Science of Computer Programming 96 (2014) 444–459

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A survey of grammatical inference in software engineering

Andrew Stevenson [*], James R. Cordy

School of Computing, Queen's University, Kingston, Ontario, K7L 3N6 Canada

HIGHLIGHTS

- We survey grammatical inference as it relates to software engineering.
- A background on the theory of grammatical inference is provided.
- We explore a variety of applications in software engineering.
- These include programming languages, DSLs, visual languages, and execution traces.

ARTICLE INFO

Article history:
Received 16 March 2013
Received in revised form 6 March 2014
Accepted 5 May 2014
Available online 27 May 2014

Keywords:
Grammatical inference
Software engineering
Grammar induction

ABSTRACT

Grammatical inference – used successfully in a variety of fields such as pattern recognition, computational biology and natural language processing – is the process of automatically inferring a grammar by examining the sentences of an unknown language. Software engineering can also benefit from grammatical inference. Unlike these other fields, which use grammars as a convenient tool to model naturally occurring patterns, software engineering treats grammars as first-class objects typically created and maintained for a specific purpose by human designers. We introduce the theory of grammatical inference and review the state of the art as it relates to software engineering.

1. Introduction

The human brain is extremely adept at seeing patterns by generalizing from specific examples, a process known as inductive reasoning. This is precisely the idea behind grammatical induction, also known as grammatical inference, where the specific examples are sentences and the patterns are grammars. Grammatical inference can be described concisely: "The learning task is to identify a 'correct' grammar for the (unknown) target language, given a finite number of examples of the language" [82].

The main challenge of identifying a language of infinite cardinality from a finite set of examples is knowing when to generalize and when to specialize. Most inference techniques begin with the given sample strings and make a series of generalizations from them. These generalizations are typically accomplished by some form of state-merging (in finite automata), or nonterminal merging (in context-free grammars).

Grammatical inference techniques are used to solve practical problems in a variety of different fields: pattern recognition, computational biology, natural language processing and acquisition, programming language design, data mining, and machine learning. Software engineering, in particular software language engineering, is uniquely qualified to benefit because it treats grammars as first-class objects with an intrinsic value rather than simply as a convenient mechanism to model patterns in some other subject of interest.

* Corresponding author.
E-mail addresses: andrews@cs.queensu.ca (A. Stevenson), cordy@cs.queensu.ca (J.R. Cordy).

http://dx.doi.org/10.1016/j.scico.2014.05.008
0167-6423/© 2014 Elsevier B.V. All rights reserved.

Historically there have been two main groups of contributors to the field of grammatical inference: theorists and empiricists. Theorists consider language classes and learning models of varying expressiveness and power, attempting to firm up the boundaries of what is learnable and how efficiently it can be learned, whereas empiricists start with a practical problem and, by solving it, find that they have made a contribution to grammatical inference research.

Unfortunately, research results by one group are not always easily applied by the other. Empiricists often use knowledge specific to their problem domain to help the inference, knowledge that does not exist in the general case. Likewise, theorists rarely try their algorithms on realistic software engineering cases, for example inferring a production-level programming language from millions of lines of code. This makes it difficult for software engineers to determine if a theoretical result is suitable for their particular problem.

Grammatical inference is, intuitively as well as provably, a difficult problem to solve. The precise difficulty of a particular inference problem is dictated by two things: the complexity of the target language, and the information available to the inference algorithm about the target language. Naturally, simpler languages and more information both lead to easier inference problems. Most of the theoretical literature in this field investigates some specific combination of language class and learning model, and presents results for that combination.

In Section 2 we describe different learning models along with the type of information they make available to the inference algorithm. In Section 3 we explore the learnability, decidability, and computational complexity of different learning models applied to language classes of interest in software engineering: finite state machines and context-free grammars. Section 4 discusses the relationship between theoretical and empirical approaches, and gives several practical examples of grammatical inference in software engineering. In Section 5 we list the related surveys, bibliographies, and commentaries on the field of grammatical inference and briefly mention the emphasis of each. Finally, in Section 6 we discuss the main challenges currently facing software engineers trying to adopt grammatical inference techniques, and suggest future research directions to address these challenges.

This survey builds on our previous overview [85] by expanding Section 2 and Section 4, in particular the inference of grammars from execution traces. We also include a discussion on the benefits of grammatical inference to other grammar-based systems in Section 6.

2. Learning models

The type of learning model used by an inference method is fundamental when investigating the theoretical limitations of an inference problem. This section covers the main learning models used in grammatical inference and discusses their strengths and weaknesses.

Section 2.1 describes *identification in the limit*, a learning model which allows the inference algorithm to converge on the target grammar given a sufficiently large quantity of sample strings. Section 2.2 introduces a teacher who knows the target language and can answer particular types of queries from the learner. This learning model is, in many cases, more powerful than learning from sample strings alone. Section 2.3 discusses the PAC learning model, an elegant method that attempts to find an optimal compromise between accuracy and certainty. Finally, Section 2.4 describes how artificial neural networks can be used to learn DFAs from training data. Different aspects of these learning models can be combined and should not be thought of as mutually exclusive.

2.1. Identification in the limit

The field of grammatical inference began in earnest with Gold's 1967 paper, titled "Language Identification in the Limit" [35]. This learning model provides the inference algorithm with a sequence of strings one at a time, collectively known as a *presentation*. There are two types of presentation: positive presentation, where the strings in the sequence are in the target language; and complete presentation, where the sequence also contains strings that are not in the target language and are marked as such. After seeing each string the inference algorithm can hypothesize a new grammar that satisfies all of the strings seen so far, i.e. a grammar that generates all the positive examples and none of the negative examples. While the term "presentation" implies a sequence of strings presented to the inference algorithm, the more general term "information" is also seen in the literature and does not imply any particular input format. Positive, negative, and complete information refers to strings in the target language, strings not in the target language, and both, respectively.

The more samples that are presented to the inference algorithm the better it can approximate the target language, until eventually it will converge on the target language exactly. Gold showed that an inference algorithm can identify an unknown language in the limit from complete information in a finite number of steps. However, the inference algorithm will not know when it has correctly identified the language because there is always the possibility the next sample it sees will invalidate its latest hypothesis.

Positive information alone is much less powerful, and Gold showed that any superfinite class of languages cannot be identified in the limit from positive presentation. A superfinite class of languages is a class that contains all finite languages and at least one infinite language. The regular languages are a superfinite class, indicating that even the simplest language class in Chomsky's hierarchy of languages is not learnable from positive information alone.

There has been much research devoted to learning from positive information because the availability of negative examples is rare in practice. However, the difficulty of learning from positive data is in the risk of overgeneralization, learning a