An Adaptive Parsing Technique for ProRules Grammar

Sri Fatimah Tjong[1], Nasreddine Hallam[1], and Michael Hartley[2]

[1] *School of Engineering and Computer Science*
University of Nottingham Malaysia Campus
Jalan Broga 43500 Semenyih, Selangor Darul Ehsan
Tel: +603-89248350, Fax: +603-89248001, E-mail: {kcx4sfj, Nasreddine.Hallam}@nottingham.edu.my

[2]*School of Mathematics*
University of Nottingham Malaysia Campus
Jalan Broga 43500 Semenyih, Selangor Darul Ehsan
Tel: +603-89248137, Fax: +603-89248001, E-mail: {Michael.Hartley}@nottingham.edu.my

Abstract

This paper discusses the proposed technique for analysis and production of English grammar rules by using the structure and meaning of human language. We come up with ProRules grammar that is based on grammar adaptation and compilation. ProRules serves as a basis scheme in processing the natural language requirements specifications. This paper will further outline the ideal algorithms for lexical scanning and parsing of the natural language requirements specifications. Since ProRules is designed to eliminate the occurrence of left recursion, an adaptive recursive descent parsing strategy is chosen to build the Parser. Therefore, to show the applicability and adaptability of our algorithms, we come out with Scanner and Parser prototypes system. We also apply some heuristic strategies in our parsing strategy to deal with lower level of natural language ambiguity.

Keywords:

ProRules, Requirements Engineering (RE), natural language requirements specifications (NLRSs), Scanner, Parser

Introduction

As far back as 1970s and as current as present dates [2, 7, 15], grammar compilation to speed up the parsing of natural language such as Augmented Transition Network (ATN) grammars has always been a challenging research topic. Parsing sentences of natural language such as English has also become significant work in NLP domain such as the work of Charniak's Statistical Parsing [8] and Klein and Manning' Unlexicalised Parsing [4]. These natural language parsers extract the syntactic structure of the natural language and return mostly one analysis per sentence.

Most of the parsers developed to date use Probabilistic Context-Free Grammars (PCFG) as their backbone formalism. We define ProRules as the newly designed grammar that is designated to assist the development of adaptive recursive-descent parsing prototype system. ProRules grammar is originated from Context-Free Grammar that generates a context-free language. Then, we adopt the programming-language compiler technology to compile Production-Rules (ProRules) grammars of NLRSs that will be used in RE domain. In a given well-formed language, a Compiler is normally used to translate a high-level (programming) language to a lower-level (programming) object language.

The question then arises the background reason of choosing this approach amongst others. Furthermore, recall that this compiler approach is rarely adopted in parsing NLRSs. One possible reason is because of the level of difficulty in parsing the language correctly based solely on its grammar transition rules. Another reason might be the programming complexity that one has to mastermind. Compiler approach is not something new for it has been an active topic of research and development since the mid 1950s and considered as a relatively mature computing technology. We believe that its underlying principles, multiple memory banks, and clustered architecture make it better than other approaches.

The paper is organised as follow. Section 2 discusses the background and brief overview of the NLRSs compilation phases. Section 3 briefs the parsing methodology adopted in this research. Section 4 describes discussion on the design and implementation characteristics of Scanner and Recursive-Descent Parser and finally Section 5 concludes the work.

Background

Context-Free Grammar (CFG), known as the Type-2 grammar in Chomsky hierarchy, generates a context-free language [13]. CFG is powerful in describing the syntax of programming language and we apply this concept in describing the syntax of NLRSs. We believe that CFG is powerful enough to describe most NLRSs structure and restricted enough to allow adaptive parsing. The CFG adopted in this research is represented by

$$EG = \{N, T, S, R\}$$

where:
EG: a tuple of the context-free grammar
N: a finite set of non-terminals
- Phrasal categories: DS, SS, NP, VP, etc.
- POS: N, VB, SPRP, etc.

T: a finite set of terminals
S: a start symbol, S ε N
R: a finite set of production rules

It is obvious that in order to parse NLRSs, a grammar is needed. Therefore, we design a new type of grammar coined ProRules, It originates from CFG and consists of a set of formal rules for structures allowed particularly in NLRSs. However, it is not only designed and compiled mainly for finding constituent labels of linguistic significance, but also to help in 'programming the parser'. ProRules is to be considered as a relatively broad-coverage grammar of the English language and is meant to express NLRSs in a simpler way. More importantly, it has the ability to manage the effects of grammar expansion by selectively filtering subsets of source grammar rules through specific compilation procedure.

