Evaluation and Comparison of Inferred Regular Grammars *

Neil Walkinshaw, Kirill Bogdanov, and Ken Johnson

The Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello, S1 4DP Sheffield, U.K.
{n.walkinshaw, k.bogdanov, csken}@dcs.shef.ac.uk

Abstract. The accuracy of an inferred grammar is commonly computed by measuring the percentage of sequences that are correctly classified from a random sample of sequences produced by the target grammar. This approach is problematic because (a) it is unlikely that a random sample of sequences will adequately test the grammar and (b) the use of a single probability value provides little insight into the extent to which a grammar is (in-)accurate. This paper addresses these two problems by proposing the use of established model-based testing techniques from the field of software engineering to systematically generate test sets, along with the use of the Precision and Recall measure from the field of information retrieval to concisely represent the accuracy of the inferred machine.
Keywords: Evaluation, State-Merging, Model-Based Testing, Precision and Recall, FSM Testing

1 Introduction

Inferring an unknown regular grammar from a sample of valid and invalid sentences is a well-established problem [13]. In practice it is often difficult to collect a representative sample of valid and invalid sentences that suitably encapsulates every required behavior of the target grammar. This has spurred the development of inductive approaches [19, 16], which can produce a reasonable inferred grammar even if the provided set of sentences is only a sparse sample.

The ability to reliably measure the accuracy of an inferred grammar is fundamental to the evaluation of a technique as a whole. Conventionally, the accuracy of a grammar is evaluated by generating a random 'test' sample from the target grammar, and counting the proportion of tests that are correctly classified by the inferred grammar.

In this paper we point out that the standard evaluation techniques of regular grammars can present a skewed perspective of their accuracy. We outline the two main challenges and propose the use of software engineering and information retrieval methods to tackle them.

1. Generating a representative test sample: Certain aspects of the grammar (a) can be difficult to exercise with random tests, but (b) represent key

* This work has been funded by the AutoAbstract EPSRC grant EP/C511883/1.

language features. To address this problem we propose the use of test-generation techniques from the field of model-based testing [17] in software-engineering. Many conformance testing algorithms have been designed to establish with certainty if an implementation of a software system conforms to a known specification of the system. Both implementation and specification are modeled by deterministic finite automata, and so it becomes straightforward to apply these techniques for the sake of establishing the accuracy of inferred grammars.

2. Measuring the accuracy of the inferred grammar: Conventionally, an inferred grammar is evaluated by measuring the proportion of correctly classified sentences. However, this provides little insight into grammar properties that are of interest such as exactness or completeness.

Instead of the traditional single value to quantify machine accuracy, this paper illustrates the use of precision and recall [24], a well understood measure from the field of information retrieval. This can be visualised in an accessible manner, and can be used to establish to what extent the approach over / under generalises. We show how this correlates with Dupont's lattice-based representation [10] of the regular inference search space.

The format of the paper is as follows. Section 2 introduces the regular inference problem, and shows how inferred grammars are conventionally evaluated. Section 3 describes state machine testing techniques. Precision and recall are described in Section 4 with their relation to an established lattice-based inference search space shown in Section 5. Section 6 contains a small case study. Related work and conclusions can be found in Sections 7 and 8 respectively.

2 Regular Inference and the Evaluation of Inferred Grammars

This section provides a brief background to the challenge of regular inference, followed by a description of the conventional evaluation approach along with its pitfalls. First, we list some basic definitions and set our notation based on Dupont *et al.* [9, 10]. A deterministic finite automaton (DFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is a partial function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting states. We say a sequence $s \in \Sigma^*$ is *accepted* by A if there is a path from the initial state q_0 to an accepting state q. In symbols, $q_0 \xrightarrow{s} q$, for $q \in F$. Throughout this paper, we only consider a special type of DFA where all states are accepting states. A labelled transition system (LTS) A is a DFA whenever $F = Q$. The consequence of this is that for any sequence accepted by an LTS, all prefixes of that sequence are also accepted. That is, every LTS is *prefix closed*.

