE
S          SHAW       WAS        SHOWED     SHOW       GHOSTS
     1 TOKNS SIMILAR TO LURE OF TH SIMILAR  OKENS ARE
```

Note in connection with Test 2 that the test program used happens to break words occurring at line end into parts, leading to the presence in our listing of various word fragments.

4. Generalizations. Parsing against an initial dictionary of words. Parsing indefinite input. Combinability of parse processes.

A crucial point concerning the procedure presented in Sec.2 is that it will tend to accumulate composites of composites just as rapidly as composites of single elements. Thus the parse tree which it grows tends toward balance rather than toward a highly unbalanced structure. The relevant words which survive at the end of the parse will be simple and composite patterns of common occurrence in the input.

This procedure is of course presented as a model for learning, applicable in case of serial input. Thus it might be applicable to the learning of phonemes, syllables, words, and phrases. However, the manner in which generalizations and abstractions are formed still remains unclear, as does the analysis of visual inputs, which must depend on principles of proximity other than the principle of temporal proximity which may be sufficient for the discussion of serial inputs. On this last point, however, the 1965 experiments of Hubel and Wiesel are suggestive.

Note that the collection of items learned by the program will depend not only on the mass of text presented to it, but also on the order of items in this text. For items to be learned easily, they should occur frequently in highly repetitive text, or, more generally, occur with fair frequency in text in which they are separated by items already known. This suggests a series of 'graded readers' as being ideal for the training of programs of the type presented above. Alternatively (for computers) the word dictionaries used in the preceding algorithm can be initialized. In dealing with a purely associative, slowly reacting organic memory this last is unfortunately not possible, but suitably concentrated repetition (with enough variation to prevent repeated-single-symbol fatigue from cancelling the input stream) might have similar effects. Note however that the desirability of

We shall now indicate how the algorithm of section 2 can be modified to allow the principle of similarity described in section 3 to replace that of identity. The necessary similarity primitive will be represented by a boolean-valued function simtosub(a,b) which has the value t if a is similar to a substring of b. The data-structures used in the modified algorithm are exactly like those used in the unmodified algorithm. Most of the code remains the same in both cases. However, the section of code from the comment

```
/* find longest word terminating here and longest
     word s_arting here */
```

to the comment

```
/* pair operation only if left and right words
     contain more than one character */
```

is replaced by code which reads as follows:

```
mℓ=1; (while if m  ge n then f else
     ∃tℓ ∈ ℓwdict(nowc) orm nℓ |
     <tℓ(2),tℓ(3)> islike <n-mℓ,mℓ+1>)
     mℓ= mℓ+1; tlf = tℓ; end while;
mr = 1; nowcr = if(n+mr) gt #revinput then Ω
     else revinput(n: 2);
(while if (n+mr) gt #revinput then f else
     ∃tr ∈ rwdict(nowc) orm nℓ |
     <tr(2),tr(3)> islike <n,mr+1>)
     mr = mr+1; trf=tr; end while;
```

The binary boolean operator islike will be defined in terms of the *simtosub* primitive immediately below. Essentially, α islike β will be true if each sp-symbol-long subsection of the string β is similar to a subpart of a similarly placed but slightly longer subsection of α. Here, *sp* is some measure of 'attention span', which in our intended application may be taken to be approximately 5 characters.

```
definef pairl  islike pair2;
/* the input string revinput is assumed to be transmitted
     globally; sp is a constant having the significance just
     explained */
     <startl,lenl> = pairl; <start2,len2> = pair2;
     /* strings of radically different length are rejected */
if(abs(lenl-len2))   gt sp then return f;;
return l ≤ ∀n ≤ lenl |
     simtosub(partof(revinput,startl+n,sp),
     partof(revinput, start2-sp+n, 3*sp));
end islike;
```

The substring-extraction function partof used in this routine has the following definition.

```
definef partof(string,start,len);
realstart = start max 0;
reallen = #string - start + l min len;
return string(realstart: reallen);
end partof;
```

The principle of retrieval by feature similarity embodied in the modified algorithm just presented has important properties of stability which allow the parsing procedures we have considered to be generalizedd in quite significant ways. As already noted,   a reasonable collection of features to use in handling 'serial' inputs may generally be derived by dividing the input into local 'elements' or 'characters' in some suitable way and then collecting pairs of adjacent characters. These pairs, hashed, may be taken as features. We establish an important property of this method of feature formation by considering the case in which the successive characters of an input text are not known with perfect precision, i.e. in which it can only be asserted that the character in the j-th position is one of some set s_j of possible characters. If in such a situation all possible pairs of adjacent characters are collected and used to form features, the number of features present will be multiplied by the square of the average number of elements in the set s_j , rather than by any higher power of this number. Thus the 'blurring' occasioned by the indefiniteness of the characters in the input stream has relatively limited effects, and, provided that the collection of features formed was scattered into a reasonably large range, will not necessarily lead parse processes of the above type to catastrophically indefinite results.