An example of ProRules grammar shown below means the *Predicate* rule may consist of *Verb* or alternatively *VerbPhrase* followed by Complement and optional occurrence of Adverbial.

Predicate ::= (Verb|VerbPhrase) Complement [Adverbial *]

Conceptually, ProRules is developed from English Grammar found in [11] and presented in the form of BNF (Backus-Naur Form) and EBNF (Extended Backus-Naur Form). BNF is a notation used to represent the context-free grammars in a much more intuitive way whereas ENBF is a combination of Regular Expression (RE notations are like '|', '*', '(', ')', etc.) and BNF [6]. EBNF is powerful as it can improve the readability and conciseness of BNF in expressing the recursive rules so that the grammar is more understandable.

Using ProRules grammar, the NLRSs compilation process proposed in this paper involves the following phases:

- *Lexical and Syntactic Analysis*- the source language is transformed into a stream of tokens and each token represents a single atomic unit of the language. The subphases of Syntactic Analysis:
  - *Scanner* tokenises the source language based on their criteria such as noun, verb, identifier, operators, etc.
  - *Parser* parses the source language (represented by a stream of token) to determine the order of token and the language hierarchical structures.
- *Semantic Analysis*- the conformity of the meanings of the source language is checked to their corresponding contextual constraints
- *Language Generation*- the generation and optimisation of the target language.

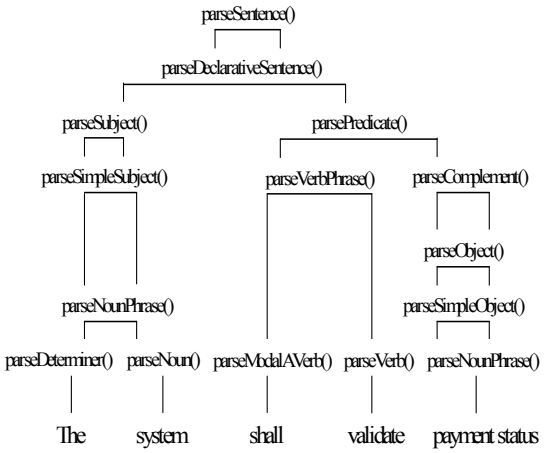
Note that the main concentration in this research is to implement the NLRSs compilation of the Lexical and Syntactic Analysis phases and further up with Semantic Analysis phases.

Parsing Methodology

Top-down and Bottom-up parsing algorithms are probably most preferable amongst the wide variety of parsing algorithms available. However, parsing using Probabilistic Context Free Grammar (PCFG) either lexicalised or unlexicalised [4, 8] have as well, gained certain popularity in NLP domain.

Parsing a given NLRSs with respect to ProRules is the process of determining whether NLRSs belongs to the language specified by ProRules and, if so, finding all the structures that the ProRules pairs with the sentence. To parse ProRules, an adaptive recursive-descent parsing technique based on the compiler-approach has been implemented by using the Java programming language.

Observations on the parsing algorithms motivate the choice of Recursive-Descent parsing [6] algorithm. Recursive-Descent parsing is believed to employ a strategy that is helpful in matching various grammar production rules. If a match is not met, then the parser will back up to the position it was when it attempted to match the failing rule [17]. In order to work with the recursive-descent parsing concept, we design the ProRules grammar to be mutually recursive and prevent the possibility of left recursion. An adaptive recursive-descent parser for an English Grammar *EG* consists of a group of methods *parseX()*, where *X* is a non-terminal symbol *N*. An instance of this adaptive parser



applied on a simple NLRS is depicted in Figure 1.

Figure 1. Recursive-descent parsing of a natural language requirement

Discussion

An experimental scanning and parsing prototype systems

based on the ideas presented here have been implemented in Java. They are made possible by involving both WordNet® [9] and some of the Penn Treebank tagsets [12].

Tags

This section contains a list of alphabetical part of speech (POS) tags and the parts of speech corresponding to them. When the scanner read the token and verified by the parser, the parser will label the token with its corresponding tag.