A *grammar* is a set of rules which specify a subset L, called a *language*, from the set Σ^* of all possible sequences of characters in Σ. We use deterministic finite automata to represent regular grammars and write $A(L)$ to denote a DFA A which produces the language L. When referring to two or more languages, we denote the language produced by the DFA A as L_A; when no ambiguities arise, the subscript is omitted. We define the concatenation $L_A L_{A'}$ of two languages

L_A and $L_{A'}$ to be the set $\{ls \mid l \in L_A,\ s \in L_{A'}\}$. For a state $q \in Q$ define $L_A(q) = \{s \in \Sigma^* \mid q \xrightarrow{s} q_f \text{ for some } q_f \in Q\}$ to be the language accepted by A in q.

Given sample sets of *valid* sequences which are in L and optionally *invalid* sequences that are not in L, the *regular grammar inference problem* is to identify a regular grammar which defines L, using the samples. That is, given $S^+ \subseteq L$ and S^- such that $S^- \cap L = \emptyset$, construct a DFA $A(L) = (Q, \Sigma, \delta, q_0, F)$ where for each $l \in L$ there exists a $q \in Q$ such that $q_0 \xrightarrow{l} q$, and for all $s \in S^-$ there is no such a path.

The DFA A is a prefix tree acceptor (PTA) [20, 22] of S^+ if each sequence in S^+ has a unique path from the start state to an accepting state, with common prefixes sharing the same path. If a set S^- is available, it is trivial to augment the PTA to include them, preventing false merges (below) from occurring. This is conventionally referred to as the *augmented prefix tree acceptor* (APTA).

State merging techniques take a PTA or APTA as input, and proceed to merge states that are deemed to be equivalent. Ultimately they aim to converge on the most general machine that is consistent with the given samples. We recall a partition π maps a set Q to a disjoint family of subsets whose union is Q. For $q \in Q$, $\pi(q)$ is the unique subset containing q; when states are merged, they are placed in the same subset $\pi(q)$. Let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA. A *quotient automaton* [10] $A/\pi = (Q_{A/\pi}, \Sigma, \delta_{A/\pi}, q_{A/\pi}, F_{A/\pi})$ is derived from A with respect to a partition π as follows. The set of states $Q_{A/\pi}$ is defined as $Q_{A/\pi} = \{\pi(q) \mid q \in Q_A\}$ and the set of final states $F_{A/\pi}$ as $F_{A/\pi} = \{\pi(q) \mid q \in F_A\}$. Let $Q, Q' \in \pi(Q_A)$ and $a \in \Sigma$. Define the transition function $\delta_{A/\pi} : Q_{A/\pi} \times \Sigma \to Q_{A/\pi}$ as $\delta_{A/\pi}(Q, a) = Q'$ if, and only if, there exists $q \in Q$ and $q' \in Q'$ such that $\delta_A(q, a) = q'$, for $a \in \Sigma$. The initial state $q_{A/\pi}$ is the set $q_{A/\pi} = \pi(q_A)$. As a consequence of state merging, any path that is accepted in A is also accepted in A/π. In fact, A/π is a generalization of A where $L_A \subseteq L_{A/\pi}$.

Given a DFA A, the set S^+ is *structurally complete* with respect to A if it covers every transition in A. If one uses a PTA built from such an S^+ and there is sufficient information to prevent an incorrect merge from occurring, the state merging process can be guaranteed to correctly converge at $A(L)$. In other words, then there exists a partitioning π such that the quotient automaton $PTA/\pi = A(L)$ [10]. This requirement is however unrealistic for most real-world applications; if the target machine is unknown it is often difficult to guarantee structural completeness, and if the target automaton is nontrivial, a complete set of samples can simply be too large to obtain or difficult to identify. In practice, techniques need to be able to infer fairly accurate grammars given only sparse samples. Spurred on by several competitions, a number of promising state-merging techniques have emerged [16, 9]. The ability of these approaches to infer reasonably accurate grammars from sparse samples, coupled with the scalability of these techniques, renders them particularly appealing for many practical applications.

Grammar inference techniques are evaluated in terms of their accuracy at classifying a test sample of sequences [15, 16, 4, 9]. The set of test sequences (re-

ferred to as a *test set*) is usually compiled by tracing a selection of random paths over the target machine, ensuring that they are evenly split between sequences that should be rejected and accepted, as well as ensuring that their lengths fit a uniform distribution. The accuracy is then measured as the proportion of test sequences that are correctly classified as either accepted or rejected by the inferred grammar.

This approach to evaluation is however problematic for two reasons. The validity of the accuracy metric is entirely dependent upon a test set that is representative of the target machine, and this is often improbable if the test set generation process is essentially random. The second reason is that, even if a representative test set is found, a single value provides very few insights into what might make one approach to grammar inference superior / inferior to another. These two problems are elaborated below.