An algorithm for parsing indefinite input can in fact be formally identical to the above-presented second version of our 'grammarless parse' algorithm. It is only necessary to note that the *simtosub* primitive invoked in this algorithm can apply with little change to a pair of sequences, each component of which is a set of characters rather than a single character.

Note that the retrieval/parsing primitives described above can be used to convert a partially indefinite input stream to a partially indefinite output stream. This can be accomplished as follows: the input stream is viewed on each successive input cycle through an sp-character wide 'window' (where sp is an 'attention span' parameter). On each cycle of input, pairs of adjacent characters are hashed to generate features and the set of features thus generated are applied to a dictionary of 'known symbol combinations', leading to the retrieval of all items judged to be similar to a subpart of the input stream. The successive states of this varying collection of dictionary items defines the output stream corresponding to the received input stream. A general rule something like the following could be used to define the features present in this output stream. the output stream is viewed cycle by cycle, through an sp-cycle wide window. Each pair of dictionary items a,b present in this span of input is used to form a feature, provided that a is not similar to a subpart of b, or vice-versa. Features are formed as follows: the word dictionary index of a and the word dictionary index of b are hashed together to produce an integer in some appropriately restricted range; this integer defines the required feature.

Since transformations like that just described produce an output stream of features from an input stream, such transformations can be compounded. This observation will be elaborated upon below.

The procedures for parsing indefinite input outlined above also apply to situations in which it is the order rather than the identity of successive input symbols that is in question. Pushed to the limit in which order becomes *totally* indefinite, they become procedures for the formation of associations between unordered but simultaneously occurring items.

In the presence of a substantial collection of pre-learned words and phrases, and taken in connection with the immediately preceding remarks on the storage of sequences, these procedures suggest a system for the imposition of an order on an initially unordered collection S. A mechanism like the following would have the desired effect: form all sets of pairs of items from S, and use this for the formation of a collection F of features. From a dictionary containing not only the items in S but also storing element pairs and triples, perform a retrieval based upon F. If an initial element is designated, a sequential 'cueing' process, which we shall now explain, defines a sequence. We call this the sequence *defined by* or *retrieved by* the initially given set S of items.

If one considers a purely associative memory, i.e., a memory in which cells can be addressed only by their content and not by any 'serial' address of conventional form, the problem of how to store tuples or sequences raises easy but suggestive problems. (Note that in conventional techniques one often stores sequence components in the order of memory cells; even in a list technique cells are generally chained by their physical address.) A plausible technique would involve storing the following information in a collection of free cells:

(a) an identifier for the entire sequence

(b) an item of the sequence

(c) the next item

(d) in the first item, a flag indicating that it is evoked by the sequence identifier alone, without requiring any additional prior item.

Then the sequence identifier s would evoke the first sequence item i_1; s and i_1 together would evoke i_2, and so forth, each item 'cueing' the next. This scheme will only work if the map from (s,i_j) to i_{j+1} is single valued. To handle cases in which this condition fails, information additional to the association (s,i_j,i_{j+1}) would have to be stored in certain cells. This additional information may be thought of as designating 'phase' or 'context' within the total sequence.

Considerations of this sort may explain why words like 'Mississippi' are relatively hard to spell correctly. The pattern of pairs in this word is

mi
is, ip
ss, si,
pp, pi

which could lead after sequence reconstruction to the following spellings ('p' being taken as an end signal)

mipi, misipi, missipi, misisipi, etc.

Even if two symbols of left context are used, one faces the following dictionary of triples

mis
iss
ssi
sis, sip
ipp
ppi .

These could lead after sequence reconstruction to the spellings

missippi, mississippi, missississipi, etc.

The notorious elusiveness of even medium-sized binary patterns may have a similar origin.

These reflections suggest a hypothesis which I stae in an esaggerated form in order to make it memorable: that the brain, as an associative computer, stores only sets (including sets of ordered pairs).

