PARALLEL PROCESSING OF NATURAL LANGUAGE

Eliezer L. Lozinskii and Sergei Nirenburg
Department of Computer Science
The Hebrew University
Jerusalem, 91904 Israel

Abstract: An approach to parallel processing of natural language is described. It exploits the phenomenon of locality of text elements and is based on a multiprocessor system architecture which is aided by a special type of table grammars. The operation of the system is illustrated by an example of syntactic parsing in a subset of English. We describe the FRIENDLY-NEIGHBOURS and PYRAMID algorithms for parallel parsing.

Keywords and phrases: artificial intelligence, natural language processing, parsing, locality, parallel processing.

Introduction.
Among the various traditions accumulated during the twenty-odd years of research in computer processing of natural languages, the understanding that any such processing is to be done sequentially has the deepest roots.

Even the most advanced natural language processing systems employ the sequential mode as a necessary evil (see e.g. Schank & Ricebeck, 1981; Arens, 1981; Marcus, 1979). The recent outstanding advances in VLSI technology suggest that a re-evaluation of this tradition may be in order.

And indeed, there has been some progress in "parallel thinking", notably in problem solving (see e.g. Kornfeld, 1981) and speech understanding (see e.g. Erman et al., 1980 on HEARSAY-II, in which the computing resources are allocated on the basis of the relative priority of hypotheses thus introducing a kind of functional parallelism. A somewhat similar approach was suggested by Phillips and Hendler (1981).

In what follows a different and, we believe, much more powerful kind of parallel processing of natural language is suggested, which is not built upon functional distribution. It is based on the distribution of the input stream elements.

The proposal is to assign a processor to every item of input (generally, a word) and provide each such processor with an identical package of software, so that all of them will be equal in status and modus operandi.

In addition to improving the performance of a processing system, this property enhances the reliability of such a system because it is envisaged that if a certain number of processors fail, the remaining ones (due to their similarity) will take upon the de-activated teammates' job.

Our approach exploits a certain property found in natural language which we call locality.

Let us consider a text as a linear (one-dimensional) string made up of discrete elements w_i:

I = \{w_1, w_2, ..., w_n\}

Suppose this string enters a natural language processor (NLP),

Being fed with I, the NLP will produce a structure of the form

S(I) = \{v_1, v_2, ..., v_n\}.

The elements v_i of the structure can generally be of various types: words in the object language and/or words in a metalanguage and/or various kinds of delimiters.

Let D(v_i) be the minimal subset of I which determines v_i in the sense that the information carried in the elements of this subset is necessary and sufficient for outputting v_i by the NLP. We shall also say that v_i depends on D(v_i).

Let b_1 be the index of the leftmost element in D(v_i), and h_1, the index of the rightmost element. For example, if D(v_i) = w_3, w_5, w_10, then b_1 = 3 and h_1 = 10.

Let us define the locality of an element v_i as

\[ l(v_i) = \frac{h_1 - b_1}{h_1} \]

This function has the following interesting properties:
a) if an element w_i in the NLP output string depends on exactly one element of the input string, w_j, then b_1 = h_1 = j, and, consequently, the locality of v_i attains its maximum possible value of L(v_i) = 1; b) if an element w_i depends on a range of the input elements, w_j, ..., w_k (k > j), then L(v_i) < 1, and, as the range grows, the locality decreases towards zero.

The notion of locality has a direct correspondence to the potential performance of parallel processing of natural language.

Let T^sec_i, T^par_i stand for the time of constructing an output element v_i in sequential or parallel mode, respectively. It can be shown that the relative gain of parallel processing against sequential processing is equal to the locality of this element:

\[ \frac{T^sec_i - T^par_i}{T^sec_i} = L(v_i) \]

The notion of locality provides a basis for developing systems of essentially non-functional parallel processing of natural language. In what follows we illustrate this approach by presenting a model of parallel syntactic parsing of English text. We envisage application of the method in a number of other subfields of AI but as an initial application we have chosen parsing English since it is notionally rather low-level task in natural language processing.

System Architecture.
We consider a computational model involving a...
a large number of processing elements (PE), all of which are similar, relatively simple and interconnected by means of a simple communication network (Fig. 1).