Obtaining a test set that is 'representative' of the target grammar is very challenging. Usually, the language of the target machine will contain an infinite number of possible valid (and invalid) sequences, but it is not sufficient to simply pick a random sample for the sake of testing. Grammars that correspond to a large DFA with a large alphabet have a low observability: certain aspects are much less likely to appear in a random sequence than others. Bongard and Lipson [4] use the example of the Tomita 1 language [21], that only produces a positive classification for a binary string 2.4% of the time. As demonstrated in Lang's experiments with random DFAs [15], the size of a random test set that approximately infers a grammar invariably has to increase by orders of magnitude as the size of the target machine increases.

Alongside the problem of identifying a suitable test set, there is also the problem of how to interpret the final accuracy result - a single value provides very few qualitative insights into the resulting grammar. Assuming, for example, that we can confidently assert that a DFA has an accuracy of 70%. Does this mean that it is more likely to falsely classify a string that should be accepted or rejected? If we wanted to improve its accuracy, would we need to make it more general or more specific?

3 Model-Based Test Generation

The challenge of identifying a test set that reliably covers every behavior of some specification DFA is nontrivial. Random sets of strings may easily explore specific aspects of the machine much more thoroughly than others, and hence may result in a skewed accuracy measurement. How do we identify a finite set of strings that can be used to evaluate the accuracy of the hypothesis machine?

This problem has been considered in the area of model-based software testing where the focus is to check by testing whether an implementation is similar to a model. Testing uses a model (reflecting the intended behavior) and generates test sequences from it. If an implementation produces a different result to a model on any of those sequences, such an implementation is considered faulty; otherwise, one may wish to have a confidence that an implementation is similar

to a model. The extent of such a similarity depends on the specific testing method used to generate test sequences and on the properties of both a model and an implementation. In the context of this paper, the aim is to compare two DFAs, the hypothesis and the target. For this reason, the focus is on testing methods which can demonstrate an equivalence between languages accepted by the two DFAs rather than, for instance, whether one language contains another one.

The problem of checking whether two DFAs are accepting the same language by experiment cannot be solved in the most general case — since a test set has to be finite, it is always possible for an implementation DFA to contain more states than could be explored with a chosen test set and those extra states may have undesired behavior. For this reason, all DFA testing methods assume that it is possible to estimate the maximal number of states in an implementation, in advance. In addition, an alphabet of an implementation is usually assumed to be known. Finally, both machines are expected to be deterministic, minimal and feature a reliable reset (a special input which brings them to their respective initial states, not usually shown on a transition diagram); this simplifies testing and holds in the case considered in this paper. State-based testing methods systematically explore the (unknown) transition structure of an implementation DFA, comparing it to the model. From every state which is included in a model, they attempt every symbol in an alphabet. This verifies that all transitions absent in a model are also absent in an implementation; symbols which label transitions in a model are followed with specific sequences to verify target states of those transitions. This way, every state and every transition in a model are checked in an implementation. Target state verification is based on an assumption that all states in a model are different, i.e. they accept different languages. For this reason, for every pair of states, it is possible to choose a sequence which distinguishes between them and a set of such sequences (called a *characterisation set* below) can be used to check that an implementation has reached the expected states. The testing method used in this paper is the implementation by the authors of the original Vasilevski/Chow W-Method [6].

Construction of a test set using the W-Method is briefly described following [3]. For an implementation DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a model DFA S, the W-Method constructs $Y \subseteq \Sigma^*$ such that $(L_A \cap Y = L_S \cap Y) \Rightarrow L_A = L_S$. Such a set is called a *test set* of A. A *state cover* C is a prefix-closed subset of Σ^* containing all sequences of inputs needed to visit every state of a DFA from the initial state; in symbols, $\epsilon \in C$ and for all states $q \in Q \setminus \{q_0\}$ there exists a $c \in C$ such that $\delta(q_0, c) = q$. For a subset $W \subseteq \Sigma^*$ the states $q_1, q_2 \in Q$ are called *W-distinguishable* if $(L_A(q_1) \cap W) \neq (L_A(q_2) \cap W)$. We call W a *characterisation set* [12] of A if any two distinct states of A are W-distinguishable. Given an estimate k as to how many more states an implementation may have compared to a model, a test set $Y = C(\{\epsilon\} \cup \Sigma \cup \cdots \cup \Sigma^{k+1})W$.