1.	INTLITERAL	0,1,..,9
2.	CHARLITERAL	a,..,z and A,..Z
3.	IDENTIFIER	(this tag will be returned in case of mistyped input)
4.	OPERATOR	+, -, ..
5.	MV	Modal Auxiliary Verbs
6.	PV	Primary Auxiliary Verbs
7.	SPRP	Subjective Personal Pronoun
8.	OPRP	Objective Personal Pronoun
9.	PPRP	Possessive Personal Pronoun
10.	PRP$	Possessive Pronoun
11.	RFPRP	Reflexive Pronoun
12.	DPRP	Demonstrative Pronoun
13.	IPRP	Indefinite Pronoun
14.	ITPRP	Interrogative Pronoun
15.	RLPRP	Relative Pronoun
16.	RCPRP	Reciprocal Pronoun
17.	RB	Adverb
18.	JJ	Adjective
19.	CC	Coordinating Conjunction
20.	SC	Subordinating Conjunction
21.	DT	Determiners
22.	WHQ	Wh-Questions
23.	IN	Preposition
24.	NN	Noun
25.	VB	Verb Infinitive
26.	VBS	Verb Singular
27.	VBP	Verb Past Tense
28.	VBPP	Verb Past Participle
29.	VBG	Verb Present Participle
30.	TO	to
31.	EXCLAMATION	!
32.	STAR	*
33.	DOT	.
34.	SEMICOLON	;
35.	COMMA	,
36.	LPAREN	(
37.	RPAREN	)
38.	LBRACE	{
39.	RBRACE	}
40.	SEPARATOR	|
41.	VARIABLE	_
42.	QUOTATION	‘
43.	QUESTION_MARK	?
44.	ALOC	@
45.	HASH	#
46.	DOLLAR	$
47.	PERCENTAGE	%
48.	AND	&
49.	COLON	:
50.	DOUBLE_QUOTATION	“
51.	BACKSLASH	\
52.	FORWARD_SLASH	/
53.	REF_LBRACE	[
54.	REF_RBRACE	]
55.	EOT	End of Text
56.	ERROR	

Figure 2. List of part-of-speech tags

In a case where an unidentified or mistyped token are inputted and scanned, the parser will label it as an IDENTIFIER tag, which can be seen from the list.

Scanner

The principal purpose of having a scanner system is to recognise lexical item in the given text. Analogous to parsing process, scanning works at a finer level of detail. Unlike the parser that groups all the tokens into large phrases or sentences, a scanner scans the individual characters and discards separators (such as blank space), which are then to be grouped into lexicons.

The main processes of a scanner are tokenisation and dictionary look-up. Tokenisation signifies the recognition of each token from a given language and Dictionary look-up fetches the tokenised string, checks whether it exists and returns its POS information.

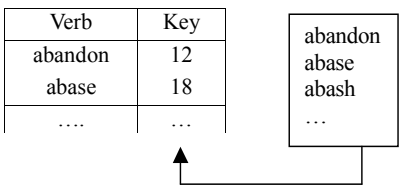
To capture each token, we discuss some of the algorithms that are used in developing the Scanner as follows:

- Private *scan* method (Algorithm 1.) is used to scan and discard any *Separator* tokens and return the token that follows them.

```
{tokenKind and start_position are Integer type
where start_position := 0,
currentScanningToken is a Boolean type and
separators are white spaces, new lines, etc.}
currentScanningToken := false;
while separators do begin
   scanSeparator
currentScanningToken := true;
start_position := current line of source file
tokenKind := scanToken
finish_position := current column of source file
end
{scanSeparator and scanToken are
respectively methods}
```

Algorithm 1. *scan()* method

- Lists of words taken from WordNet and grouped from Penn Treebank lexicon such as Noun, Verb, Adjective, Adverb, Determiner, etc. are saved in the Hash-Tables designated for each POS category (as shown in figure 3. and 4.).



Streams of words obtained from ‘Verb’ word-sets (WordNet®) are saved in the ‘Verb’ HashTable

Figure 3. Verb HashTable

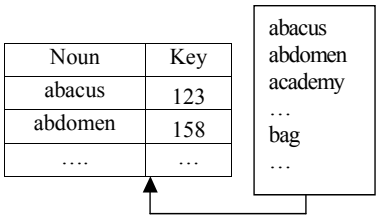


Figure 4. Noun HashTable

- Each of the ProRules grammar is converted into its scanning method as illustrated in Algorithm 2.

```
{currentChar is a Char type, sb is a StringBuffer
type and str is a String type}
switch currentChar begin
   for all case 'a' – case 'z' or
   case 'A' – case 'Z' do
         sb := append(currentChar)
         takeIt
   while currentChar isLetter or currentChar
   isDigit do
         sb := append(currentChar)
         takeIt
   end
   str := toString(sb)
   if tableVerb containsKey(str) is true then
         return verbToken
   else if tableNoun containsKey(str) is true then
         return nounToken
   else if tableAdjective containsKey(str) is true
   then
         return adjectiveToken
   else if tableAdverb containsKey(str) is true
   then
         return adverbToken
   end
end
{tableVerb, tableNoun, tableAdjective and
tableAdverb are HashTable types, takeIt is a
method type}
```