Every processor \( P_i \) contains two registers, the internal one, INT\(_i\), which is used for processing that \( P_i \) carries out and is not accessible from outside \( P_i \); and external register, EXT\(_i\), intended for the purposes of communication with other processors. Every \( P_i \) has direct access to the contents of the external registers of its immediate neighbours to the right and to the left, EXT\(_{i-1}\) and EXT\(_{i+1}\). These links between the neighbouring processors constitute the local bus of the system.

In the course of parallel processing there may appear a need for broadcasting signals (for instance, certain kinds of alarms) from one processor to all the others. A global bus provides a means for this. The local and the global buses are installed in a ring-like configuration allowing the circulation of messages through the system.

Every processor is controlled independently by means of its own operating system. The software packages of every processor are identical, which means that all the processors are interchangeable.

Illustration: Parsing English.

For the purposes of this presentation we have restricted ourselves to a subset of English which includes such syntactic work classes as Determiners, Nouns, Adjectives, Verbs, and Prepositions. In this example the elements of the input stream, \( w_i \), will be either English words or punctuation marks, whereas the elements of the output stream, \( v_j \), will include dictionary forms of the words appearing in the input stream and a restricted number of auxiliary (non-terminal as well as terminal) symbols signifying the roles of a word in a sentence, displaying its class affiliation, etc. The output will be close to the deep structure tree of the transformational grammar. It follows from the definition of locality that the locality of an element increases with \( k_j \) and can approach infinity indefinitely closely when \( k_j > n \), where \( n \) is the number of words in a string. Therefore, if \( k_j \) can approach \( n \), we say that the element \( v_j \) has high locality.

Proposition: In syntactically correct sentences of our subset of English
a) NP (NounPhrase) has high locality;
b) VP (Verb Phrase) has high locality;
c) PP (Prepositional Phrase) has high locality.

Proof. a) Let a certain input word \( w_i \) be 'the'.
Then there is an element \( v_j \) in the output string such that \( v_j \) is "NP" and \( k_j = i \). 'The' can approach the end of an input string.
b) Let now a certain \( w_i \) be a verb; then there is a \( v_j \) such that \( v_j \) is "VP" and \( k_j = i \). A verb can approach the end of a sentence.
c) Let \( w_i \) be a preposition. Then there exists \( j \) such that \( v_j \) is "PP" and \( k_j = i \). A preposition can approach the end of a sentence. QED.

Since our subset there are only five form-classes of elements (words) this proof seems simplistic. But in "real" language this theorem will be proved similarly, though in a possibly more complicated manner.

Encouraged by the finding that certain sentence constituents are highly local, we proceed to build a syntactic grammar-cum-interpreter for the abovementioned subset of English. The "parallel" approach and the architectural environment chosen allow grammars of a kind different from those implemented in the sequential models of processing.

We obtain for processing a sentence \( n \) words long (for the purposes of our analysis we would consider the punctuation marks separate words). Every processor, as we already know, is assigned to a single \( w_i \). The processing starts with the "morphological" stage, which consists in dictionary searching for all the possible citation-forms of the input words. Note that morphological analysis has not been a specific concern of ours, since there are several well-known computer programs which carry out this kind of information processing (cf., e.g., Peterson, 1980).

The results of this "morphological" stage of analysis are displayed as a two-dimensional array each member \( w_{ij} \) of which corresponds to a single reading \( j \) of a word \( w_i \) and is a list containing information on a syntactic meaning of a word. Every column in this array is the 'field' of a single processor. This 'field' is the set \( W_i \) of the readings of the given \( w_i \).

On obtaining the above array we proceed to the syntactic analysis as such. It is subdivided into two stages:
1) the search for candidates for a grammatically acceptable string of category symbols (the latent stage) and 2) detecting sentence constituent boundaries and building the result tree.