There is a clear overlap between the two fields of conformance testing and grammar (DFA) inference, because tests generated from a model can be interpreted as membership queries to be answered by a hidden implementation machine to establish whether the two are equivalent. Several software engineer-

ing researchers have previously explored the relationship between the two areas [2, 14], and the use of conformance testing algorithms for answering equivalence queries posed by Angluin's L^* algorithm.

4 Evaluating Accuracy with Precision and Recall

Grammar inference techniques, and classifiers in general, are commonly evaluated in terms of the probability that they will return a correct response [16, 9, 4]. This measure is often suitable as a coarse summary of classifier behavior, but provides a user with very little insight into any particular strengths / weaknesses of the technique. A single accuracy figure can give no insight into questions such as (a) whether a hypothesis machine over/under generalised and (b) whether a hypothesis language contains too many false positives or negatives.

4.1 Precision and Recall in Grammar Inference

Precision and recall [24] is a more descriptive measure, because it quantifies the similarity of two objects with two variables instead of one – precision (exactness) and recall (completeness). Originally from the domain of information retrieval, it is used to measure the overlap between what has been retrieved and what is relevant.

The conventional precision-recall evaluation process works by trying to establish the "overlap" between an inferred model and its target. This is achieved by computing random samples from the inferred and target models, and adding sequences to the RET and REL sets depending on how they are classified. This classification is illustrated in the table below; if a sequence (from either machine) is accepted by both machines, it is added to both RET and REL sets, if the string is accepted only by the inferred machine, then it is added to RET etc. The final RET and REL sets are then used to compute precision and recall as follows: precision is computed by $\frac{|REL \cap RET|}{|RET|}$ and recall by $\frac{|REL \cap RET|}{|REL|}$. Precision and recall have been used in the past to measure the accuracy of inferred models in both the domains of software engineering and grammar inference. As an example, in software engineering, Lo *et al.* [18] use it to establish the accuracy of reverse-engineered software specifications. In grammar inference, Tu and Hanovar [23] use it to measure the accuracy of their context-free grammar inference technique. The two variables reflect complementary aspects of the inferred model, and are more descriptive as a result, helping to answer the above questions that arise with the use of a single "accuracy" measure.

H Machine (Hypothesis)	S Machine (Specification)	RET	REL
accept	accept	×	×
accept	reject	×	
reject	accept		×
reject	reject		

Unfortunately, obtaining reliable precision and recall scores when comparing DFAs is not straightforward. The conventional approach has two flaws that can undermine confidence in the results:

1. Sampling: It relies on the assumption that the random positive samples computed for each of the machines are thorough enough to capture the differences between the two machines. This approach will only identify disagreements between the machines that are easy to reach with random sequences, and will ignore those that are less likely to be exercised. There is also the danger that the sample will simply represent the training sample, which is often also a random sample from the target machine. Ultimately, any measure of accuracy that is computed from samples that are constructed in this way is at best indicative of the correspondence between two machines, and risks being misleading.

2. Measuring: The computation of RET and REL sets (as shown in the table above) is biased towards the accepting behavior of the two machines. As noted above, conventional samples do not include invalid sequences, but even if they did, they would not be accounted for according to the scheme in the table above. If both machines correctly reject a sequence, this would not be incorporated into the computation of precision and recall. In the context of grammar inference, it is as important to evaluate an inferred grammar in terms of the sequences it rejects as well as the sequences it accepts. Hence, approaches that do not use precision and recall (using the conventional single-valued approach) [9] tend to ensure that test samples are evenly split into sets of valid and invalid sequences.

A naive solution to the two problems would be to (a) evenly construct a sample from both valid and invalid sequences, and (b) add invalid sequences to both *RET* and *REL* when the two machines are in agreement about their rejection. This will however still bias the results, because the sampling emphasises those parts of the machine that are easy to reach. There is also the problem that the split between valid and invalid sequences is in effect arbitrary; an even split is making the unlikely assumption that the language of the machine is evenly balanced in terms of its valid and invalid sequences.

