Applying Grammar Inference Principles to Dynamic Analysis

Neil Walkinshaw and Kirill Bogdanov
Department of Computer Science, The University of Sheffield, UK
E-mail: {n.walkinshaw, k.bogdanov}@dcs.shef.ac.uk

Abstract

Grammar inference and dynamic analysis share a number of similarities. They both try to infer rules that govern the behaviour of some unknown system from a sample of observations. Deriving general rules about program behaviour from dynamic analysis is difficult because it is virtually impossible to identify and supply a complete sample of necessary program executions. The problems that arise with incomplete input samples have been extensively investigated in the grammar inference community. This has resulted in a number of advances that have produced increasingly sophisticated solutions that are much more successful at accurately inferring grammars. This paper investigates the similarities and shows how many of these advances can be applied with similar effect to dynamic analysis problems.

1. Introduction

Techniques that are based upon dynamic program analysis are appealing because of their inherent precision. A dynamic technique is supplied with a collection of program traces (a trace records the state of a program throughout its execution), and this concrete information can be used as a basis to make precise conclusions about the program behaviour. However, these conclusions are only valid with respect to the set of supplied traces and do not necessarily generalise to every possible program execution [10].

Dynamic analysis results have to be interpreted with a degree of skepticism. They can only be regarded as representative of general program behaviour (regardless of input or environment) if the supplied set of program traces is 'complete', i.e. it provides a total coverage of program behaviour, as is the purpose of functional test sets. Depending on the complexity of the program, this set of necessary program traces can be extremely large. If the additional assumption is made that the developer has no prior familiarity with the program (which is probable in domains such as program comprehension), the precondition that she can provide a set of traces that is complete becomes unreasonable. Instead, dynamic analysis techniques are commonly provided with an incomplete set of traces, in the hope that they are sufficient to result in a model that is an approximation of general program behaviour.

This problem is not unique to dynamic analysis. Grammar inference is an example of another field that is subject to a similar weakness. Here the challenge is to identify the grammar of a language by analysing a sample of sentences that belong to (and optionally do not belong to) it. The sample of sentences can often be sparse, which means that the resulting grammar is inevitably only partial or even false. However, a substantial amount of research in grammar inference has focused on minimising this problem, and has largely achieved this by using simple solutions, such as the provision of negative strings as well as positive ones to avoid unsound results, and the use of oracles to guide inference by answering simple questions about system behaviour.

This paper looks at some of the similarities between grammar inference and dynamic analysis. It presents some of the established theoretical limits on solutions to grammar inference problems, which also apply to dynamic analysis techniques that infer models equivalent to deterministic finite automata (this is the case for a large proportion of dynamic analysis techniques). It also shows how certain principles that have been successful in increasing the reliability of inferred grammars can as easily be used to improve the soundness of dynamic analysis results.

2. Regular Grammar Inference and its Limits

This section introduces the grammar inference problem, provides an overview of some inherent limits on certain traditional approaches, and introduces some of the most successful recent solutions. It does not provide an in-depth overview of the underlying mechanics of grammar inference techniques. The purpose of this section is to provide a high-level view of some of the key insights that have led to the most substantial advances in the field. For a more comprehensive overview, there are recent authoritative surveys

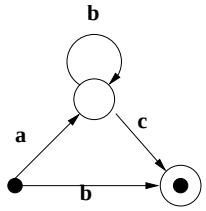

Figure 1. DFA that recognises the grammar corresponding to (ab*c)+b

by Parekh and Honavar [17] and De la Higuera [7].

2.1. The Grammar Inference Problem

The problem of grammar inference (also known as grammar induction) was formalised by Gold [11] in 1967. It is concerned with the identification of a language grammar from a finite sample of valid and (optionally) invalid sentences. Regular grammars have received the greatest amount of attention from the inference community. They are the simplest form of grammar in terms of Chomsky's hierarchy [4], and can be equivalently represented as a Deterministic Finite Automaton (DFA), where transitions are labelled by words and the automaton represents every valid ordering of those words in a sentence. The DFA representation is particularly appealing because [17]: (a) DFAs are easy to understand and (b) there exist several efficient DFA algorithms that are useful for a number of inference techniques (such as minimisation, determining the equivalence of two DFAs, and determining whether the language produced by one DFA is the super set of the language produced by another).