The search for a grammatically acceptable string starts with the help of a grammar represented in Table 1. The table, organized as a matrix, contains

<table>
<thead>
<tr>
<th>O</th>
<th>Det</th>
<th>Prep</th>
<th>N</th>
<th>Adj</th>
<th>V</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>V</td>
<td>N</td>
<td>U</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Prep</td>
<td>V</td>
<td>Prep</td>
<td>V</td>
<td>Prep</td>
<td>Adj</td>
<td></td>
</tr>
<tr>
<td>Det</td>
<td>N</td>
<td>ALARM</td>
<td>N</td>
<td>N</td>
<td>Adj</td>
<td>N</td>
</tr>
<tr>
<td>Prep</td>
<td>N</td>
<td>Prep</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Adj</td>
<td>N</td>
<td>Prep</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>V</td>
<td>Prep</td>
<td>V</td>
<td>Prep</td>
<td>Prep</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td>N</td>
<td>V</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Prep</td>
<td>V</td>
<td>Prep</td>
<td>V</td>
<td>Prep</td>
<td>Prep</td>
<td></td>
</tr>
<tr>
<td>Adj</td>
<td>N</td>
<td>Prep</td>
<td>N</td>
<td>Adj</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adj</td>
<td>Prep</td>
<td>Adj</td>
<td>Adj</td>
<td>Adj</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. A grammar for a subset of English.

(continuation on next page)
information on the grammaticality of all the possible triads (substrings of length 3 in the input string) of syntactic classes marked in our sublanguage. The rows of the matrix are denoted by the left elements of the triads; the columns, by the right elements; at the intersection of the given row and column one will find a list of syntactic classes whose members can appear as the middle element of the triad. An empty set of syntactic classes for a certain context causes an alarm in the system. The symbol '∅' signifies the beginning of a sentence, and '.∅', the end of a sentence. 'U' denotes the general set of syntactic classes (in this example its cardinality is 5). N stands for 'noun', V for 'verb', Det for 'determiner', Adj for 'adjective' and Prep for 'preposition'.

What follows is a description of the program 'FRIENDLY-NEIGHBOURS' which carries out the first stage of the syntactic analysis in the autonomous (local) mode (cf. the flowchart in Fig. 2). It chooses the reading of the given word $w_1^i$

<table>
<thead>
<tr>
<th>Continuation of Table 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>Adj</td>
</tr>
<tr>
<td>Prep</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>Prep</td>
</tr>
</tbody>
</table>

The above behaviour, of course, 'friendly', but there is a danger that this postponement will become infinite. To avoid this deadlock the program must sometimes act in an 'unfriendly' way: choose a reading on its own. The procedure 'HEADS-OR-TAILS' contains the mechanism which decides upon a course of action. It essentially 'tosses a coin'. If the outcome is 'tails', control is returned to 'NEIGHBOURS' for another round of inquiries. If, however, the outcome is 'heads', the program acts 'unfriendly', i.e. the procedure 'INTERSECT' is activated. It is also activated when the output of 'NEIGHBOURS' is not \(q,∅\). 'INTERSECT' is the only place in the program where grammatical information from the grammar Table 1 is solicited. It retrieves a set of readings compatible with \(EXT_{i-1}^i\) and \(EXT_{i+1}^i\) from the grammar (Table 1) and then intersects this set with \(R_i^i\). The resultant set \(RES_i^i\) is the space of readings of \(w_1^i\).

The cardinality of \(RES_i^i\) is counted by 'COUNT2', and three possible outcomes are distinguished. If \(RES_i^i\) contains only one candidate, then it is recorded in \(EXT_i^i\) by the procedure 'RECORD' and the program terminates unless an Alarm is received from a neighbour. If \(RES_i^i\) is empty, then the program issues an Alarm.

If \(RES_i^i\) contains more than one element, then the procedure 'READING-CHOICE' decides which of these to choose as the candidate reading. The simplest mode of decision is the random one. But in general, this is a point at which heuristics can be introduced.

The output of 'FRIENDLY-NEIGHBOURS' consists of all the strings of compatible readings of \(w_1^i, w_2^i, \ldots, w_n^i\). This signifies the end of the first stage of the syntactic analysis.

Each candidate string, as soon as it is obtained, is transferred to the second stage of the analysis (which can be carried out by a separate set of processors), while the first stage resumes operation until no more well-formed strings can be found.

The task of Stage 2 is to build the upper levels of the parse tree. To achieve this one has to detect where sentence constituents (such as noun, verb or prepositional phrases) begin or end, and then match the corresponding boundaries and check for correct composition of constituents in the sentence. A part of this task can be done locally. The rest requires a wider scope. The whole process is performed, therefore, in two phases.