4.2 Authoritative Measurement of Precision and Recall by Conformance Testing

One apparent solution to the sampling problem mentioned above is to apply conformance testing techniques (see Section 3). For a given DFA, techniques such as the W-Method will produce a finite set of sequences that is not overly biased towards any particular part of the machine. The conventional approach discussed above generates a composite set of random samples from both the inferred and the target machine, in the hope that this will highlight the differences between the two machines. Instead, using model-based testing techniques, it is only necessary to generate one test set from the target machine. This is guaranteed to highlight every discrepency between the two machines.

Although test sets that are generated by techniques such as the W-Method are comprehensive, they do have two characteristics that should be noted when applied for the sake of evaluating inferred grammars:

1. Scale: The test set is usually very large. With software and hardware systems, depending on the latency of the system under test, each test execution incurs a cost that can in practice render the execution of a complete test set infeasible. Large test sets are much less of an issue when they are used to evaluate the accuracy of hypothesis grammars. The time taken by test execution is reduced by many orders of magnitude if it merely consists of traversing a path in a graph that is stored in memory. For this reason, this work does not use more efficient testing methods such as Wp and HSI which use a subset of distinguishing sequences depending on a state to be checked (although these methods could just as well be applied).

2. Partial inclusion of training set: The test set invariably incorporates a proportion of the training set, which could be seen to bias the results. A certain overlap between the training and test set is inevitable – the prefixes of invalid sequences are valid ones, many of which are a necessary part of a rigorous test set. This overlap can be problematic when the training and test set are both sampled from the same distribution of sequences in the target language; in this context test sequences are meant to test how well the classifier generalises from the training set, and this does not happen if training sequences appear in the test set. However, by generating the test set with a test set generation algorithm, the test set is no longer testing the generalisation of the classifier, but instead serves to produce an absolute measure of difference between the inferred and the target grammars.

Although conformance testing solves the sampling problem, there still remains the challenge of using these tests to accurately measure precision and recall. As mentioned earlier, the conventional approach used to work out the values of RET and REL relies on the assumption that the set of samples contains an equal number of valid and invalid sequences. However, in the case of the W-Method, the vast majority of the tests test for invalid behavior (making sure that the machine does not have extra transitions). This is a property of models of software where an alphabet is a set of commands and from each state only a small subset of those commands can be executed. Numerous invalid sequences can result in skewed precision and recall results: even if the inferred machine does not accept any of the sequences it should, if it correctly rejects most of the invalid sequences, the precision and recall will be disproportionately high.

This problem is addressed by refining the conventional scoring approach to distinguish between accepting and rejecting behavior. Instead of computing a single precision and recall pair, we compute one that describes the accuracy of the hypothesis machine H in terms of the set of traces it should accept, and the other in terms of the set of traces it should reject. For this reason, we divide RET and REL into RET^+, RET^-, REL^+ and REL^-. Test sequences can thus be categorised according to the table below.

H Machine (Hypothesis)	S Machine (Specification)	RET^+	REL^+	RET^-	REL^-
accept	accept	×	×		
accept	reject	×			×
reject	accept		×	×	
reject	reject			×	×

If a sequence is accepted by H but should in fact be rejected, it is added to RET^+ and REL^- as shown in Row 2. Thus, $precision^+ = \frac{|REL^+ \cap RET^+|}{|RET^+|}$ and $recall^+ = \frac{|REL^+ \cap RET^+|}{|REL^+|}$, and the same approach is used to compute the negative precision and recall from RET^- and REL^-. The above definitions of positive and negative precision and recall can be interpreted as follows: $precision^+$: high value means that the positive sequences represented by the hypothesis machine are largely correct; $recall^+$: high value means that the set of positive sequences represented by the hypothesis machine is largely complete; $precision^-$: high value means that the negative sequences represented by the hypothesis machine are largely correct; $recall^-$: high value means that the set of negative sequences represented by the hypothesis machine is largely complete.

5 Relationship between Precision, Recall and the State-Merging Search Space

Precision and recall are more suitable for characterising the success of state merging sequences than the traditional single-valued accuracy approach. By adopting the precision and recall metrics introduced in Section 4.1, it is possible to determine with greater certainty whether a particular merge improves machine accuracy or not. This is best illustrated with the lattice-based characterisation of the state-merging search space.

Inference techniques invariably involve searching a potentially very large space of hypotheses in order to arrive at some result. The representation of this space is key to their efficiency. A representation might, for example, indicate that the selection of one hypothesis would rule out the subsequent selection of another (potentially more suitable) hypothesis. This sort of information can be very valuable during the inference process.

