

Context-free Grammar Induction using Genetic Programming

F. Javed[†]

B. R. Bryant[†]

M. Črepinšek[‡]

M. Mernik[‡]

A. Sprague[†]

[†] Department of Computer and Information Sciences
University of Alabama at Birmingham
1300 University Boulevard Birmingham, AL 35294-
1170, USA

{javedf, bryant, sprague}@cis.uab.edu

[‡] Faculty of Electrical Engineering and Computer
Science

University of Maribor

Smetanova 17, 2000 Maribor, Slovenia

{matej.crepinsek, marjan.mernik}@uni-mb.si

ABSTRACT

While grammar inference is used in areas like natural language acquisition, syntactic pattern recognition, etc., its application to the programming language problem domain has been limited. We propose a new application area for grammar induction which intends to make domain-specific language development easier and finds a second application in renovation tools for legacy systems. We use the genetic programming approach for grammatical inference. Our earlier work used grammar-specific heuristic operators in tandem with non-random construction of the initial grammar population and succeeded in inducing small grammars. We extend that work and propose the use of derivation trees and syntax graphs in order to be able to infer a more comprehensive set of context-free grammars.

Categories and Subject Descriptors

D.3.4 [Programming Languages]: Processors – *compilers, code generation and parsing.*

General Terms

Languages, Theory

Keywords

Context-free grammars, genetic programming, grammar-induction.

1. INTRODUCTION

Making domain-specific language development easier for domain-experts not versed in programming language design is an open-problem in the area of domain specific languages. While many possible approaches can be taken to build a domain specific language, our work advocates the use of an evolutionary approach, more specifically genetic programming, to context-free grammar induction. Genetic programming (GP) is a successful technique for getting computers to solve problems automatically and finds application in domains ranging from robotic control to data mining. GP has been shown to successfully evolve methods

working on typical data [1], but comes up short when larger programs (e.g. compilers) are desired. It has been shown that formal language specifications can be used to generate a complete compiler/interpreter [2]. Although such specifications for general purpose languages are still too large, for domain-specific language specifications the GP approach can be expected to find a successful solution.

2. RELATED AND PREVIOUS WORK

Grammar induction (or grammar inference or language learning) is the process of learning of a grammar from training data. Various algorithms exist for learning regular languages, which represent the largest class of languages which can be efficiently learned. By comparison, context-free grammar (CFG) learning requires more information than a set of positive and negative samples (e.g. a set of skeleton parse trees) which makes them a bigger challenge for grammatical inference. Wyard [3] explored the impact of different grammar representations and experimental results show that an evolutionary algorithm using standard context-free grammars (BNF) outperformed other representations. This performance differential was attributed to the larger grammar search space of the other representations, which was a consequence of them having a more complex grammar form.

Our work is related to renovation and legacy systems where renovation tools can be rapidly built once a grammar is available. In this section we present a short overview of our previous work [4] in inferring CFG's using the GP approach. In genetic programming, a program consists of a terminal set T and a user-defined function set F , which in our case consisted of terminal symbols defined with regular expressions and non-terminals respectively. The encoding of a grammar into a chromosome was done using direct encoding as a list of BNF production rules as suggested in [3]. The GP system was populated using the algorithm defined in [4], and additional control parameters were introduced to prevent the grammars from becoming too large. Specific one-point crossover, mutation and heuristic operators were proposed as genetic operators. The heuristic operators *option*, *iteration* * and *iteration* + were chosen for the mutation operation. At the end of each generation, an LR(1) parser was generated for each grammar in the population using the compiler generator tool LISA[5]. The parser was then run on fitness cases and a fitness value derived using a grammar fitness function. Experiments performed in [4] show that with the described approach, CFG's could be inferred albeit the number of productions were small. On examples where the underlying CFG was bigger, this approach was not successful.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACMSE'04, April 2–3, 2004, Huntsville, Alabama, USA.

Copyright 2004 ACM 1-58113-870-9/04/04...\$5.00.

3. NEW IDEAS

3.1 The Need for an Augmented Approach

It was observed in [6] that heuristic operators greatly improved the search and induced better grammars – without them the search was not successful. The current set up uses a slightly modified heuristic operator set in that the sequence of right-hand side (RHS) symbols can appear optionally or iteratively. However, induction of bigger grammars remained elusive. A case in point was finding context-free grammars for simple arithmetic expressions and simple assignment statements where the RHS could be just a number. The approach failed to find a solution when the both the sub-languages were combined.