5. Retrieval/parse procedures as a model for the mental processing of sensory input.

The retrieval/parse procedures described in the preceding pages seem to define reasonable though undoubtedly very crude models of the manner in which the organic brain might process sensory input. In such a model, a continuing input stream of sensory data would cause the retrieval of similar items from one or more stored 'dictionaries', and at the same time the contents of these dictionaries would be modified by the parse-like process described above. In the following pages we shall pursue the line of thought, attempting to point out places in which processes like those which we have described can be used to model aspects of mental function.

The remarks made at the end of the last section are offered as a model of the internal process by which thoughts are converted into sentences. The thoughts producing a sentence are assumed to arise as an unordered collection of internal stimuli excited at some level of an overall process of associative retrieval set in train by ultimate external or internal cause. The ordering mechanism sketched at the end of section 4 then acts to arrange these initially unlinked elements. During such a process, elements of a syntactic character could also be retrieved and be integrated into the sequential structure being composed. Thought elements integrated into such a sentence might be 'cancelled' once the sentence was enunciated or written; remaining unintegrated elements could serve as nuclei for the formation of additional sentences. It may also be noted that, once enumerated or written down, the sentences produced by a speaker or writer themselves become external stimuli. Elements not perceived during the formation of a sentence will become visible in its external form, making possible repeated attempts at correction which lead eventually to a connected external sentence, and to the growth of a mass of sentences from an initial nucleus.

From this point of view lingustic behavior is seen to be merely a special variant of a much more general type of mental process, namely processes of error-correction and arrangement by means of which impromptu and highly variable sequential behavior (or plan) chains can be produced. Linguistic processes thus lie close to more fundamental processes of thought, of which they give an explicit, slightly specialized, representation. It may be conjectured that the process of serialization which we imagine to lie at the root of linguistic behavior is substantially the only process at the unconscious level supporting higher mental function, i.e., function which goes beyond a more elementary process of retrieval based on feature commonality. If this is the case, then beyond its innate lingustic ability the mind's only resource in dealing with logically complex situations is its ability to reason consciously and serially. This makes available the full power of a Turing machine, but of an inaccurate and very slow one. The preceding conjectures suggest that all complex learning at the unconscious level is the learning of one or another type of language, each such language making available a structure capable of converting unordered assortments of thought fragments into a serial pattern conforming to some 'syntax'. In this view, intuited passages of plans, mathematical proofs, computer programs, all arise in much the same way as sentence portions, i.e., as serializations of an unordered collection of elements, and especially as serializations found at an unconscious level to be syntactically well-structured. A partial plan of this kind, once become explicitly conscious, can then be elaborated in much additional detail, sometimes with success, while in other cases it may prove impossible to bring a partial plan to a state of completeness.

These same reflections suggest a model for 'light conversation', namely the bilateral digestion of sentences, each of which excites a set of associations whose ordering results in one or more additional sentences, and so forth iteratively.

It is well worth emphasizing that the processes we have described have a 'combinable' or 'algebraic' character. That is, they lead from blurred sequential input to blurred sequential output; from a string of symbols largely or slightly indefinite as to identity or position the conjectured principle of retrieval produces another such stream. In this process, an initial dictionary is incrementally modified by the action of the 'grammarless parser' which has been sketched. We can represent the overall process by writing

(1) output = input <u>retrieve</u> dict;

Note again that the evaluation of this function modifies the second argument 'dict'. The output of one such process can be used as input to another, a possibility which we could indicate by writing

(2) output = input <u>retrieve</u> $dict_1$ <u>retrieve</u> $dict_2$ ...<u>retrieve</u> $dict_n$;

For emphasis and brevity however we shall in what follows prefer to write

(2') output = input * $dict_1$ * ... * $dict_n$

as an abbreviation for (2).

Of course, the evaluation of (2) (or, equivalently, (2')) will modify all the arguments $dict_1,\ldots,dict_n$. Note that, in addition to these arguments, the effect of evaluating the expression (2) will also depend upon the feature extraction functions which convert the output of one retrieval into input for the next. However, we expect this dependence to be quite noncritical, i.e., expect that relatively wide changes in these hashing functions will not substantially affect either the output of (2) or the dictionary modifications ('learning processes') occasioned by its evaluation.