In regular inference hypotheses are commonly related to each other in terms of their respective generality [1]. A grammar B is more general than A if B is the quotient of a merge (Section 2) in A. In symbols, let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA and $P(A)$ denote the set of all partitions of the state space Q_A of A. We define a partial order on $P(A)$, as described in [10]: let $\pi_1 \in P(A)$ such that $\pi_1(Q_A) = \{Q_1, \ldots, Q_r\}$. Define $\pi_2 = \{Q_j \cup Q_k\} \cup \pi_1 / \{Q_j, Q_k\}$, $1 \leq j, k \leq r, j \neq k$. We say that π_2 *derives* from π_1, denoted $\pi_2 \geq \pi_1$. This derivation operation on partitions defines a partial ordering on the set $P(A)$.

A useful metric for the evaluation of a hypothesis machine should also be useful as a guide for the search process. In terms of the lattice search-space (Figure 1), it should be possible to tell whether a given solution is too general or too specific, to direct the search. From this perspective the traditional single-

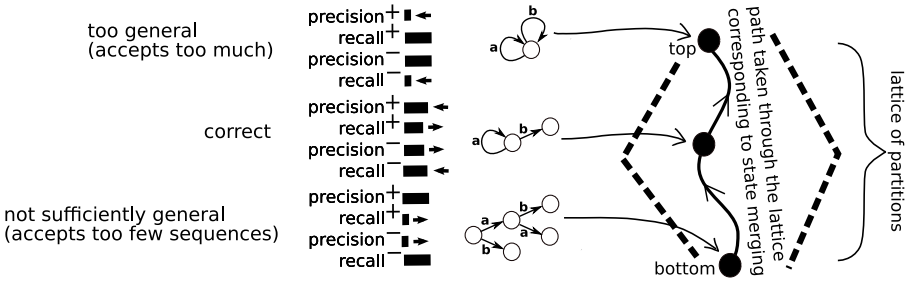

Fig. 1. Lattice-based search space

valued accuracy score can be highly misleading. For example, given a test set where half of the sequences should be valid, and half invalid, the universal DFA at the top of the lattice would result in a score of about 50% (despite the fact that the machine is grossly overgeneralised). Conversely, it is possible that the PTA at the bottom of the lattice produces a similar score, by correctly rejecting most of the negative tests, but also incorrectly rejecting most of the positive tests. During the merging process, the score could fluctuate, but not necessarily provide any guidance — there is thus no relationship between a path through the search-space, and a definite increase or decrease in accuracy.

When adopting precision and recall, there is a more direct relationship with the search-space. In the figure above we depict values of precision and recall using bars of different width. The structurally complete PTA at the bottom of the figure is the starting point of the merging process. In the course of merging states (reflected by a path through the lattice of partitions of states of the initial PTA) one aims to increase $Recall^+$ and $Precision^-$ without compromising $Precision^+$ and $Recall^-$. If the process of generalisation goes too far, the overly-general outcome has low positive recall and negative precision, but the high positive precision and negative recall.

For any pair of hypothesis π_A, π_B in the lattice where $\pi_B \geq \pi_A$, we can state the following: $Precision^+(\pi_B) \leq Precision^+(\pi_A)$, $Recall^+(\pi_B) \geq Recall^+(\pi_A)$, $Precision^-(\pi_B) \geq Precision^-(\pi_A)$, and $Recall^-(\pi_B) \leq Recall^-(\pi_A)$.

6 Case Study: Effect of DFA Generalisation on Accuracy

In this section we use a small case study to illustrate how the precision and recall measures, paired with a model-based test set generation strategy can provide more accurate and detailed quantitative insights into the performance of inference algorithms. The purpose of this case study is to illustrate the utility of our proposed evaluation technique (as opposed to the efficacy of any specific state-merging algorithms). Although the case study subject is quite specific, it is probable that the extra insights garnered from combining precision and recall with model-based test sets would apply to most evaluations of regular inference techniques.

As a basis for the case study, we compare the performance of two variants of the well-known EDSM merging algorithm [16] at inferring a randomly generated grammar. The EDSM algorithm selects suitable state merges by assigning a score for every pair of states (we use the Blue-Fringe algorithm [16] to select these pairs). The first variant merges states where the score ≥ 1, and the second one merges states where the score ≥ 2. The target grammar is randomly generated; its DFA has 50 states, and has an alphabet of 100[1].