To illustrate the regular grammar inference problem, figure 1 illustrates the DFA for a simple regular grammar. Positive samples of the grammar correspond to sequences that would be accepted by the machine, and negative samples correspond to strings that would be rejected. As an example, a positive sample could consist of $\{abbbbc, b, abc\}$ and a negative sample could consist of $\{c, aba, ba\}$. Given that we do not have any prior knowledge of the structure of the DFA, a grammar inference technique would attempt to guess it, given only the sets of positive and negative sequences.

2.2. Traditional Solutions and their Limits

The regular grammar inference problem has been shown to be NP-complete in general [1, 12] and was even compared to problems such as breaking the RSA cryptosystem [13]. However, since Gold's initial research into the subject, a number of techniques have emerged that can correctly infer a grammar in polynomial time by placing restrictions on certain factors, such as making assumptions about the initial sample of sentences, or adding oracles that can provide additional information to the inference algorithm. Some of those advances have been prompted by a substantial body of theoretical work that establishes the inherent computational limitations on particular solutions to the grammar inference problem. As an example, Gold [11] proved that (in the worst case) an inference algorithm will require an infinite number of positive input sentences to determine the target grammar – thus establishing that any practical finite solution would require some quantity of negative sentences as well.

Subsequent research by Angluin [2] proved that, given a random initial sample of positive and negative sentences, the exact language can be inferred in polynomial time by using a 'minimally adequate teacher'. This teacher is expected to answer two types of queries:

(1) If the technique suggests sequences that do not already belong to the initial positive or negative samples, the oracle can state whether or not they are valid in the target grammar - these are called *membership queries*.

(2) If the technique produces a suggested grammar (i.e. arrives at a tentative DFA), the oracle can determine whether it is the target DFA or not - these are called *equivalence queries*.

Angluin's minimally adequate teacher, in particular the requirement to be able to answer equivalence queries, is not always practical for certain application domains. The ability to answer equivalence queries implies a substantial amount of prior knowledge of the grammar (or state machine). If, for example, the teacher is a human and the state machine consists of hundreds of states and transitions, the process of establishing whether the inferred hypothesis corresponds to the human's impression of a correct machine becomes unrealistic. Besides the practicalities of comparing two grammars, there is the additional problem that the approach presumes that the oracle already has a prior knowledge of the system.

An alternative approach that does not necessarily depend on an oracle is to adopt the *state merging* approach, originally developed by Trakhtenbrot and Barzdin [18]. These broadly operate by using the sets of positive and negative samples to produce an *augmented prefix tree acceptor* - a state machine that represents exactly the samples of positive and negative sequences provided. This is illustrated in figure 2. The inference process then consists of iteratively merging pairs of nodes in the tree, producing a more general language acceptor in the process. The challenge lies in *not* merging pairs of nodes that would represent different states in the target DFA, because this would produce a machine that is too general (unsound). A worklist of node pairs to be merged is created by traversing the prefix tree in a structured fashion (e.g. breadth-first search), and this list is processed

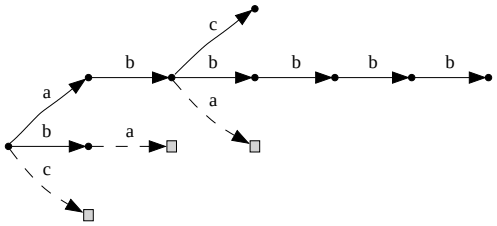

Figure 2. Augmented Prefix Tree Acceptor for positive sequences abc,abbbbb,b and negative sequences aba,c,ba

in order until a merge is found that is consistent with the supplied sample of sequences. The original state merging approach by Trakhtenbrot and Barzdin requires a complete sample of all strings up to a particular length to correctly infer the exact grammar in polynomial time.

Subsequent work by Oncina and Garcia [16] resulted in the RPNI (Regular Positive Negative Inference) algorithm. Instead of requiring an exhaustively complete set of samples up to a given length, the RPNI algorithm will exactly identify the target DFA in polynomial time, provided that the samples contain a *characteristic* sample of the target grammar. Informally, this means that, given the minimal DFA for some target language, a sample is characteristic if it contains positive sequences that cover every transition, as well as negative sequences that differentiate between any pair of non-equivalent states (see Dupont [8] for a formal definition).

Although the RPNI algorithm does not require an oracle, the requirement for a characteristic sample of the target grammar can be deemed just as impractical. Depending on the size and complexity of the target machine, the number of required sequences in the characteristic sample can be extremely large. As is the case with answering equivalence queries, constructing a characteristic sample requires a substantial amount of prior knowledge about the underlying system, ultimately rendering the requirement for a characteristic sample unrealistic for a large number of practical applications.

