Abstract

Mitchell Marcus' motivation for developing the deterministic parser was to model an intuition of how humans process language. His Ph.D thesis (1977) describes PARSEPAL, an implementation of his idea and especially the determinism hypothesis.

Other designers' implementations, which we examine, cover lexical ambiguity and computational aspects of the human sentence processing mechanism. The work is directed toward the use of the Deterministic Parser in the limited text machine translation environment.

Keywords: AL, NLP, Deterministic Parsing.

Introduction

This paper shall examine several Deterministic Parsers beginning with the first, Marcus' PARSEPAL (Marcus, 1980), followed by those parsers designed and implemented by Milne, Hendrick, Church and Hindle.

PARSEPAL's design is based on a psychological model of how we parse language and Marcus' Determinism Hypothesis (Marcus, 1980 p2).

There are inherent in Deterministic Parsers certain fundamental features. These features occur as computational restraints and as part of the parser's structure. The limitations concern the type of syntactic constructs produced by the parser. All constructs produced from the input must be output as part of a syntactic construct and all must be permanent. There are no temporary syntactic constructs built into the parser because of the restrictions in the machine.

The parser's structure has three main features. It has to some extent to be data driven, but there is a need to reflect expectations based on the constructs already formed. It has also a need for a lookahead facility.

Marcus' Parser PARSEPAL

PARSEPAL has two major data structures - a push-down stack called the Active Node Stack and a Lookahead Buffer which can hold three grammatical constituents; a grammatical constituent can range from a word to a clause.

The Lookahead Buffer is described by Marcus as the heart of PARSEPAL. This buffer works mainly on a FIFO principle. The parser processes words in the first buffer cell, having the facility for examining the remaining two cells and thus knowing with what type of word or phrase it is dealing.

Incomplete constituents are held in the Active Node Stack, to the bottom node of which constituents can be added until what is known as the Current Active Node. If the node's grammatical role is as yet unknown, the new larger node is popped from the bottom of the stack into the first buffer cell for further processing. Otherwise it is popped from the bottom of the stack and pushed into the cell above.

Using both of these data structures ensures that PARSEPAL operates in both a look-up and bottom-up manner. The activation of the rules which recognise subconstituents for the node is top-down whilst the triggering of the rules once activated is bottom-up. In the stack we have mothers looking for daughters - a top-down process and in the buffer daughters looking for mothers - a bottom-up process.

According to Marcus, PARSEPAL's grammar is designed to capture the generalizations of a generative grammar and the make-up of constructs produced comes from Chomsky's and Winograd's differing concepts of Annotated Surface Structure. It consists of packets of pattern/action rules, the names of the packets are placed on the stack, forming part of the tree structure. The patterns match with constituents in the buffer, the Current Active Node and Current Cyclic Node whilst the actions are a set of operations on the buffer constituents. The rules are ordered according to numerical priority. The rules are written in FIDGIN, a formal language and are translated by a grammar translator into LISP.

PARSEPAL also uses special rules known as Attention Shift Rules which allow the parser to shift attention from the constituent in the first cell of the buffer to another constituent if there is proof that this constituent initiates a higher level constituent of a given type. These rules are used mainly for 'noun phrases' but can initiate other constituents e.g. clauses beginning with as. When the higher-level constituent has been built, the parser's attention is returned to the first buffer cell.

Marcus found that his Deterministic Parser became stuck, as people do, when presented with a Garden Path Sentence e.g.

The horse raced past the barn fell.

His explanation as to why such sentences cause problems - on average people can only access up to three buffer cells of lookahead which is the problem is to initiate a higher level conscious grammatical problem solver to then reinitiate parsing of the fragments of input not already parsed. These fragments would then be
analyzed by a set of heuristics to discover where the parser failed.

Milne's Parser PADS

Robert Milne researches further the topic of
determinism, especially the area of lexical
ambiguity which Marcus had only touched upon
(Milne, 1978, 1986). Milne's PARS, ROBIE, is
implemented in Prolog. It has an Active Node
Stack identical in design to PARSIFAL's.
However, Milne designed different structures for
ROBIE's buffer mechanism - two static lists
where he regards three buffers to be excessive.
The buffers can move to the right or left
depending on whether constituents are being
pushed onto or popped from the stack. Words
enter the second buffer, before being processed
in the first.

ROBIE's grammar is also made up of packets
of pattern-action rules, eighty percent of which
are identical to PARSIFAL's. ROBIE's grammar
rules also have a numbering priority. ROBIE
does not use attention shift since Milne
considers that two buffers can handle most
constructs. In those cases where three buffers
are necessary, Milne suggests that a non
syntactic processor would be called. The
grammar also includes rule context semantic
conds. The information obtained from the semantic
condings produces predicate calculus assertions.

How does ROBIE resolve the problem of
lexical ambiguity? Firstly the dictionary contains
compound lexical entries; each one has
attached all the features for all its parts of
speech. After dictionary consultation, words are
morphologically analyzed. The process which
occurs next is disambiguation. ROBIE's rules
match with the features of the constituents of
either one of the two buffers and the features
have matched, the word has always that feature.