The training set is generated in the traditional way by tracing a selection of random paths across the target machine. We chose a small sample size to emphasise the performance of the algorithms with respect to sparse samples. The sample is composed of 50 valid and 50 invalid sequences. The length of each path is a random number between 2 and $n + 5$, where n is the diameter of the target DFA. For each variant of the inference algorithm, the accuracy was charted for every iteration (state-merge), using both the traditional single-valued measurement, and the precision and recall approach proposed in this paper. The results are shown in Figure 2, with the left chart showing results for the case of merge threshold ≥ 1, middle chart showing the results for merge threshold ≥ 2. The single-value accuracy (black line) was established by using a test set consisting of random traces over the target grammar, whereas the test set for the precision and recall scores was generated using the W-Method.

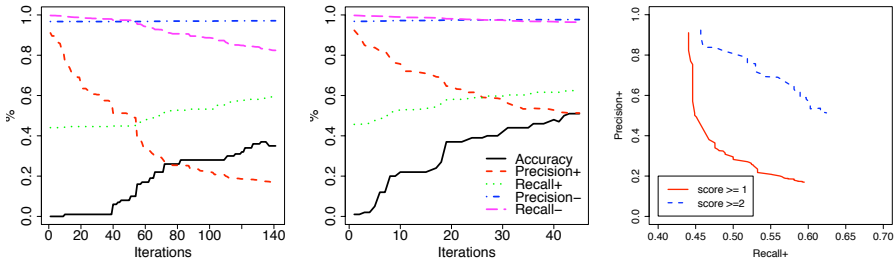

Fig. 2. Traditional Accuracy vs. Precision-Recall measurements at each iteration of a state merging algorithm, for two variants of the EDSM algorithm.

In both charts, the single-valued accuracy score has a similar shape. It starts at zero, and finshes at a score of about 40-50% accuracy. With a higher merge threshold it is slightly steeper, suggesting that it produces more accurate results early-on in the state-merging process. In isolation, the single-valued accuracy line in both charts suggests that the state merging process is moving towards

[1] A GraphML file containing the target DFA can be downloaded from `http://www.dcs.shef.ac.uk/~nw/Files/icgiExample.xml`. The source code for the W-Method and comparison of two DFAs is part of a larger framework developed by the authors, available from `http://statechum.sourceforge.net/`.

an increasingly accurate result. This is however somewhat deceptive. Looking at the precision and recall scores, it becomes apparent that, although the positive recall is increasing, the positive precision is substantially reduced as the merging continues. This is clearest in the left chart around the 58th iteration, a merge happens that reduces the positive precision by about 10%, whereas at the same point the accuracy score increases sharply.

For the sake of illustration, the purpose of this study is to investigate the performance difference in the EDSM algorithm for different thresholds. The single-valued measure gives little insight in this respect. The precision and recall scores on the other hand are much more descriptive. They show that the difference in accuracy between the two versions is due to a combined improvement in positive precision and negative recall. In practice this means that with a lower merging threshold the resulting DFA accepts too many false positives and negatives. On the left chart, the positive precision drops by about 70% throughout the inference process, whereas in the middle chart it only drops by about 40%. The negative recall drops by about 20% in the left chart, it stays around 95% in the second one.

Finally, precision and recall can be visualised together, to provide an overall summary of the accuracy of a particular search technique. For the sake of simplicity, we omit the rejecting behavior in this case, where there is only a small trade off between precision and recall. The rightmost chart in Figure 2 plots the positive precision versus the positive recall for the two EDSM variants, as measured using the W-Method test set. These plots [24] clearly depict the performance increase for the higher score threshold, where the precision is compromised to a much lesser extent as the recall increases.

7 Related Work

The problem of measuring the accuracy of learner hypotheses forms the basis of a substantial amount of discussion in machine learning literature. Receiver Operator Characteristic (ROC) curves have emerged as a useful solution to this problem [5]. These make explicit the relationship between the number of true-positives and false-positives and are closely related to precision and recall. However, if the data set upon which the curve is built is skewed (which is usually the case with grammar inference data sets), precision and recall is a preferable measure [7]. To the best of the authors' knowledge, precision and recall have not been applied in the context of regular grammar inference.