3.2 Syntax Graphs and Derivation Trees

Upon closer analysis of our results, it became clear that the randomly generated initial grammar population was an impediment for the induction process. Instead, it was decided to exploit knowledge from positive samples and generate a few derivation trees by simple composition of consecutive symbols. During this process a sequence of non-terminal symbols which appear iteratively can be detected and an appropriate grammar can be constructed. After this modification to the initial grammar population construction, the remaining GP system remains the same. We were able to induce a correct CFG for our arithmetic expression example using this simple enhancement.

If the CFG's right-hand sides are viewed as a collection of regular sets connected by union, concatenation and recursion, then we believe that inferring more general CFG's is possible.

Table 1. Representing CFG's as Regular Expressions

Context-Free Grammars	CFGs as regular expressions
$E \rightarrow E + T \mid T$	$E \rightarrow T (+ T)^*$
$T \rightarrow T * F \mid F$	$T \rightarrow F (* F)^*$
$F \rightarrow (E) \mid id$	$F \rightarrow (E) \mid id$

A syntax graph is a convenient data structure unifying regular sets and CFG's - the terminal symbols are enclosed in circular boxes, the non-terminals (i.e., calls to other syntax graphs, possibly recursive) in rectangular boxes and connections between vertices are via edges, possibly cyclic. **Table 1** and **Figure 1** represent the traditional grammar for arithmetic expressions (e.g. [7]) using our new formulations.

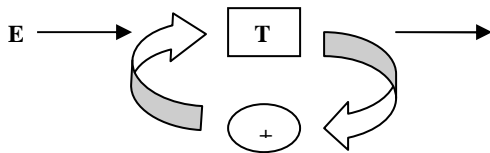


Figure 1. A syntax graph for rule $E \rightarrow T (+ T)^*$

Our proposed approach is to identify regular sublanguages of the context-free language (CFL) in question and compose these

languages using the three aforementioned combination operations. The syntax graph data structure is particularly useful for detecting regular sublanguages (which are sub-graphs) and some context-free sublanguages (although the general problem of CFL containment is undecidable, we may represent CFL's defined by different syntax graphs by non-terminal variables and assume equivalence of variables only if they have the same name). Different precedence structures for operators may also be detected using the graph representations. As an example of these concepts, we may infer patterns $id + id$ and $T (+ T)^*$. It is straightforward to determine that the first pattern is included in the second. For more complicated examples, consider $T + E$ and $(T +)^* T$. In fact, the former case reduces to the latter by replacement of right recursion by repetition. Note that the equivalence of these is based on syntactic correctness, and precedence and associativity of operators are not explicitly taken care of at this time. We expect that some knowledge about operators will be necessary to properly structure the grammar for correct precedence and associativity.

4. CONCLUSION AND FUTURE WORK

Grammatical Inference finds application in many programming languages related problems like renovation problems. However learning CFG's is still a real challenge in grammatical inference. We extend previous attempts at learning CFG's using evolutionary algorithms [3] by providing grammar-specific heuristic operators and better construction of the initial population. Our research aims are to not only infer syntax from grammar inference, but also the semantics from a given set of data, as well as to automatically generate a complete compiler from examples.

5. REFERENCES

- [1] Langdon, W.B., *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, Kluwer Academic Publishers, 1998.
- [2] Henriques, P., Varanda Pereira, M., Mernik, M., Lenič, M., Avdičaušević, Žumer, V., *Automatic generation of Language based Tools*, Electronic Notes in Theoretical Computer Science, Vol. 65 No.3, Elsevier Science Publisher, 2002.
- [3] Wyard, P., Representational Issues for Context-Free Grammar Induction Using Genetic Algorithm in *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications, Lecture Notes in Artificial Intelligence, Vol 862, pp. 222-235, 1994*
- [4] Mernik, M., Gerlič, G., Žumer, V., Bryant, B., Can a Parser be Generated from examples?, *Proceedings of the ACM Symposium on Applied Computing*, pp 1063 – 1067, 2003.
- [5] Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V., LISA: An Interactive Environment for Programming Language Development, *11th International Conference on Compiler Construction, CC'2002, Lecture Notes in Computer Science, Vol. 2304, pp. 1 – 4, 2002.*
- [6] Mernik, M., Črepinšek, M., Gerlič, G., Žumer, V., Bryant, B.R., Sprague, A., *Learning Context-Free Grammars using an Evolutionary Approach*, submitted for publication
- [7] Aho, A.V., Sethi., R., Ullman., J.D., *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.