I. Constituent Boundary Generation.

This is a description of the intermediate stage of the syntactic parallel parsing mechanism, sandwiched between the FRIENDLY-NEIGHBOURS and the PYRAMID. Its objective is to take the output of the FRIENDLY-NEIGHBOURS program (a string of syntactic category symbols) and produce a string of those constituent boundaries which can be detected locally.

Every processor PE (remember that they work in parallel) tries to decide whether a certain constituent starts or ends with the word assigned to
This task exploits the locality inherent in the grammar that defines a language being processed. Consider the following grammar for a subset of English (this is already a certain extension over the subset used for illustrating the FRIENDLY-NEIGHBOURS):

Grammar 1:

\[
S \rightarrow S \ delim S^* \ | \ [[PP^*] NP \ VP] \ | \ [NP^* \ NP] \ | \ [Det] Adj^* N^* \ N \ [PP^*] \ | \ [Det] Adj^* N^* \ N \ [S] \ | \ S \ VP \ | \ Adv^* V \ NP \ | \ Prep \ NP^* \ PP \ | \ Prep \ NP^* \ delim \ | \ and \ | \ or \ | \ NP^* \ | \ which \ | \ whom \ | \ Det \ | \ the \ | \ this \ | \ these \ | \ his \ | \ her \ | \ its \ | \ their \ | \ N \ | \ book \ | \ man \ | \ John \ | \ school \ | \ Adj \ | \ big \ | \ red \ | \ wooden \ | \ Adv \ | \ quickly \ | \ seldom \ | \ Prep \ | \ in \ | \ out \ | \ of \ | \ to \ | \ NP^* \ is \ the \ symbol \ for \ the \ "virtual" \ NP, \ the \ one \ which \ is \ represented \ in \ the \ sentence \ as \ a \ relative \ pronoun.
\]

Every processor PE makes its decision depending on the data in its immediate neighbours (should a grammar so require, one can envisage taking into account more than one neighbour on each aide). It executes the program driven by a decision table based on the grammar for a language subset chosen. Table 2 shows a fragment of such a table for the subset of English specified by Grammar 1. In Table 2 the column i corresponds to the data of the PE itself; the columns i-1 and i+1 correspond to its neighbours; the decision column contains the constituent boundaries detected by the PE.

<table>
<thead>
<tr>
<th>i-1</th>
<th>1</th>
<th>i+1</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Det</td>
<td>N</td>
<td>(SNP)</td>
</tr>
<tr>
<td>Det</td>
<td>N</td>
<td>NP'</td>
<td>(SNP')</td>
</tr>
<tr>
<td>N</td>
<td>NP' Det</td>
<td>(SNP')</td>
<td></td>
</tr>
<tr>
<td>NP'</td>
<td>Det</td>
<td>N</td>
<td>(NP)</td>
</tr>
<tr>
<td>Det</td>
<td>N</td>
<td>V</td>
<td>(NP)</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>#</td>
<td>(VP)</td>
</tr>
</tbody>
</table>

Table 2. A locality-based decision table for constituent boundaries (a fragment).

The result produced by each PE is either nil or one or more quadruples:

\[<a \ y \ z>\]

where "a" is the name of the constituent obtained by the PE, an element from the set \{S, NP, NP', VP, PP\}; x and y are the coordinates of the constituent. If the beginning of a constituent has been obtained by the processor P[i] then \(x = 1\) and \(y = 0\). If the end of a constituent is detected then \(x = 0\) and \(y = 1\). If the constituent begins and ends in one and the same position, then \(x = y = 1\). At this stage \(z = 1\) for \(i = 1\), \(n - 1\), \(z = 2\) for \(i = n\) (end of sentence). The meaning of \(z\) is explained in the next section. Thus, the output of this stage is a string of the above quadruples, numbered from left to right continuously; each quadruple represents one sentence constituent CT:

\[\{CT[i]\}, \ i = 1 \ldots k, CT[1] = <a \ y \ z>\]

where \(k\) is the number of generated constituents.

This process does not guarantee that all the relevant boundaries will be discovered (as a consequence of the extreme locality of the decision table). This task will be accomplished at the next, "pyramidal", stage of the parsing mechanism.