However, if the input stream falls clearly into two streams of separate modality, i.e., if each input character x_j may appropriately be regarded as a pair $x_j = (y_j, z_j)$ of independent inputs from which features are separately formed, then the proportion of features formed from y to features formed from z will give an overall 'emphasis' or 'slant' both to the output of (2) and to the dictionary changes which it occasions. The extreme cases are those in which either no features are formed from y or none from z, i.e., in which an input stream (which may possibly be the output stream produced by some other retrieval parse process) is seen in a particular 'projection'. To indicate the relative proportion in which features formed spearately from several input streams enter into the input to a particular retrieval, we might write

(3) $$\mathrm{com}(\langle \mathrm{input}_1,\alpha_1\rangle, \ldots, \langle \mathrm{input}_k,\alpha_k\rangle) ,$$

where $input_1,\ldots,input_k$ are input streams and $\alpha_1,\ldots,\alpha_k$ are coefficients of proportionality satisfying $\alpha_1 + \ldots + \alpha_k = 1$. In the degenerate case $k = 1$ the 'combine' function reduces to the identity; it is this special case that is shown in (2). For the function call that could conventionally be written as (3), I shall prefer the specialized, somewhat more condensed notation

(3') $$\mathrm{input}_1 + \mathrm{input}_2 \ldots + \mathrm{input}_k[\alpha_1,\ldots,\alpha_k]$$

It will also be convenient in writing the class of expressions which I wish to define to make use of an operator identical with the SETL <u>is</u> operator whose value is the value of its left-hand argument, and which assigns this value to the variable standing immediately to its right. We choose to designate this operator by the symbol '→'. Using this operator, together with the 'retrieval' and 'conbination' operators introduced above, we may write a class of formulae of the form illustrated by

(4) $$\begin{aligned} \mathrm{output} = \mathrm{input}_1{*}\mathrm{dict}_1 + \mathrm{input}_2{*}\mathrm{dict}_2[\alpha_1,\alpha_2]{*}\mathrm{dict}_3, \\ \mathrm{input}_3{*}\mathrm{dict}_4[\alpha_4,\alpha_4] \\ {*}\ \mathrm{dict}_5 \rightarrow \mathrm{outtemp} + \mathrm{input}_4{*}\mathrm{dict}_6[\alpha_5,\alpha_6]{*}\mathrm{dict}_7 \\ + \mathrm{outtemp}[\alpha_7,\alpha_8]{*}\mathrm{dict}_8 \ldots \end{aligned}$$

Such a formula indicates the manner in which several successive retrievals are occasioned by one or more input streams and the manner in which the outputs of these retrievals are in turn used as inputs to later retrievals. The final outcome of a process like that described by (4), and also the dictionary changes occasioned by this process, depends on the initial state of each of the dictionaries appearing in the formula. Of course, more general retrieval/parsing processes like those which we have described may involve still further parameters. In particular the rate at which items are added to dictionaries ought to depend not only on a crude threshold parameter of the sort we have envisaged, but on parameters describing maximum dictionary size and related "dictionary almost full" effects, as well as other parameters which may be imagined.

To the extent that processes like that described by (4) give an adequate representation of the numerous fragments of association and conditioning whose processing constitutes an aspect of mental activity, formulae like (4) may be taken as gross structural or anatomical descriptions of minds, at least in part. (In much the same way, a description of the manner in which a system of limbs and muscles are interconnected, written in a suitable algebraic notation embodying particular elementary mechanical relationships, would define the general structure of a particular 'body'.) Formulae of this type might, to the extent that their details could be filled in, serve several purposes. In the first place, they define a 'space' within which parametrized families of 'minds' can develop by evolutionary increments.

The view of mind suggested by this remark embodies certain elements which deserve to be made explicit. In particular mind is regarded as being rather homogeneous (as much so as organic tissue of any other sort). In particular, special algorithms and elaborated internal procedures resembling special purpose program subroutines are assumed to be absent. In favor of such an assumption, it may be argued that any non-trivial algorithm is a highly discrete entity, and therefore not the sort of thing which could evolve through a series of small changes in the manner typical for organ adaptation in the physical sphere. Second, a description such as (4) suggests that mind is relatively universal, i.e., that minds will differ among themselves in virtue of the size and modifiability of their dictionaries; the number of logical processing layers they incorporate; and in virtue of the 'dictionary initializations', particularly significant for dictionaries which communicate directly with an input stream originating externally, which determine the most immediately recognized elementary and compound features of external input; etc. The description (4) suggests that minds are similar to within some such degree of variation; the specific content which they may come to develop is of course a function both of such 'genetic' factors and of the sequence of external stimuli to which they are exposed.