Competitions, most notably the Abbadingo competition [16], have played a major role in driving the development of new inference techniques. These are usually operated by setting up a server that randomly generates a (hidden) target DFA, along with an accompanying random training and test set. Conventionally, a winning inference technique has to be able to produce a hypothesis DFA that produces an accuracy score of 99%. As we have shown, depending on the test set and the target machine, this accuracy score can be misleading. In the context of such a competition, this has the potential to result in the selection of inference

techniques that might not fare as well if the criterion for success was that the resulting machine should produce high precision and recall scores, and if the test set was generated systematically as opposed to randomly.

The lattice-based representation of search-space (and its relation with precision and recall) is particularly relevant to heuristic grammar inference techniques that depend upon a notion of "fitness". As an example, Dupont's GIG method [8] uses a genetic algorithm to search the lattice of automata, where one search result is considered to be "fitter" than another if it has fewer states and misclassifies fewer strings in S^-. If we assume that part of the (unused) training sample can be used as a more complete test set, then it becomes possible to work out the positive and negative precision and recall for each automaton. This measure could then be used as a more fine-grained fitness function and could form a suitable basis for a multi-objective search-based inference algorithm [11].

8 Conclusions

Due to the fact that most target machines are inherently unbalanced, the conventional use of evenly-split random traces in the target machine as test sets is insufficient, and can provide a skewed view of the accuracy of the final machine. The use of a single value to summarise this accuracy is too simplistic, and does not provide enough of an insight into why a particular inference algorithm becomes inaccurate. In this paper we have shown how the use of precision and recall, combined with a systematic test set generation strategy, can be used to evaluate inferred grammars in an authoritative manner.

We distinguish between the precision and recall for rejecting and accepting behavior of the inferred machine because the target machine is usually unbalanced. This provides a more detailed means for evaluating machines and makes it easier to see whether a hypothesis machine has been under or over generalised. We have also shown how this means of evaluation links in with the established lattice-based view of the inference search space.

References

1. D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.
2. T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen. On the correspondence between conformance testing and regular inference. In M. Cerioli et al., editors, *Fundamental Approaches to Software Engineering FASE*, volume 3442 of *LNCS*, pages 175–189. Springer, 2005.
3. K. Bogdanov, M. Holcombe, F. Ipate, L. Seed, and S. Vanak. Testing methods for X-Machines: A review. *Formal Aspects of Computer Science*, 18:3–30, 2006.
4. J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6:1651–1678, 2005.
5. A. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997.

6. T. Chow. Testing Software Design Modelled by Finite State Machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.
7. J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 233–240. ACM, 2006.
8. P. Dupont. Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In *Proceedings of the Second International ICGI Colloqium on Grammatical Inference and Applications*, volume 862, pages 236–245. Springer Verlag, 1994.
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*, 22:77–115, 2008.
10. P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In Rafael C. Carrasco and Jose Oncina, editors, *Proceedings of the Second International ICGI Colloqium on Grammatical Inference and Applications*, volume 862 of *LNAI*, pages 25–37, Berlin, September 1994. Springer Verlag.
11. C. Fonseca and P. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.
12. A. Gill. *Introduction to the Theory of Finite State Machines*. McGraw-Hill, 1962.
13. E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
14. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. *Logic Journal of the IGPL*, 14(5):729–744, 2006.
15. K. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *COLT*, pages 45–52, 1992.
16. K. Lang, B. Pearlmutter, and R. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Proceedings of the International Colloquium on Grammar Inference (ICGI'98)*, volume 1433, pages 1–12, 1998.
17. D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.
18. D. Lo and S. Khoo. QUARK: Empirical assessment of automaton-based specification miners. In *WCRE*, pages 51–60. IEEE Computer Society, 2006.
19. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E. Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, Singapore, 1992.
20. R. Parekh and V. Honavar. *Grammar Inference, Automata Induction, and Language Acquisition*, chapter 29. Marcel Dekker, USA, 2000.
21. M. Tomita. Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108, Ann Arbor, Mi, 1982.
22. B. Trakhtenbrot and Y. Barzdin. *Finite Automata, Behavior and Synthesis*. North Holland, Amsterdam, 1973.
23. K. Tu and V. Honavar. Unsupervised learning of probabilistic context-free grammar using iterative biclustering. Technical Report 00000572, Dept. Computer Science, Iowa State University, May 2008.
24. C.J. van Rijsbergen. *Information Retrieval*. Butterworth-Heineman Newton, 1979.