Algorithm 2. *scanToken()* method

The usefulness of a scanner for syntactic parsing is a question little addressed in the literature. Scanner is somewhat similar to a tagger in term of its task in suggesting possible lexical category to a given token. In case of superfluous or ambiguity analyses turn out, the parser will eventually decide their appropriate tags. We observe that by having a scanner, it helps to prepare the input token and return the suggested lexical category to the parser whenever a parser is to read the next input token. Finally the parser will then check ProRules grammar whether this lexicon category conform to the grammar.

An Adaptive Recursive-Descent Parser

The general idea of parsing with ProRules grammar is to start generating possible tree structures until a rule generates a lexical category. With ProRules, the parser is forced to travel only the nodes that are defined in the grammar. This is then checked with the next word in the sentence. If it is of the appropriate lexical category, the parse continues. However, if it is not of the appropriate lexical category, the parser will explore another node in the search space.

Before parsing, it will be convenient to view the source or given text as a stream of lexicons or symbols such as operators, literal, punctuation, etc. since the source text actually consists of individual of characters, and a lexicon consists of several characters. Thus, it is a significant help to have a scanner to group the characters into lexicons and to discard other text such as blank space. Practically, the process of parsing has been supported by a Scanner.

Nodes in parse trees are labeled with the name of the ProRules licensing the local tree rooted at that node. Being able to view the parse trees and displaying a tree from the perspective of the production rules associated with the nodes help to facilitate the parsing process and show whether right or wrong rules have been explored. Algorithm 3 shows how ProRules grammar is used to construct the parser.

```
begin
 parseVerb
 if parseVerb doesn't return a matched
 result then
    parseVerbPhrase
 parseComplement
 for any adverbials occurred after
 parseComplement do
   parseAdverbial
end
{parseVerb, parseVerbPhrase,
parseComplement and parseAdverbial are
respectively parsing methods}
```

Algorithm 3. *parsePredicate()* method

The rules inside ProRules grammar are built into methods which later cooperate to parse the given input NLRSs. With scanner's help in returning the possible lexical category to parser, it would then know which rule (or method) to travel. Algorithm 4 illustrates how *parseDeCSent* method calls *parseSubject* and *parsePredicate*, one after another, to parse the subject and predicate respectively.

```
Command parseDecSent() throws SyntaxError {
   Command dcAST = null;
   SourcePosition dcPos = new SourcePosition();

   start(dcPos);
   Attribute sbj = parseSubject();
   Attribute pdc = parsePredicate();
   finish(dcPos);

   return dcAST;
}
```

Algorithm 4. Method of *parseDecSent*

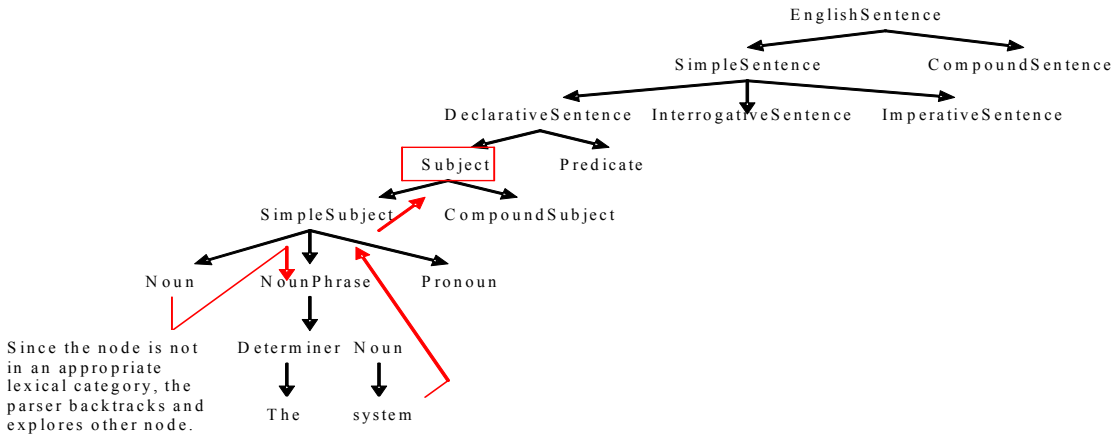


Figure 5. The Flow of Recursive-Descent Parsing with all the possible tree structures

Results

For evaluation, we apply some heuristic strategies to deal with certain types of ambiguity and then use the parser to parse 50 samples of NLRSs. The heuristic strategies referred are disambiguation rules written by the authors to analyse the corpora with a reasonably high precision. Recall the domain scope in which the Recursive-Descent Parser is intended to parse is RE even though ProRules covers most of the English Grammar.