It may also be noted that the learning processes described by the algorithms presented above faintly resemble some of the notions formerly embodied in F. Rosenblatt's much earlier 'perceptron' proposal. However, quite in contrast to this earlier work, our algorithms embody a principle of locality; which is to say that they allow the elementary details of a situation to be learned before any attempt is made to deal with the situation as a whole. This will hopefully allow algorithms of the type suggested here to learn much more rapidly than the older type perceptron algorithm. Note also that our algorithms learn structural properties of input strings merely from contact with these strings; the learning processes that have been described above do not require any system of 'rewards' or 'punishments' for their operation.

6. Some additional remarks. Anatomical structure of tissues which could support the retrieval/parse processes described above. Some comments on present efforts in artificial intelligence. An important problem ignored in the preceding discussion.

The parse-like processes which we have described can be supported very efficiently by logically active tissue consisting of neurons with the following properties.

(a) Neurons are initially 'blank'; they are wired in a pattern of layers, with each neuron of layer n stimulating a large, relatively random collection of neurons of layer n+1. Each neuron of layer 1 is stimulated by some relatively random subset of a collection of 'feature' extractors which signal the presence (or absence) of some basic sensed feature in an external input.

(b) Each neuron of layer n+1 is stimulated by a fairly large number of inputs from layer n. If the neuron is blank, each of these inputs is potentially significant.

(c) The first stimulation of a neuron by more than some minimum number of its inputs 'imprints' it with this pattern of inputs. Thereafter, it will fire whenever at least half these inputs are activated.

The suggestion made as (c) represents only one of a number of related possibilities, many of which might lead to similar results.

It is an attractive feature of (a), (b) and (c) that they describe a logical mechanism which can learn but which is very stable and primitive. In particular, the projected mechanism incorporates no sophisticated algorithm. As already remarked, this seems desirable from an evolutionary point of view.

It deserves to be noted that the similarity-finding operation which plays a central role in the foregoing algorithms is quite close to the operation known to be performed by single neurons in the brain. Neurons summate incoming stimuli and will fire if the sum of their excitations (minus some term describing the inhibitory inputs which play a role) exceeds a firing threshold. This observation suggests the following calculations (which are plausible, but may of course be quite misleading). A neuron testing its inputs for similarity with a stored pattern could accomplish 200 tests/sec., or 1 test in 5,000 μs. The same test requires approximately 1 μs on the CDC 6600. If this oepration is of central importance, it follows that large present-day computers have a power equivalent to not more than 5,000 neurons, as compared to the 10^{10} neurons present in the brain. Thus an obstacle facing efforts in artificial intelligence may be that the computers being used fall short of what is required by 6 orders of magnitude. Of course, such an enormous gap (if it exists) can in known cases be covered up by the use of clever, specially tailored algorithms. It may however be that in certain of the situations with which researchers in artificial intelligence are attempting to grapple, such algorithms simply do not exist. In such situations parallel feature-similarity steered association may be the method of choice. This, if true, would lead one to regard present artificial intelligence efforts as attempts to substitute very artificial trick algorithms for the use of parallelism on an enormous scale, an attempt in which difficulties of the kind presently experienced may be inherent.

It is obvious from the above reflections that an optimal method for implementing the similarity primitive introduced in section 3 ought to be sought. In the absence of any non-obvious method for retrieving items similar to a given input from a large dictionary of items, the duplication of brain function might require a computing device capable of performing 5×10^{13} comparisons/second, and possibly storing as many as 10^{13} bits. Assuming that in 15 years 1 megacycle, million bit computers are available for approximately $1, this could require a $50 million computer even in so very advanced a technology.

The models of learning proposed in this newsletter ignore a very important problem, which deserves to be mentioned explicitly. This problem may be formulated in various ways. How can a learning device of the type envisaged, which forms a randomly hashed internal representation of an input stream, be linked to an output device (or algorithm)? In particular, how could a learning device to which an output device was attached learn which of its own internal processes led to the production of a given output pattern (problem of imitation)? More generally, what mechanisms, in addition to those which have been proposed, can account for the formation of 'conditioned reflexes', i.e., of internally stored associations γ between pair α,β of items, associations of a form which allow γ, when excited, to have the effect of its component α? The models which have been described do not answer these questions and thus remain incomplete in a fundamental way.