Noun phrase ambiguities such as singular
head-noun, verb/adjunct ambiguity and other
pronominal ambiguities can be disambiguated quite
easily because of the word order of the noun
phrase. If we take the example of the noun
phrase - A broken arm - broken in different
ways: adjective and a verb in the dictionary and
arm as a noun and a verb; broken will be
disambiguated as an adjective since it is preceded
by A and arm is disambiguated as a noun since it
follows an adjective.

Milne found that other examples of part-of
speech ambiguity could be handled if certain
rules could be called upon to aid disambiguation,
such as rules for checking verb sub
-categorisation, the structure of verb groups and
paren/number codes. Thus, ROBIE is designed
tochke that words and subjects agree as well as to
reject ungrammatical sentences.

Marcus developed special diagnostic rules to
cover cases of ambiguity for have, to, what,
which, that. However, such ambiguities, are
handled by the checks that are encoded into
ROBIE's semantic rules that could not be
resolved by syntax alone Marcus' special
diagnostics also could not handle i.e. Garden
Path sentences:

The cotton clothing is made of grass in
Mississippi.

Milne refutes the notion that Garden Path
Sentences are categorised by not being
disambiguated within three buffer cells. He
sites the example:
The prime number few
which fits into three buffer cells but also
causes people to Garden Path. Milne suggests
that non-syntactic information is significant when
processing Garden Path Sentences and that people
call upon semantics to help disambiguation. ROBIE has a semantic checker.

Berwick's Parser L.PARSIFAL

Robert Berwick's parser L.PARSIFAL is a modified
version of PARSIFAL used as part of an
implementation of a classical parser flat to
process English (Berwick, 1985). The rules which
are of the pattern/action type are acquired
in the processing of the input sentences; this
provides the learning theory.

The modifications designed into L.PARSIFAL
are included to simplify its use. For example, parser -> NP at it
would be easier for the acquisition procedure to
learn, and also to have a more standard design.
L.PARSIFAL has an Active Node Stack and three
cell Lookahead Buffer similar to PARSIFAL. It
is implemented in LISP without the use of PEGON.
However the grammar rules of the parser are
somewhat different. The rules which correspond to the pattern of packets
L.PARSIFAL does not use attention shift or
packets of rules but does make use of X-bar
theory to create and label phrase nodes.

We can look at these changes in practice.
Berwick dropped the packeting system of rules to
aid simplification. He suggests that the use of
the packeting system in PARSIFAL is derived from
producing a tree structure of the input sentence
as it is parsed. He suggests that this
tree structure output could be dropped and a more
complete non-terminal symbol called a dotted rule
could be used to effect how far one has got in
parsing the sentence. A dotted rule is a context
free re-write rule with a marker dot
placed at some point in the the right-hand-side
of the expansion of the sentence. The action is
not different. The parser does not process
the input sentence, but instead parses it into
the first buffer cell where it is
processed again by the next grammar rule which
usually attaches it to the parent node.

Berwick decided that the dotted rule should
be put on the Active Node Stack in place of
the simple non-terminals such as 'S' or 'NP' since
these non-terminals are part of dotted rule
information. The dotted rules are an exact
representation of a tree structure so it is a
good replacement to put on the Active Node Stack.
L.PARSIFAL's packet names are, in fact, in one-to
one correspondence with dotted rules.

The actual grammar rules themselves are also
modified. In L.PARSIFAL, the regular rules do
not have a numbered priority. Instead, specific
rules execute before general rules. A specific
rule requires different from the
first buffer cell. If there are no specific
rules the general rules will fire. Grammar rules are
not allowed to have a sequence of actions.
The patterns of the grammar rules must
refer to an item in each cell of the buffer which
they did not in PARSIFAL.

L.PARSIFAL operates in a more standard
manner to PARSIFAL, which Berwick considered
to deviate from a bottom-up parser in two ways.
The first deviation in PARSIFAL is that it is a
sequence of constituents e.g. NP, DP, NOUN, can be dealt with individually, that is pushed
when a phrase is closed.

YAP is a Deterministic Finite State Machine. Finite State is achieved by the buffers being out to a fixed length and the buffer cells also being limited. However, the bounds on the two buffers have not been defined. The first three buffer cells of both buffers are referenced frequently. The length of the lower buffer measures the degree of lookahead. Church, unlike Marcus, is not committed to limited lookahead.

In YAP, unlike PARSIFAL, parse nodes are abstractions of features not the parse tree. The node size must be restricted so that unbounded memory cannot be encoded into it; this results in a predicate being tested in a fixed amount of time. PARSIFAL stores subconstituents in stack cells leading to an arbitrary amount of time being taken up in the search of a subconstituent for a node.

PARSIFAL works on a number of pattern/action rules. For Church, problems of both performance and competence when using this type of rule, since it can entangle components. He, thus, decided to use phrase structure rules, as Berwick did, so that the next act is a memory in an ordered fashion. The phrase structure component could cover most normal unmarked cases, production rules are used for marked examples.