2.3. Pragmatic Solutions

The results for the techniques presented above are generally discouraging. Only if there exists an oracle that invests a substantial amount of effort, or the initial positive and negative sentence samples are sufficiently abundant that they happen to include an exhaustive or characteristic sample, is it possible to identify the exact target DFA in polynomial time. In the majority of practical applications these assumptions are unrealistic. However, recent techniques produce vastly improved results by relaxing these assumptions. They accept that the supplied set of samples might only be sparse, and apply various heuristics to identify states that might be equivalent. They accept that the resulting grammar might be only approximate, but ensure that it is at least an accurate approximation.

Traditional state-merging techniques fail when the supplied sample is sparse. They use simplistic techniques to construct work lists of possible state merges. If the provided sample of sentences is not complete or characteristic, there is not going to be enough information to prevent a wrong merge from happening, and there is a high probability that the algorithm will generate an erroneous machine as a result. At any point in the traditional merging algorithm, a worklist of candidate state merges is generated by simple strategies such as carrying out a breadth-first search of the successor tree to a node. This creates an arbitrary list of merge candidates that will only not be merged if there is a negative string to show that they are not equivalent. If this negative string is not present, a false merge occurs that in turn produces an incorrect state machine. Subsequent merges compound the error, resulting in a highly inaccurate final machine. Lang [14] reinforces this point by demonstrating empirically that for a sparse (incomplete / non-characteristic) sample of sequences, a traditional merging algorithm will only *approximately* identify the correct target machine if the size of the (random) sample is exponential in the size of the target machine.

A number of authors have realised that the key to improving the performance of state merging algorithms, particularly in the case of sparse samples, is to improve the way candidate pairs of nodes are selected. This has resulted in a number of heuristic approaches - the most popular of which is Price's Evidence-Driven State Merging (EDSM) algorithm [15]. The EDSM algorithm constructs a list of possible merges in a two-step process. The first step compares every possible pair of states and assigns a 'similarity' score to each pair, which indicates how much evidence there is in the sample to suggest that the pair are equivalent. This is computed by counting the number of overlapping outgoing labels in the prefix tree. An invalid merge (where a suffix from one state leads to an accept state but the identical suffix from the other state leads to a reject state) results in a negative score. The second step merges the highest scoring pair and the search process is restarted on the merged machine. Unlike the arbitrary merge sequences produced by traditional state-merging approaches, the merge order for the EDSM approach depends on the characteristics of the supplied samples. The EDSM approach was originally developed as a winning entry to the Abbadingo competition, which posed various grammar inference challenges for large random target machines with sparse data sets, where it excelled at identifying the largest class of machine (~512 states).

The EDSM algorithm can arrive at a close approximation to the target DFA even if the initial sample of sequences is not complete or characteristic. Nonetheless, the requirement for a relatively large number of samples is still inevitable. Dupont realised that it is easier to supply these samples iteratively, by employing an oracle. His Query-driven State Merging (QSM) approach [6, 9] is equipped with a question generation component that generates questions with the aim of guiding the oracle towards providing a (complete) characteristic sample. The algorithm uses the same heuristics as the EDSM algorithm to select states to merge, and for every merge the question generator produces a set of sequences that would have been valid *if* the merge was correct. These are posed to the oracle, and are then added to the appropriate positive or negative sample sets. Dupont shows that, through its targeted acquisition of necessary samples, the QSM technique arrives at an accurate solution with a much lower amount of initial sample sequences than the EDSM algorithm.

3. Augmenting Dynamic Analysis

There is an obvious overlap between the problems that are addressed in the grammar inference and dynamic analysis communities. Both attempt to derive facts or models from a finite sample of observations. (Regular) grammar inference aims solely to infer a DFA, whereas dynamic analysis has a broader range of applications. This paper argues that dynamic analysis has not taken advantage of the aforementioned advances in the field of grammar inference. If the purpose of dynamic analysis is to discover a model that is equivalent to a DFA, the analysis problem can be recast as a grammar inference problem, and the solution can take advantage of the many advances that have made regular grammar inference more tractable. Even if the target of dynamic analysis is not to produce a DFA-equivalent model, there are still many sufficiently general principles that can be applied regardless of the target model.