II. The PYRAMID.

From the previous stage we obtain a string of quadruples \(CT[1] = <a \ y \ z>\) in which some of the quadruple coordinates \(x\) or \(y\) are nil (this means that the previous stage supplied only one of the corresponding constituent boundaries of the needed two). Now we embark on finding the missing coordinates.

As a rule, a constituent extends over a number of words in the original sentence. So in order to account for both boundaries one is to match the "field of vision" of a processing element to the length of the constituent. Incidentally, the incompleteness of boundary search at the previous stage is a consequence of its strict locality accompanied, naturally, by a very narrow field of vision.

Sentence constituents for which a missing boundary is to be found come in various sizes. Some of them are just two words long, one of them in the whole sentence (note that boundaries of one-word constituents are always determined at the previous stage). The presence of a number of distinct constituents in a sentence speaks for making their processing parallel. The variability of their size compels the use of an adaptable field of vision for every processing element.

The above considerations make the basic for the "pyramidal" algorithm for finding the missing coordinates \(x\) and \(y\), the description of which follows.

The processing is carried out by PE's organized as shown in Fig. 1 and equipped with identical software packages. The core of the package is an ATN grammar (Woods, 1970; Bates, 1978) working on the lexicon of constituent boundary names.

The input for the pyramid is the output of the previous stage, a string \(CT[i]\) which becomes the base of the pyramid in Fig. 3.

There are several steps in "pyramidal" processing (from bottom to top). At every step the PE's involved are assigned a chunk of the input string. A chunk is a string of quadruples such that in the last one \(z = 1\) (end-of-chunk) and in the rest of the chunk \(z = 0\) (at end-of-sentence). Every PE processes its chunk, that is finds whatever possibilities there are for completing coordinates within the latter and records the result in a buffer. When any two neighbouring PE's reach end-of-chunk then one of them attains the active status. The other (passive) PE

![Fig.3. The Pyramid.](image-url)
resigns and joins the pool of unemployed PE's. The active PE proceeds to consume the output buffer of the other, thus incorporating it in its own chunk (and this signifies the beginning of the next step). After this the memory space previously owned by the passive PE is returned to the system pool of available resources (see procedure CLEAR in program below).

We can now see that the number of PE's involved in the pyramid drops with every passing step and, more importantly, the length of chunks grows - until at last the whole sentence is a single chunk. At this moment the process reaches the top of the pyramid, and the one remaining PE will be responsible for wrapping the processing up and outputting the results.

The pyramid processing is controlled by the variable status of the PE's involved. Status can be "active", "passive", "final" (this triggers the wrap-up operations) or "neutral" (self-explanatory). The assignment of a status value to a PE P[i] done along the lines of the following protocol:

Procedure STATUS-CHOICE;
(* every PE contains an output buffer, BOUT, a pointer to this buffer, PBOUT, the 'left' and 'right' flags, LF and RF, pointers to PE's neighbours, LEFT and RIGHT *)

Procedure LOOK-LEFT :
begin
LF := 1;
(* RF(LEFT) - right flag of left neighbour of given PE *)
if RF(LEFT) = 1 then STATUS := passive
else LF := 0
end (*LOOK-LEFT*);

Procedure LOOK-RIGHT;
begin
RF := 1;
if LF(RIGHT) = 1 then STATUS := active
else RF := 0
end (*LOOK-RIGHT*);

begin
STATUS := neutral;
repeat
if LEFT = nil then
  if RIGHT = nil then STATUS := final
  else LOOK-RIGHT
else
  if RIGHT = nil then LOOK-LEFT
else begin
  toss-a-coin
(* call random number generator *)
  if "heads" then LOOK-RIGHT
  else LOOK-LEFT
end
until STATUS <> neutral
end (*STATUS-CHOICE*);

Main processing is done by the procedure CORE which activates an interruptable version of an ATN (procedure IATN) working on constituent names (members of CT's) as input words and producing the upper levels of the parse tree for the initial input sentence (the two lower levels, - the terminal symbols and the names of their syntactic categories, - are inherited from the FRIENDLY-NEIGHBOURS stage and appended to the result tree by the procedure WRAP-UP). The mechanism of interrupts and activations is managed with the help of a stack. The input string (a chunk) for IATN is pointed at by the variable WF ("where-from").