Charniak's PARAGRAM Eugene Charniak's parser is another modified version of PARSIFAL (Charniak, 1983). Charniak developed PARAGRAM to prove that a syntactic parser could handle ungrammatical sentences. He regarded this a necessary development since humans can process ungrammatical sentences. PARAGRAM is semi-grammatical which means that it can parse a correct grammar of English for as long as it is necessary, but will also process sentences that do not conform to the grammar, making note of where they are ungrammatical.

PARAGRAM's rules have been defined to capture the relevant generalisations about language similar to Transformational Grammar. We are told that the parser is reasonably efficient, but would be more so if the implementation was run in a parallel fashion.

According to Charniak, it only requires a small modification to the original design of PARSIFAL to allow PARAGRAM to handle ungrammatical sentences. PARAGRAM's rules are not tested sequentially, in order of priority, but are considered as being tested in parallel. The text does not return a yes/no answer, but a goodness rating with the higher numbers reflecting the good match between rule and input. PARAGRAM repeatedly processes the rules, beginning with those with the highest number and runs it, allowing it to change the stack and buffers. The idea is the more tests that succeed the higher the score, with failures obtaining negative points. The significance of this is the marking of ungrammatical sentences; the ungrammatical part attains a low score since no grammar rules will match exactly the input. Using this system of numbering, PARAGRAM can indicate to the user the ungrammatical sentences i.e. those with a score below zero. It is also possible that PARAGRAM can tell why the sentence is ungrammatical because the rule which succeeded, but with a low score, should have matched exactly. PARAGRAM also includes all ungrammatical sentences but those of the type below it can parse:

1. Ann sold Sue book. 2. Tom wants go to the bus...
As was said in relation to YAP and L.PARSIFAL, PARSIFAL does not use phrase structure rules. Charniak also saw the need for reducing the complexity of the rules. He suggested that the explicit activation and deactivation of packets should be dropped so that phrase structure rules would be similar to their transformational equivalents. Phrase Structure rules also eliminate the need for a regular rule to do what in a transformational grammar would be done by the base component.

Charniak added base rules to PARAGRAM, thus creating rules which do not perform explicit tasks. Thus, the grammar need not have explicit rules for finishing the parsing of a constituent and only an occasional rightward movement rule e.g. there deletion, is needed to explicitly attach a constituent to one higher.

PARAGRAM cannot, as yet, automatically create rules for creation of new nodes. In both PARSIFAL's and PARAGRAM's grammars such rules have to be highly variable and currently there does not seem to be any way to predict their form simply from looking at the base rules. However, PARAGRAM has in its grammar only nine non-transformational rules whereas PARSIFAL has twenty one.

Hindle's Parser: Fidditch

Donald Hindle's parser Fidditch is based on the design of PARSIFAL (Hindle, 1983). It has similar data structures, i.e. an Active Node Stack and a call Locked Buffer, and also a grammar of pattern/action rules. Fidditch has been used in the areas of navy-military messages and spontaneous speech.

Hindle states that Fidditch was constructed to parse transcripts of spontaneous speech. (On top of Fidditch, Hindle attached an editing system, triggered by an editing signal, to deal with the self correcting errors which occur in spontaneous speech.) The results produced consist of a single analysis, which could be partial, of each sentence of the set of interview transcripts. When it is not possible to construct constituents from the subconstituents given, the parser passes on to the following constituent.

The words of the input sentences are given a lexical category or category as a result of dictionary look-up and morphological analysis. After lexical analysis the sentence is parsed as with other Deterministic Parsers; using the buffer and stack facilities.

The editing system is comprised of a set of self-correction rules which erase elements from the parser's view when an editing signal is found. The rules, of which there are two types, also work deterministically in conjunction with the stack and buffer. One type of rule (of which there are three) erases copies and the other type of rule (of which there is only one), is used to lexically trigger restarts. The editing signal is placed at the end of a constituent and can be detected by the buffer.

For the real-time application of parsing navy-military messages, Bachenko et al regard necessary for efficiency the imposing of a restriction on the parser (Bachenco et al, 1983). This restriction involves constraining the parser to examine only the left-most element of a constituent and its lexical category; this was given the name the left corner constraint.

Bachenco et al state that their work corresponds directly to Chomsky's G.B. theory. They indicated the constraint as imposed on verb complements and found that in the majority of cases that the left-most element of the complement carried enough information to describe the whole constituent. However in these other cases where the left-most element did not provide sufficient information, features and devices are required to aid parsing.

Current Work

Current work being done involves the investigation of the practicability and efficiency of using a Deterministic Parser to process the sublanguage of an aircraft maintenance manual for Machine Translation.

At present, a Deterministic Parser is being designed which will incorporate all of what we regard as the best features of a Deterministic Parser: Active Node Stack, Two Buffer Lockahead, a grammar of individual Pattern/Action rules corresponding to Base Phrase Structure rules and Transformational rules with the dotted rule being used in place of packets, and rule Priority.

Conclusion

In surveying these Deterministic Parsers, we have examined aspects of their implementation and use. Those parsers following PARSIFAL can be considered an improvement on the original; having been designed to correspond more with linguistic and grammatical principles. However, most of the parsers are used to mimic the Human Sentence Processing Mechanism. Current work hopes to prove that the Deterministic Parser also can be used for its engineering properties.

References


146