This analogy between grammar inference and dynamic analysis was first realised over 30 years ago when Biermann and Feldman [3] proposed their $k - tails$ state-merging algorithm that could generate state machines from sample executions. Since then a number of papers with similar aims have emerged that explore the analogy (e.g. Cook and Wolf [5]). However, these papers invariably restrict themselves to a rigid dynamic analysis framework that does not permit them to take advantage of some of the substantial advances that have taken place in grammar inference.

The conventional dynamic analysis framework assumes the provision of a single random selection of execution traces, from which the analysis technique must generate a hypothesis model in a single step. Invariably, the implicit assumption is made that this set of traces is in some sense 'representative'. Given that in the conventional case dynamic traces provide no negative data, it cannot be a characteristic sample (which necessarily contain negative sequences to distinguish between states). The only alternative requirement on the sample that can guarantee an exact result is that *every* positive sample up to a given length is provided, which in the case of dynamic analysis becomes impossible for any non-trivial system. Consequently, the result of a dynamic analysis is inevitably an approximation of the target system which, given only positive samples, is inherently prone to over-generalisation.

In practice, this means that the model that is presented by such an analysis will exactly show some set of rules that govern the provided set of program traces, but will not be able to make any useful inferences about the general system behaviour; it cannot infer impossible behaviour and, due to a combination of insufficient negative information and incomplete set of samples, any steps that attempt to infer behavioural rules beyond the supplied set of samples will probably be false. However, a number of the advances that have vastly improved the performance of grammar inference algorithms can be adapted to dynamic analysis. These are summarised below:

(1) Negative samples: Dynamic traces (positive samples) need to be accompanied by negative samples if the model inference is to be accurate and efficient. Negative samples can be obtained by a variety of techniques, such as proposing test sequences (if they fail / cannot be executed then they are negative) or by static analysis techniques which, although often too conservative to produce accurate positive input, are a powerful means of determining what is *impossible* in a software system, and are therefore a useful source of negative input.

(2) Accuracy wrt. incomplete traces: It can often be analytically shown that a dynamic analysis technique will produce an accurate model of the target system *if* the provided set of traces is complete. However, the ability of a dynamic analysis technique to cope with incomplete sets of traces is rarely evaluated. It is only through such empirical evaluation (often in the context of competitions such as the Abbadingo competition [15]), that real progress has been made in the field of grammar inference. These have produced the advanced heuristic techniques, such as EDSM, that excel in the average case, by making the most out of the sparse samples that they are given.

(3) Active dynamic analysis: Dynamic analysis, particularly in the context of program comprehension, is not necessarily restricted to passive techniques (which expect a single initial set of traces and produce a result in a single step). Oracles are available that can iteratively guide the analysis process, both in the form of the human program developer, as well as the software system itself. In the field of program comprehension, where it is unreasonable to expect

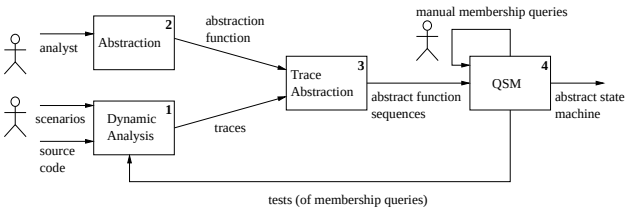

Figure 3. Combining dynamic analysis with QSM

the developer to know much about the system, queries about the system can be formulated as system tests instead (i.e. "Is the sequence of calls *xyz* possible?"). This removes the necessity that the initial set of traces is complete, because the set of samples can be grown iteratively, by guiding the oracle(s) to providing the further missing samples.

4. Small Example: Reverse Engineering State Machines

The authors are currently applying the principles mentioned above to the process of reverse engineering state machines from software [19]. The ability to reliably reverse engineer a state machine from a software system is particularly valuable for a range of development and maintenenace tasks such as model-based testing, documentation, and comprehension of system behaviour. The aim is to generate the necessary program traces to produce a state machine that is as accurate as possible.

The problem of reverse engineering state machines by dynamic analysis is an obvious application for regular grammar inference techniques because the target model is a state machine in each case. Our approach is based upon Dupont's QSM grammar inference technique [6, 9]. QSM espouses the three features mentioned in the previous section; it can accept negative samples, is relatively accurate when provided with incomplete samples, and is active (i.e. will try to obtain more information if it is missing from the initial set of input samples).