This present method makes it possible to prune the morphological and syntactic ambiguities in running simple NLRSs. If the NLRSs retain the correct morphological and syntactic reading within the RE domain's scope, the overall parsing success rate may vary from *80% - 100 %*.

One worth noting critic on the recursive descent parsing technique is its failure to handle left recursion. We are aware of this limitation and utilise programming tactics to handle it such that the parser will parse the input given adaptively. Each (non-terminal) rule of ProRules was programmed as a method and each method would only be called according to the given rule. To avoid the recursion, whenever any rule (especially the left most rule) isn't able to return the appropriate lexicon category, it will exit the current rule/method and makes call to the next adjacent rule.

Conclusion

Progress is being made in NLRSs parsing but there is still a long way towards Natural Language Understanding (NLU). We discussed and showed that compiler-approach can be used to build an Adaptive Recursive-Descent Parser which has worked and comparatively good. Here, the adaptive recursive-descent parsing is an inherently non-deterministic process. In constructing a derivation, ProRules is applied to the sentential form and the parsing process terminate when it finally produces a derivation of the input string (the lexicon tagging). In an unlikely event when an incorrect derivation has been made, an algorithm implemented inside the parser enables the ability to backtrack and generate alternative derivation.

To support the parsing process, we developed a scanner that is mainly used to tokenise each inputted token and then check whether the token's existence in the dictionary (whether it is a morphologically correct lexicon). Improvements might be achieved by adding more disambiguation rules and involving the semantic rules. Hence, future work will concentrate on the continuation and enhancement of the semantic analysis of the parser in processing the language.

Acknowledgements

This work receives contributions of advices and discussions from Mohammad Saleh, Sarah Aw and Xu Xiang Dong.

References

[1] Aho, A. V., Sethi, R., and Ullman, J.D., (1986). *Compilers Principles, Techniques, and Tools,* © Bell Telephone Laboratories, Incorporated

[2] Stehno, B., and Retti, R., (2003). *Modelling the logical structure of books and journals using augmented transition network grammars,* The Emerald Research Register

[3] Klein, D., and Manning, C. D. (2001). Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank pp. 330-337 ACL 39/EACL 10

[4] Klein, D., and Manning, C. D. (2003). Accurate Unlexicalized Parsing pages 423-430, ACL 41

[5] Klein, D., Manning, C. D., Levy, R., Grenager, T., and Andrew, G. (2004) Stanford Parser, Stanford Natural Language Processing Group, Stanford University

[6] Watt, D. A., and Brown, D. F. (2000) *Programming Language Processors in Java*, © Pearson Education Limited

[7] Duchier, D., Roux, J. L., and Parmentier, Y. (2004) The Metagrammar Compiler: An NLP Application with a Multiparadigm Architecture, LORIA

[8] Charniak, E. (1997) Statistical parsing with a context-free grammar and word statistics, *In Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 598-603

[9] Miller, G. A., Fellbaum, C., Tengi, R., Wolff, S., Wakefield, P., Langone, H., and Haskell, B. 2006. *WordNet® 2.0* Cognitive Science Laboratory Princeton University. Citing Internet sources URL http://wordnet.princeton.edu/

[10] Eisner., J. and Satta, G. (1999) Efficient parsing for bilexical context-free grammars and head automaton grammars, *Association for Computational Linguisitics 37*, pp. 457-464

[11] Seely, J. (2001) *Oxford Everyday Grammar,* Oxford University Press Inc., New York

[12] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. (1993) Building a large annotated corpus of English: the Penn Treebank, *Computational Linguistics, vol.19*

[13] Chomsky, N. (1959). On certain formal properties of grammar, *Information and Control 1*, pages 91-112

[14] Rus, T. (2001). *A Unified Language Processing Methodology* , Elsevier Science

[15] Woods, W. A., (1970). *Transition Network Grammars for Natural Language Analysis,* Computational Linguistics

[16] Ambriola, V. and Gervasi, V. (1997). Processing Natural Language Requirements, *Proceedings of the 1997 International Conference on Automated Software Engineering*

[17] Diggins, C. 26 October 2005. YARD (Yet Another Recursive Descent Parser). Citing Internet sources URL http://www.ootl.org/yard/