Procedure CORE (stack);
beg
WF := PBOUT (RIGHT);
(* PBOUT (RIGHT) points at BOUT of right neighbour *)
IATN (stack);
(* IATN-SIGNAL is set by procedure IATN when it encounters unacceptable inputs and this invokes procedure ALARM-PROC which records the place of a grammatical mistake and, if necessary, sends the string of PE's if one crashes so that the processing could still be carried on by remaining PE's *)
if ALARM-SIGNAL = true then ALARM-PROC
else begin
V := RIGHT(RIGHT);
(* V points at the right neighbour's (* right neighbour *)
CLEAR (RIGHT);
RIGHT := V
RF(RIGHT) := RF;
end
end (*CORE*);

The above procedures are united in the main-program PYRAMID:

Program PYRAMID (input-string, result-tree);
begin (* n is the length of the input-string *)
  for all i := 1 to n in parallel do
  begin
    (* start initialization *)
    STATUS := active;
    stack := nil;
    IATN (stack);
    (*end of initialization *)
    while STATUS = active do
      begin
        STATUS-CHOICE;
        case STATUS of active : CORE;
        final : WRAP-UP(result-tree);
        passive : TO-POOL;
      (*TO-POOL returns passive PE to the system*)
    end
  end
end (*PYRAMID*);

Let us scan the above process with the help of a small example. Consider: "The man whom the boy saw left!" Should this sentence enter the above system, the FRIENDLY-NEIGHBOURS output on it will be the string

DET N NP' DET N V V

This string is the input for the constituent boundary generation stage. At this stage we can use Table 2 (fortunately, we have the right fragment) for generating the possible boundaries in the local mode. The output here is the string of quadruples CT:

<51y1> , <N1y1> , <S3y1> , <NP'331> ,
<N3y1> , <V6y1> , <VP7y2> .

The steps through which the PYRAMID program fills in the missing coordinates are illustrated in Table 3, in which the numbers of processors and the contents of their output buffers is shown for every step. Note the changes in the value of y in the quadruples. Points of decision as regards a boundary are underlined. The grouping of processor at any step is done at random (cf. the procedure STATUS-CHOICE).
<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE BOUT</td>
<td>PE BOUT</td>
<td>PE BOUT</td>
<td>PE BOUT</td>
</tr>
<tr>
<td>1 S1 y1</td>
<td>1 S1 y0</td>
<td>1 S1 y0</td>
<td>1 S1 y0</td>
</tr>
<tr>
<td>2 NP1 y1</td>
<td>NP1 y0</td>
<td>NP1 y0</td>
<td>NP1 y0</td>
</tr>
<tr>
<td>3 S3 y1</td>
<td>S3 y1</td>
<td>S3 y1</td>
<td>S3 y1</td>
</tr>
<tr>
<td>4 NP'331</td>
<td>NP'330</td>
<td>NP'330</td>
<td>NP'330</td>
</tr>
<tr>
<td>5 NP 4y1</td>
<td>NP 4y1</td>
<td>NP 4y1</td>
<td>NP 4y1</td>
</tr>
<tr>
<td>6 VP6y1</td>
<td>VP6y1</td>
<td>VP6y1</td>
<td>VP6y1</td>
</tr>
<tr>
<td>7 VP7y2</td>
<td>VP7y2</td>
<td>VP7y2</td>
<td>VP7y2</td>
</tr>
</tbody>
</table>

Table 3. An example of PYRAMID application.

In this example our "net profit" over sequential processing is the determination of the y = 5, 6, and 7 in the three last lines, which was done before the last step, in parallel with other processing.

Conclusion.

This paper describes an approach to parallel processing of natural language. It exploits the locality phenomenon and a multiprocessor system architecture and is aided by a special kind of table grammars. We also hypothesize that locality finds justification in human language processing. This means that such an approach can be extended to the most general problems of language analysis, use and synthesis. This method provides for a parallel and quasi-independent processing of different chunks of the sentence. This has a number of implications, in particular, makes the method suitable for processing incomplete or incorrect sentences; the system will process the sentence despite the mistakes (while an ATN parser, for example, would stumble on an unexpected input symbol) and will produce a correct output for the "undamaged" parts of the sentence.

References.