Figure 3 shows how we integrate QSM into the reverse engineering process. Essentially, the developer develops a set of mappings from sequences of particular method invocations in the traces to abstract functions. Thus a low-level dynamic trace can be lifted to a series of high-level functions (e.g. <open_file, enter_text, save_file>). Each trace is made into a string of abstract functions and fed into the QSM algorithm. If more information is required, it generates a question, in the form of a sequence of abstract functions that might be valid for the hypothesised machine (e.g. <open_file, close_file, enter_text?>). The question can either be answered directly by the user, *or* automatically, i.e. the sequence can be executed as a system test. Either way, if unsuccessful the new negative string is fed back into the QSM algorithm and the process repeats until no further questions are generated.

The benefits of active dynamic analysis are manifold. If queries are answered solely by the developer, the questioning process fosters a greater understanding of how the system works and forces them to consider aspects of system behaviour that they might not have envisaged. If, instead of the developer, the software system itself is used as an oracle (i.e. queries are posed as system tests), it is straightforward to simply supply new traces that do or do not correspond to the questions. Given a suitable test harness this process could be entirely automated, however at the moment our implementation takes manually generated traces.

Our initial evaluation of the technique shows that the approach is reasonably scalable in terms of number of questions generated and the accuracy of the final model. For specific results the reader is referred to our paper [19]. What is important to point out in the context of this paper however is the fact that the grammar inference evaluation methods provide a well established means to establish how accurate and scalable a given technique is. Grammar inference techniques are usually evaluated by generating a collection of random machines and an accompanying set of random paths across these machines. The technique in question can be run with respect to these different sets of paths of varying degrees of sparsity. This enables the precise quantification of the accuracy and scalability of the technique, with respect to increasingly populated samples of inputs. This evaluation approach is more systematic and accurate than common evaluations of dynamic analysis techniques, which are rarely evaluated with respect to the accuracy or completeness of the final model.

5. Conclusions

Many of the problems faced by dynamic analysis and grammar inference are similar. In order to construct a sound model of system behaviour, current dynamic analysis techniques place unrealistic requirements on the initial set of traces which, when unsatisfied, result in a model that can be highly inaccurate. However, by recasting the problem of inferring a model as a grammar inference problem, there are a number of straightforward techniques that can be adopted to substantially improve the efficiency and reliability of dynamic analysis results.

The domain of software analysis presents a number of advantages for inference techniques that need to be investigated. Static analysis presents an abundance of reliable negative information. Whereas grammar inference traditionally relies solely on human oracles, software engineering tools such as automated testing frameworks and model

checkers can be used as oracles alongside the developer to answer queries about hypothetical models. We have applied grammar inference techniques to the problem of reverse-engineering abstract state machines, and are currently investigating the use of static analysis (in the form of call graphs) to increase the amount of negative information about sequences of methods that are definitely impossible, in order to improve the efficiency of our current technique.

References

[1] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[3] A. W. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 21:592–597, 1972.

[4] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[5] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[6] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde. Generating annotated behavior models from end-user scenarios. *IEEE Transactions on Software Engineering*, 31(12):1056–1073, 2005.

[7] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[8] P. Dupont. Incremental regular inference. In *International Colloquium on Grammatical Inference and Applications (ICGI'06)*, 1996.

[9] P. Dupont, B. Lambeau, C. Damas, and A. van Lamsweerde. The QSM algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 2007. to appear.

[10] M. Ernst. Static and Dynamic Analysis: Synergy and Duality. In *Proceedings of the International Workshop on Dynamic Analysis (WODA'03)*, 2003.

[11] E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[12] E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[13] M. Kearns and L.Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41, 1994.

[14] K. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *COLT*, pages 45–52, 1992.

[15] K. Lang, B. Pearlmutter, and R. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference; 4th International Colloquium, ICGI-98*, volume 1433 of *LNCS/LNAI*, pages 1–12. Springer, 1998.

[16] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1, pages 49–61. 1992.

[17] R. Parekh and V. Honavar. *The Handbook of Natural Language Processing*, chapter Grammar Inference, Automata Induction and Language Acquisition, pages 727–764. 2000.

[18] B. Trakhtenbrot and Y. Barzdin. *Finite Automata, Behavior and Synthesis*. North Holland, Amsterdam, 1973.

[19] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin. Reverse engineering state machines by interactive grammar inference. In *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE'07)*, Vancouver, October 2007. IEEE.